AN EXPERT SYSTEM FOR

SOLVENT-BASED SEPARATION PROCESS SYNTHESIS

by

Jean-Christophe BRUNET

Thesis submitted to the Graduate Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Chemical Engineering

APPROVED:

Y.A. Liu, Chairman

Henry A. McGee, Jr.

Peter R. Rony

April, 1992

Blacksburg, Virginia

ABSTRACT

AN EXPERT SYSTEM FOR SOLVENT-BASED SEPARATION PROCESS SYNTHESIS

Expert systems are being used more daily in chemical engineering. This work continues the development of an EXpert system for SEParation flowsheet synthesis named EXSEP. Written in Prolog, it can generate flowsheets for four multicomponent separations: distillation, absorption, stripping and liquid-liquid extraction. For these separations, we describe a large collection of heuristics (or rules) that are used for flowsheet synthesis. EXSEP uses several of these heuristics and the Kremser equation to test the thermodynamic feasibility of separation tasks. EXSEP requires only basic input data such as the expected component flow rates in each product and the component K-values. With those data, EXSEP searches for the sets of the number of theoretical stages, solvent flow rate, and component-recovery ratios that characterize a number of feasible and economical flowsheets. The use of the component assignment matrix (CAM) combined with Prolog list processing makes EXSEP very fast (several seconds) to generate solutions. We test EXSEP with several examples of industrial separation processes and compare the results with the literature. We also compare EXSEP results with rigorous simulations using commercial CAD software (e.g., DESIGN II). In most cases, EXSEP gives very similar and even better flowsheets. However, EXSEP is limited to dilute solvent-based separations and cannot solve problems where the major feed component is also the solvent (e.g. sour-water steam stripper). The development of EXSEP on IBM-PC makes it very "user friendly". In the future, EXSEP should be expanded with additional modules such as extractive and azeotropic distillation, and bulk absorption. It should also include modules for separation method and solvent selections, which are great challenges in flowsheet synthesis.

Abstract

TABLE OF CONTENTS

CHAPTER 1

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

APPENDICES

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACKNOWLEDGEMENTS

My first thought goes to my major advisor, Dr. Y. A. Liu. He introduced me to the area of artificial intelligence and computer-aided design. He spent many hours teaching me "the art of Prolog", and was always available and patient to answer my questions.

It has been a great pleasure to work with him whom I admire as a researcher, a teacher, and an individual.

Thanks go to Thomas Quantrille for all his help and advice. Far more importantly, he had the initial ideas that were used in this thesis.

I thank the members of my graduate committee Dr. Henry McGee and Dr. Peter R. Rony.

I also thank Dr. Bob Mahan of the Department of Mechanical Engineering of Virginia Tech, and Madame Nicole Jaffrin of the U.T.C. (Université de Technologie de Compiègne, France) whose collaboration gave me the great opportunity to come and study at Virginia Tech. Thanks also go to Dr. William Conger, Chemical Engineering Department Head, for his frequent help and support.

Thanks to my future wife, "Master" Sylvaine Chardon, for her support, and to my parents for their encouragement.

NOMENCLATURE

A: Absorption factor (L/KV, dimensionless)

E: Extraction factor (KS/L, or KV/L, dimensionless)

f_i^L, f_i^V : Fugacity of component i in the liquid and vapor phase, respectively.

$f^L_{pure\ i}$, $f^V_{pure\ i}$: Fugacity of pure component i in the liquid and vapor phase, respectively.

Fi : Flow rate of Component i [mol/hr]

K: Average K value of products

K,k: K value, Henry's law constant or distribution coefficient

L : Liquid flow rate, feed or solvent [mol/hr or kg/hr]

m : Slope of the equilibrium line.

m : Mass flow rate [kg/hr]

N : Number of theoretical stages in the column

Nopt: Optimal value of the number of stages

P : Pressure [kPa]

P_i: Partial pressure [kPa]

P_i^s: Saturation (vapor) pressure of pure liquid i [kPa]

S: Stripping factor (KV/L, dimensionless)

V : Vapor flow rate, feed or lean gas [mol/hr or kg/hr]

Vmin: Value of the solvent flow rate for $N=\infty$ [mol/hr]

X: Generic separation factor (can be A,S or E)

xin, xout: molar fraction of a component entering and exiting the liquid phase, respectively

Xlg or Xsl: Maximum separation factor (from heuristics)

Xmin: Minimum separation factor (from heuristics)

Xopt: Optimal separation factor (from heuristics)

Xst: Value of the separation factor when N=Nopt

x,y : molar fraction in the liquid and the vapor phase, respectively

yin, yout: molar fraction of a component entering and exiting the vapor phase, respectively

Abbreviation

AIChE : American Institute of Chemical Engineers

CS: separation factor (dimensionless)

ESA : energy-separating agents

LLE : liquid-liquid extraction

MSA : mass-separating agents

Subscripts, Superscripts and Bracket Notation.

i : ith component

in, out: entering or exiting a separator

[] : reference of predicate in the charts (Chapter 5)

CHAPTER 1

Introduction to EXSEP

1.1 Introduction:

An expert system is a computer program designed to perform the same tasks as a human expert (calculation, decision, expertise). In the current state of research, expert systems represent an aid for human experts or technicians; they do not replace human experts yet.

Expert systems are increasingly common in chemical engineering. Currently, dozens of expert systems are commercially available to solve problems in process control, process design, product design, and other fields (Quantrille and Liu, 1991, p.399). In each AIChE national meeting, ten to twenty researchers present papers concerning expert systems and their applications in chemical engineering.

Expert systems cannot apply to every domains where experts are needed. Before the development of an expert system, the programmer should ask several questions. Is the development possible ? Is the development appropriate ? Is the development justified ? (Quantrille and Liu, 1991, p.401). The main justifications for the development is when human expertise is being lost, or when expertise is needed in many locations.

In the case of separation flowsheet synthesis, there is a need for fast and convenient preliminary design software. Indeed, before running rigorous simulations on mainframe computers with CAD software, the design engineer has to choose a convenient separation and then evaluate

its feasibility. The choice of the separation is usually the task of the expert. The shortcut method used for the feasibility evaluation is often done by hand. An expert system for separation synthesis should perform the choice of the appropriate separation, or at least, it should offer the user a set of separations that can be tested. It should also perform calculations, give preliminary design variables for the initial flowsheets that can be used in rigorous CAD and explain the validity of the results. Today, the development of an expert system for separation synthesis is possible on IBM-PC class machines, which answer the requirements of convenience and availability.

1.2 EXSEP: ESA- and MSA-modules

The idea of an expert system for multicomponent separation-flowsheet developments originated from Dr. Y.A. Liu. Dr. T.E. Quantrille (1991) worked for his PhD degree, on the concepts for this application of expert systems. He wrote the main program of an expert system and a module for ordinary distillation. Distillation uses energy (i.e., an energy-separating agent, ESA) to separate products. Quantrille and Liu named this new expert system EXSEP, which stands for **Ex**pert System for **Sep**aration Synthesis.

This work continues the development of EXSEP applied to other multicomponent separations based on solvents or mass-separating agents (MSAs). EXSEP is now able to generate good initial flowsheets for distillation (using energy-separating agents, or ESA-based) , absorption, stripping and liquid-liquid extraction (using mass-separating agents, or MSA-based). Figure 1.1 shows the component modules of EXSEP. A future development would be to combine both ESA and MSA to solve problems of extractive or azeotropic distillation.

This thesis focuses only on MSA-based separations. For more information about Prolog,

and on multicomponent separation-flowsheet synthesis by EXSEP, we recommend the book of T.E. Quantrille and Y.A. Liu, Artificial Intelligence in Chemical Engineering, Academic Press, San Diego, CA (1991).

After this introduction (Chapter 1), we present, in Chapter 2, the fundamental concepts, and also a collection of heuristics (rules of thumb) used for MSA-based separations. In Chapter 3, we discuss the chemical engineering perspective of EXSEP through the study of examples. Chapter 4 consists of a step-by-step procedure to use each of the three MSA modules of EXSEP (absorption, stripping, and extraction). In Chapter 5, we focus on the AI programming aspect of EXSEP and explain its logic structure. Finally, we summarize our conclusions and recommendations in Chapter 6. In appendices, we give other examples of EXSEP problem-solving, and we present some useful supplementary information.

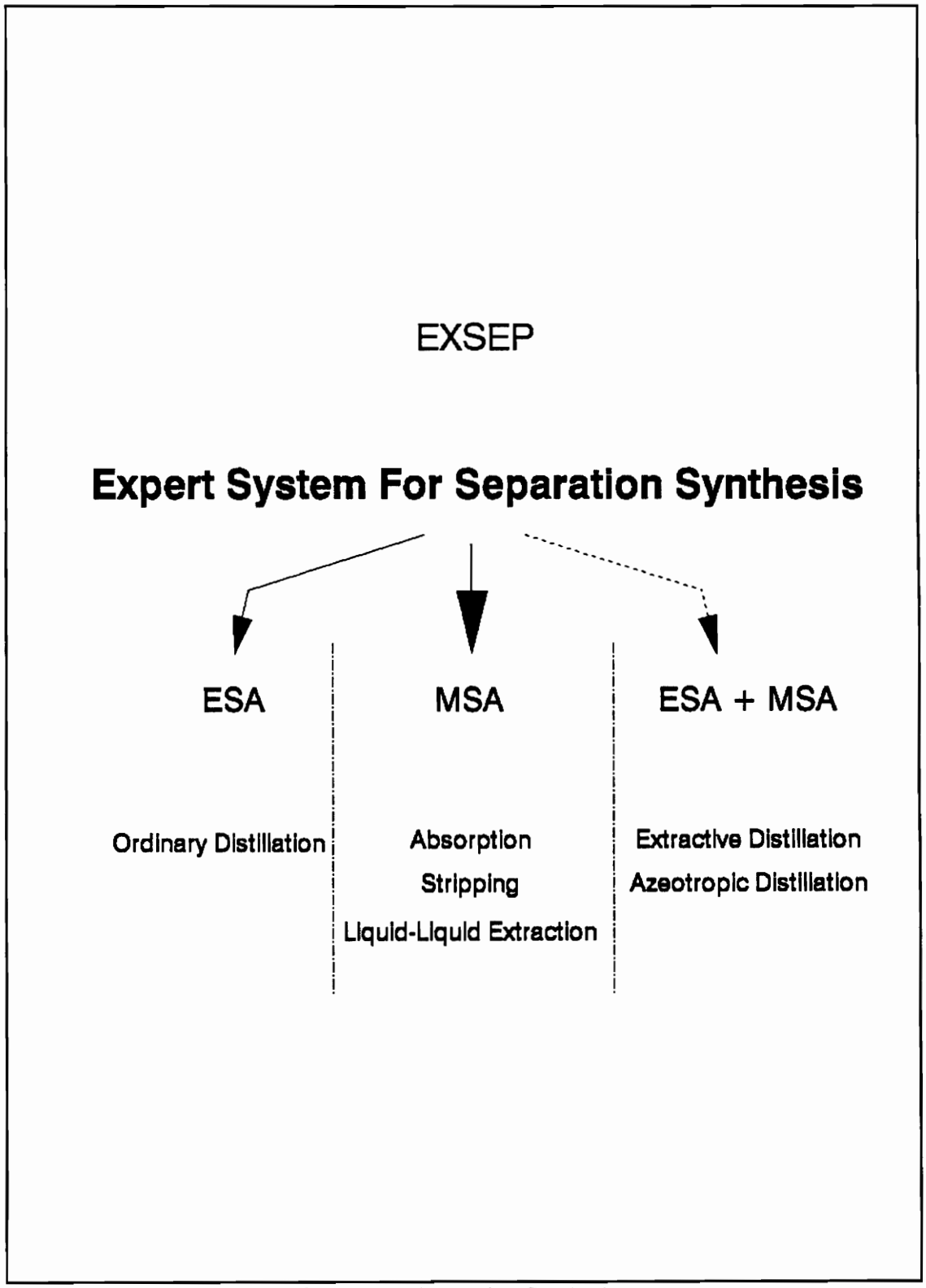


Figure 1.1 : EXSEP (Expert System for Separation Synthesis)

CHAPTER 2

Solvent-Based Multicomponent Separations

Henley and Seader (1981,p.6) state that to separate components of a single-phase mixture (gas, liquid, or solid), we generally add a second phase to make the separation economically realizable. Heat or a chemical species, such as a solvent or an absorbent, can be used to create the second phase. We talk about *energy-separating agents (ESAs)*, or *mass-separating agents (MSAs)*. Some separations like extractive or azeotropic distillation use both ESAs and MSAs.

The viability of solvent-based separations is related to their thermodynamic feasibility. The thermodynamic equilibrium of a mixture is modified by the presence of the solvent. The choice of the solvent is thus important. In section 2.1, we define the K values, which characterize the thermodynamic equilibrium. We introduce also the terms, *key component* and *separation factors*. Kremser developed an equation that links those parameters, and is used to evaluate the feasibility of solvent-based separations. We use the Kremser equation in the expert system, EXSEP, for multicomponent absorption, stripping and liquid-liquid extraction (see sections 2-2 to 2-4). For those three separations, we collected a large list of heuristics that we use for flowsheet synthesis with EXSEP.

2.1 Overview of Solvent-Based Separation Methods

2.1.1 Thermodynamic Equilibrium

When two phases are mixed, components go from one phase to the other until each component concentration reaches a steady-state value. This steady state is called *thermodynamic*

equilibrium. For example, in gas absorption a gas phase of air and ammonia reaches equilibrium in water when a certain amount of ammonia has transferred from the gas phase to the liquid phase. In the same way, equilibrium can occur between two liquid phases (liquid-liquid extraction). For example, a mixture of water and acetic acid in the presence of a liquid phase of butanol is at equilibrium when the total amount of acetic acid is distributed between the water and butanol.

In these two examples, we assume that only one component transfers (ammonia from air/ammonia to water, or acetic acid from water/acetic-acid to butanol). The reality is more complicated, especially with multicomponent mixtures. In multicomponent systems, every product transfers from one phase to the other. The amount transferred can be assumed to be proportional to a constant called the **distribution coefficient** or **K value.** The aim of a separating agent (mass or energy) is to change the K value to make the separation thermodynamically easier. When we want to design equilibrium-stage processes, equilibrium and then K values are of first importance.

The concept of an equilibrium stage assumes that the two phases leaving a theoretical stage are at equilibrium. For each component leaving a stage, the mole fraction in the first phase (y_i) can be assumed proportional to the mole fraction in the other phase (x_i). This proportionality factor (y_i/x_i) is the distribution coefficient, K_i.

$$K_i = \frac{y_i}{x_i} \qquad (2.1)$$

The problem is to know the dependence of K_i's on temperature, pressure and concentrations of the other components j ($j \neq i$).

Prausnitz et al. (1985,p.21) explain the origin of the K factor. The equilibrium equation

states that the fugacity of component i in the liquid phase f_i^L equals the fugacity of component i in the vapor phase f_i^V,

$$f_i^V = f_i^L \tag{2.2}$$

We often make two key assumptions to simplify Eq.(2.2):

Assumption 1: The fugacity f_i^V at **constant temperature and pressure** is proportional to the mole fraction y_i.

$$f_i^V = y_i f^V_{pure\ i} \tag{2.3}$$

where $f^V_{pure\ i}$ is the fugacity of pure component i as a gas at the temperature and pressure of the mixture.

Assumption 2: The fugacity f_i^L at **constant temperature and pressure** is proportional to the mole fraction x_i.

$$f_i^L = x_i f^L_{pure\ i} \tag{2.4}$$

where $f^L_{pure\ i}$ is the fugacity of pure component i as a liquid at the temperature and pressure of the mixture.

The equilibrium relation (2.2) becomes:

$$y_i f^V_{pure\ i} = x_i f^L_{pure\ i} \tag{2.5}$$

Equation (2.5) gives an ideal-solution relation:

Until this point, assumptions 1 and 2 are necessary. Eq. (2.6) can be simplified with two

$$K_i = \frac{y_i}{x_i} = \frac{f^L_{pure\ i}}{f^V_{pure\ i}} \tag{2.6}$$

more assumptions:

Assumption 3: Pure component-i vapor at temperature T and pressure P is an ideal gas. Then:

$$f^V_{pure\ i} = P \tag{2.7}$$

Assumption 4: The effect of pressure on the fugacity of a condensed phase is negligible at moderate pressures. Further, the vapor in equilibrium with pure liquid i at temperature T is an ideal gas. It follows that:

$$f^L_{pure\ i} = P_i^s \tag{2.8}$$

where P_i^s is the saturation (vapor) pressure of pure liquid i at temperature T.

Finally, Eq. (2.6) becomes:

$$K_i = \frac{y_i}{x_i} = \frac{P_i^s}{P} \tag{2.9}$$

This is Raoult's law, applicable for K values only under assumptions 1 to 6 (constant T and P, low P and ideal gas).

A separation process is usually designed for a wide range of temperatures and pressures, including conditions under which Raoult's law is no longer valid. The K factors are far from being constant. Figure 2.1 shows the variations with temperature and pressure of the K values for

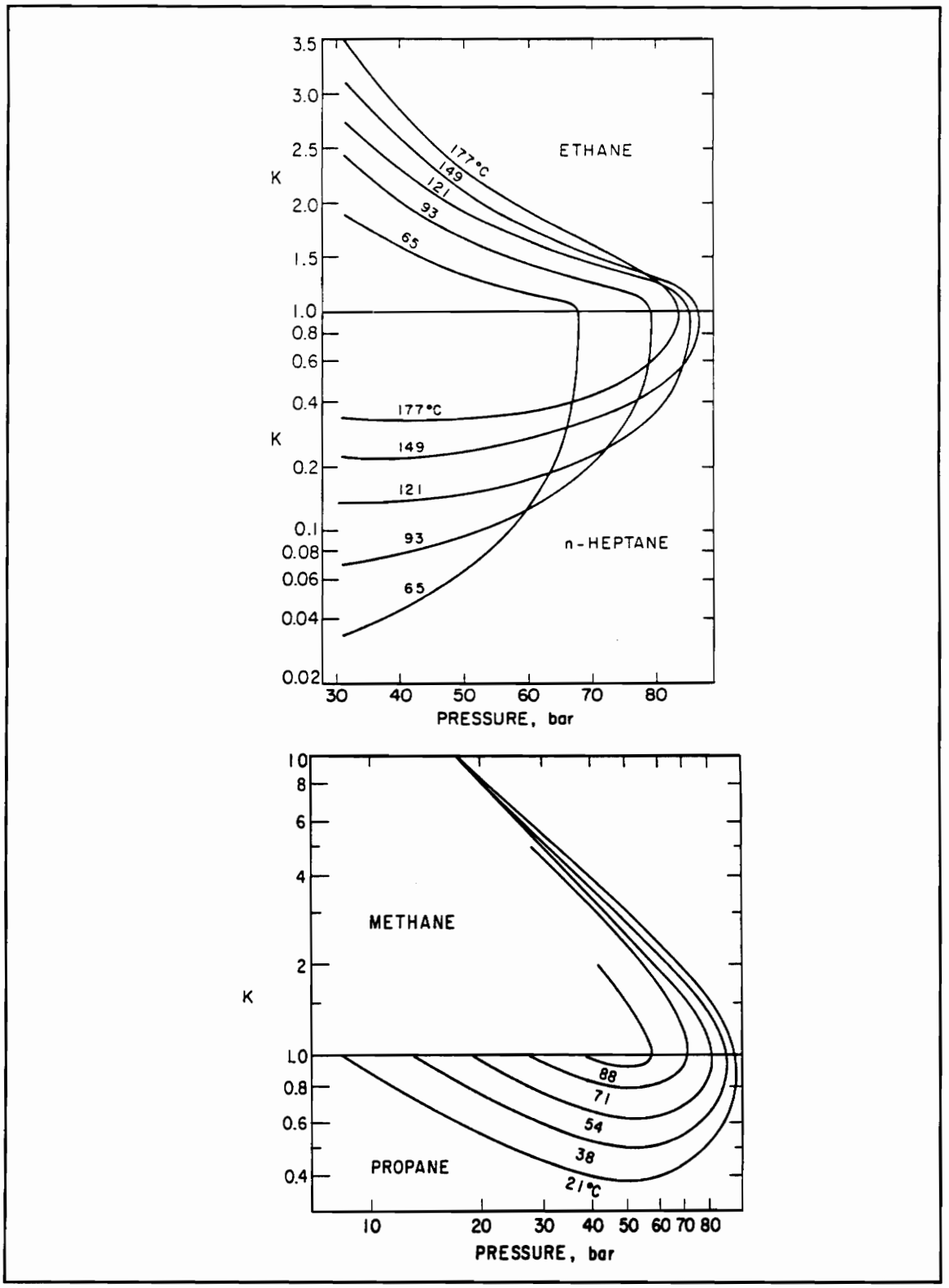


Figure 2.1: Influence of Temperature and Pressure on K Values (Prausnitz et al., 1986, pp.446,447).

two binary systems ethane/methane and methane/propane.

From Raoult's law to corresponding states theory, several methods can be used to calculate K values. Smith (1963, chapters 1 and 2) describes some of these methods. Specially for petroleum mixtures, tables and diagrams exist. A non-exhaustive list of this references appear in Appendices.

2.1.2 Key Component

To solve multicomponent separation processes, the use of a key component is necessary. In solvent-based separations, it is useful to decide how much of a certain component is allowed in one of the final products. Nelson (1969,p.853) gives an example of absorption, where he chooses butane (a low-boiling component) as the key component. Henley and Seader (1981,p.472) defines the key component to be "the heaviest component to be stripped to a specific extent". In Figure 2.2, we show how to choose the key component in two examples of absorption and stripping. For absorption (Fig. 2.2a), component D is the lightest component to be absorbed to a specific extent. Approximately 70% of the amount of D entering the column goes to the bottom, whereas only 2% of C is absorbed. Thus, D is chosen as the key component. For stripping (Fig. 2.2b), D is the heaviest component to be stripped to a specific extent (60%). D is the key component.

2.1.3 Absorption, Stripping and LLE Factors

When a key component is chosen, it is possible to calculate a dimensionless parameter (A: for absorption, S:for Stripping, and E:for LLE) which characterizes the separation.

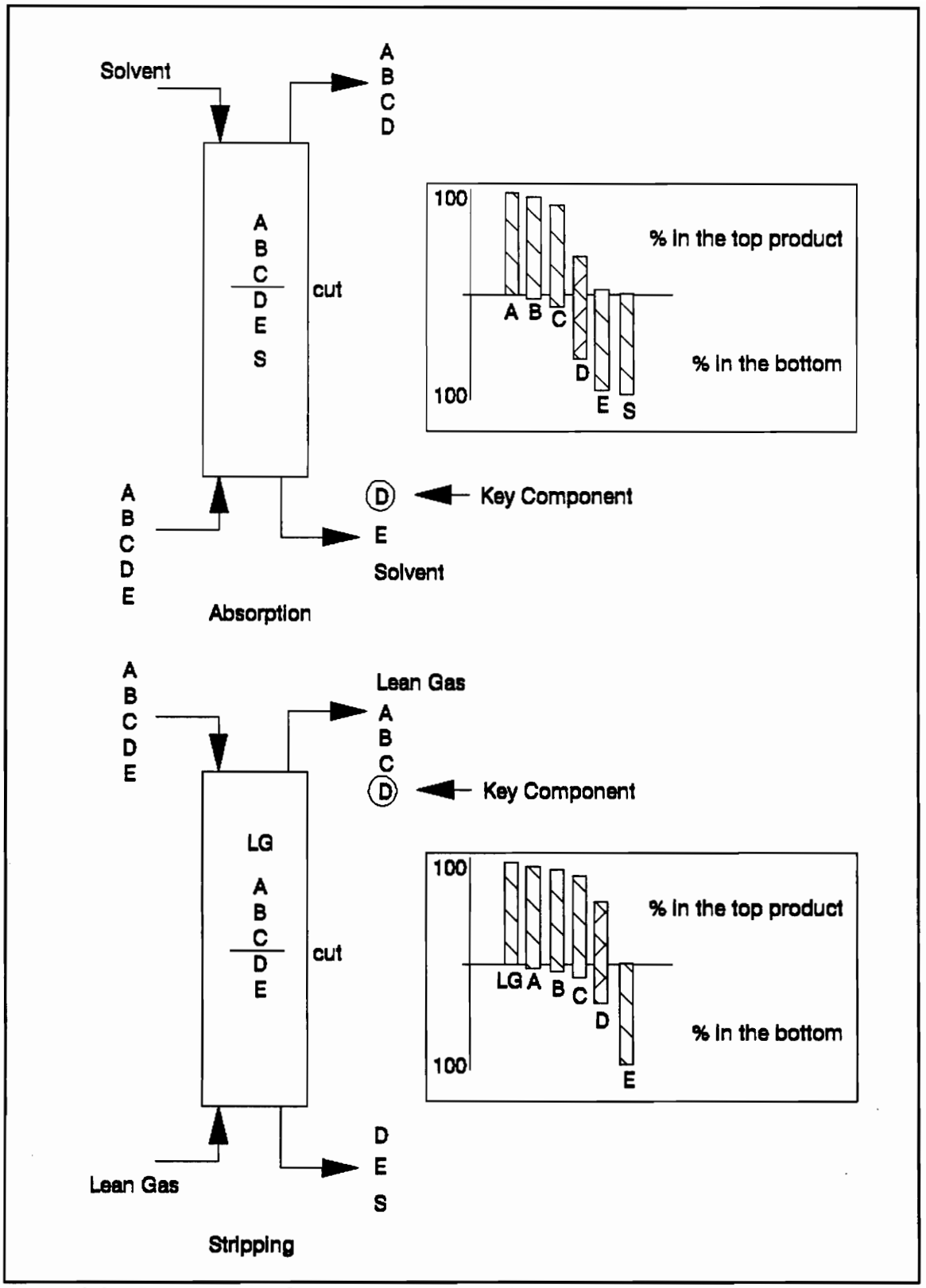


Figure 2.2 : Choice of Key Components in Absorption and Stripping.

$$A=\frac{L}{kV} \text{ for absorption}$$
$$S=\frac{kV}{L} \text{ for stripping} \qquad (2.10)$$
$$E=\frac{kS}{L} \text{ for LLE}$$

Here, k is the K-value of the key component; L is the **molar** liquid flow rate, V is the **molar** gas flow rate, for absorption and stripping; S is the **molar** solvent flow rate; and L is the **molar** feed flow rate for LLE. All the assumptions describe in 1.1.1 are necessary to use these factors.

2.1.4 Kremser Equation

2.1.4.1 Origin

Kremser (1930) originated a shortcut method to solve multicomponent solvent-based separations. Other authors such as Edmister (1957) improved the methods and emphasized particular applications like liquid-liquid separations (Brinkley, 1960). These methods are known as the *Group Methods*.

2.1.4.2 Derivation

We use the Kremser equation to evaluate the feasibility of a solvent-based separation.

The material balance on a stage gives (Fig. 2.3):

$$L\, x_{in}+V\, y_{j+1} = L\, x_j+V\, y_{out} \qquad (2.11)$$

which can be rearranged as:

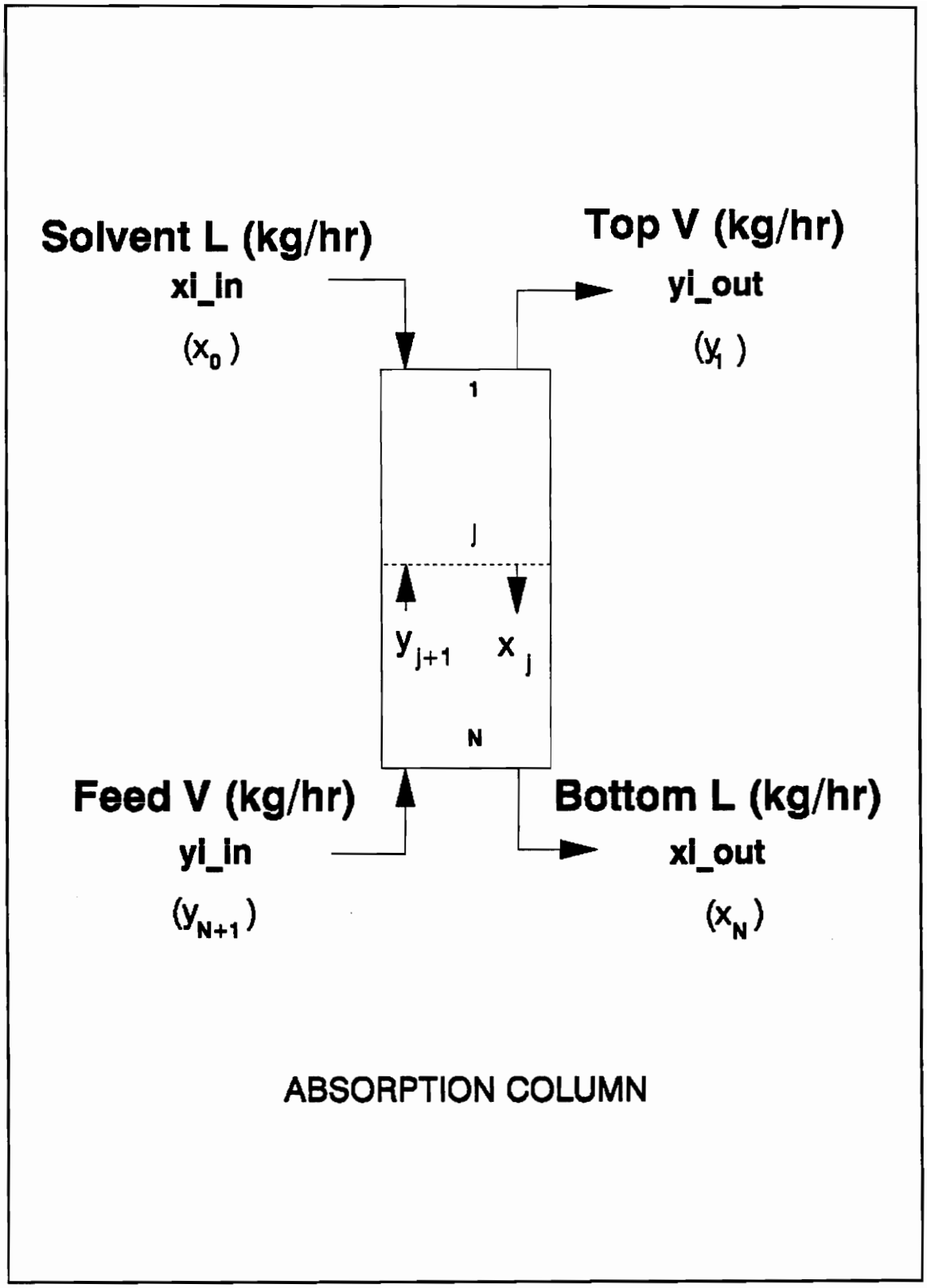


Figure 2.3 : A Stagewise Absorber

$$y_{j+1} = \frac{L}{V} x_j + (y_{out} - \frac{L}{V} x_{in}) \tag{2.12}$$

If the equilibrium line is straight and has its origin at 0, then Henry's law applies:

$$\begin{aligned} y_j &= m\ x_j + b \\ if\ b &= 0 \\ K_j &= \frac{y_j}{x_j} \end{aligned} \tag{2.13}$$

If we write Eq.(2.11) and Eq.(2.12) for N stages (j=N) and recombine all the derived equations, we obtain:

$$\frac{y_{out} - y_{in}}{y_{in}} = A\ \frac{\beta}{\alpha} - \frac{\alpha - 1}{\alpha} \tag{2.14}$$

where

$$\alpha = A^N + A^{N-1} + \ldots + 1 = \frac{1 - A^{N+1}}{1 - A} \tag{2.15}$$

and

$$\beta = A^{N-1} + A^{N-2} + \ldots + A + 1 = \frac{1 - A^N}{1 - A} \tag{2.16}$$

which leads to the Kremser equation in its form for absorption:

$$\frac{y_{i_{in}} - y_{i_{out}}}{y_{i_{in}} - K_i\ x_{i_{in}}} = \frac{A^{N+1} - A}{A^{N+1} - 1} \tag{2.17}$$

where N is the number of stages; A is the absorption factor; y_{iin} and y_{iout} are the molar fractions

of component i in the inlet and outlet gas phase, respectively; x_{iin} is the molar fraction of component i in the inlet liquid phase (solvent); and K_i is the K-value of component i.

2.1.4.3 Assumptions

Many assumptions are made in using the Kremser equation:

- To use mole fraction (instead of mass fraction), the **solution must be dilute**, that is, the component we want to remove must be in low concentration (1-5%).
- If the solution is dilute, Henry's law is applicable, then we can assume that the K values are independent of the concentrations throughout the column.
- If the solution is dilute, the variation in the molal overflow (L/V) is negligible, then we can assume that the absorption factors A_i are constant throughout the column (because the amount transfer from one phase to the other is small).

To assume that the K-values are constant, K(T,P) = constant, and temperature variation has negligible effects, we in fact assume:

- Isothermal and isobaric conditions.
- Negligible heat of absorption

2.1.4.4 Applicability

The main assumptions to be able to apply the Kremser equation are:

- Linearity of the equilibrium curve
- Dilute systems

In Table 2.1, Wankat (1988, p.552) gives more information about the applicability of the

Kremser equation.

Table 2.1: Applicability of the Kremser Equation

Operation	Feed	Constant flow	Units
Dilute Absorption	Vapor	Total flow rates V,L	y,x molar fractions
Dilute Stripping	Liquid	Total flow rates L,V	y,x molar fractions
Very Dilute Extraction	Liquid Raffinate	Total flow rates L,S	y,x weight (or mole) fractions
Washing	solids + underflow liquid	U=underflow liquid kg/hr O=overflow liquid kg/hr	y,x weight fractions
Leaching	solids & solutes	F solv=kg solv/hr F solid=kg insoluble solids/hr)	$Y=\frac{\text{kg solute}}{\text{kg solvent}}$ $X=\frac{\text{kg solute}}{\text{kg solid}}$
Adsorption & ion exchange	Gas or Liquid	S=absorbent solids,kg/hr G=fluid,kg/hr	$Y=\frac{\text{kg solute}}{\text{kg solvent}}$ $q=\frac{\text{kg solute}}{\text{kg absorbent}}$
three-phase	one or 2 phases	L,W,V	y,x mole fractions (both phases)

2.1.5 Mass- and Energy-Separating Agents

Extractive and azeotropic distillations use both MSA and ESA.

2.1.5.1 Extractive Distillation

Extractive distillation is used for the separation of azeotropes and close-boiling mixtures. (Wankat, 1988, p.303). A solvent is added during the distillation process to make the distillation feasible. Butane and butene, for instance, are separated by this method with furfural as the

solvent.

2.1.5.2 Azeotropic Distillation

Azeotropic distillation is performed by adding a solvent or entrainer that forms an azeotrope with one or both of the components (Wankat, 1988 p.309). For example, a mixture of cyclohexane [boiling point at 101.3 kPa: 80.8°C] and benzene [80.2°C], forms a minimum-boiling homogeneous azeotrope [77.4°C]. A separation of this mixture by simple distillation is not possible. The mixture is fed to an azeotropic-distillation column together with acetone as an entrainer. A minimum-boiling binary homogeneous azeotrope forms [53.1°C] with acetone [56.4°C]. Thus near-pure benzene is removed as bottoms.

The acetone-cyclohexane azeotrope is removed as distillate and is treated with water in a liquid-liquid extraction column. Near-pure cyclohexane leaves as overhead. An acetone-water mixture leaves the bottom of the extractor and is separated by simple distillation into product streams that are recycled (Perry and Green, 1986, p.13-58).

2.1.6 Separation-Selection Heuristics

Wankat (1990, section 14-5) gives some heuristics to select a separation method. We outline some of them.

- Choose MSA-based separations when distillation (ESA) is not possible. *This is the most important heuristic to choose for MSA based separations.*
- Try not to use an MSA to recover another MSA.
- Prefer processes with only one, and not two, MSAs.
- In organic systems, remove water early.

2.2 Absorption

2.2.1 Characteristics

Wankat (1988,p.471) defines absorption as the unit operation where one or more components of a gas stream are removed by being transferred (absorbed) from the gas to a nonvolatile liquid called the solvent. This liquid solvent (or absorbent) is called a *mass-separating agent (MSA)*. Stripping is the opposite unit operation. One or more components of the liquid stream are removed by being transferred to an insoluble gas stream (MSA).

Kohl and Riesenfield (1987, p.340) explain the three distinct types of absorption. In the case of physical absorption, the absorbed component (or absorbate) is more soluble in the solvent than the others, but it does not react with it. This is usually what occurs in the recovery of light hydrocarbons with oil.

Chemical absorption is of two kinds. A reversible reaction occurs when the absorbed product reacts with the liquid solvent or one of its components. The vapor-liquid equilibrium is no longer linear and the rate of absorption is affected by the rate of reaction. An example is the absorption of carbon dioxide into a monoethanolamine solution. The absorption of H_2S by strongly oxidizing solutions is an example of absorption with irreversible reaction. The absorption itself is less affected when the reaction is irreversible. **This entire study concerns only physical absorption.**

2.2.2 Use of Kremser Equation for Absorption

The assumptions of section 2.1.4.3 have to be verified. The assumption of a dilute feed mixture is easy to check because the feed is known. Assumptions, such as constant temperature

and pressure, can be checked after rigorous simulations.

2.2.2.1 Absorption Factor

The absorption factor for a component i is:

$$A_i = \frac{L}{K_i V} \tag{2.18}$$

where L is the liquid flow rate and V the vapor flow rate. K_i is the K value for component i.

2.2.2.2 Number-of-Stages Evaluation

We choose a component i; if we estimate the liquid flow rate L, we can calculate the absorption factor A_i:

$$A_i = \frac{L}{K_i V} \tag{2.19}$$

Here, K_i is the K value of component i, V is usually known, and L is estimated.

The Kremser equation is:

$$\frac{y_{i_{in}} - y_{i_{out}}}{y_{i_{in}} - K_i x_{i_{in}}} = \frac{A_i^{N+1} - A_i}{A_i^{N+1} - 1} \tag{2.20}$$

where N is the number of stages; A_i the absorption factor of component i; y_{iin} and y_{iout} are the molar fractions of component i in the inlet and outlet gas phase, respectively; x_{iin} is the molar fraction of component i in the inlet liquid phase (solvent).

Then, we can find the number of theoretical stages, N:

$$N=\frac{\ln\left[\frac{\frac{y_{i_{in}}-y_{i_{out}}}{y_{i_{in}}-K_i\,x_{i_{in}}}-A_i}{\frac{y_{i_{in}}-y_{i_{out}}}{y_{i_{in}}-K_i\,x_{i_{in}}}-1}\right]}{\ln A_i}-1 \qquad (2.21)$$

with i=fixed component

If $A_i=1$, the equation is:

$$N=\frac{y_{i_{in}}-y_{i_{out}}}{y_{i_{out}}-K_i\,x_{i_{in}}} \qquad \textit{with i=fixed component} \qquad (2.22)$$

2.2.2.3 Component Recovery

In the Kremser equation, usually y_{iin} and x_{iin} are known from the problem statement. The recovery of component i, i.e., y_{iout}, is then given by:

$$y_{i_{out}} = y_{i_{in}} - (y_{i_{in}} - K_i\,x_{i_{in}})\left(\frac{A_i^{N+1}-A_i}{A_i^{N+1}-1}\right) \qquad (2.23)$$

Eq. (2.23) is valid when $A_i \neq 1$; when $A_i=1$, we use:

$$y_{i_{out}}=\frac{y_{i_{in}}+N\,K_i x_{i_{in}}}{N+1} \qquad (2.24)$$

Usually, x_{iin}, the molar fraction of component i in the solvent, is 0. When x_{iin} is different from zero, it means that a component is present in the solvent. Thus, Eqs. (2.23) and (2.24) show that

the Kremser equation is still valid when the solvent contains one or several components of the feed.

2.2.3 Heuristics for Absorption

We classify heuristics for absorption into five categories: method heuristics, species heuristics, composition heuristics, design heuristics and solvent-selection heuristics.

a) Method Heuristics

Small amounts (2-3 mole%) of high-boiling components are to be removed from a vapor stream. *Energy cost is too high for distillation* (Keller, 1982, p.45).

The feed stream has to be cooled, or compressed, or both, before its bubble point is reached. *Costs are less than for distillation* (Keller, 1982, p.45).

The relative volatilities are too close for distillation, and the use of a solvent can significantly improve them. *The solvent provides a medium with highly different affinities for components* (Keller, 1982, p.45).

Small amounts of components can be removed by an irreversible reaction with a solvent. Only for systems where: 1) the material to be removed is in low concentration (1000 ppm to 10,000 ppm), 2) solvent cost is low, and 3) spent solvent is easily salvaged or disposed. *Such systems exist when air-pollution or toxicity occur and a very high percentage of removal is required* (Keller, 1982, p.45).

b) Species Heuristics:

Favor high absorption factor (L/KV) for highly valuable solute. Lower the factor for less valuable solutes (Perry and Green, 1986, p.14-29). (see also: energy-conservation and heat-integration heuristics below).

c) Composition Heuristics:

For trace removal, choose a solvent with a strong affinity for the solute. *Will require both temperature elevation and pressure reduction to recover the solvent* (Keller, 1982, p.42).

For bulk removal, favor a solvent with a lower affinity for the solute. *Utilize pressure reduction only for solvent recovery* (Keller, 1982, p.42).

d) Design Heuristics:

We summarize the design heuristics in two categories: the column heuristics and the energy-conservation heuristics.

- Column heuristics:

Theoretical number of stages $N \geq 5$. *The only exceptions are: 1) low desired recoveries of solute (< 80-90%), 2) extremely soluble solvent. Increasing N reduces the solvent recirculation, the energy required for solvent circulation declines substantially as the number of stages increases, at least up to 5 stages* (Keller, 1982. p.49).

A second phase is added as the separating agent (liquid). *Then the design is easier because the column does not have reboiler and condenser* (Wankat, 1988, p.472).

The stage efficiencies range from a few percent (highly viscous solvent or slow reaction) to no more than about 50% (Keller, 1982, p.49).

Overdesign the column in the absence of complete data on equilibrium and stage efficiency (Keller, 1982, p.50).

Favor bubble-cap tray design only when there is: 1) low liquid flow rates, 2) high turndown-ratio service, or 3) multiple feed and draw-off points. Investigate valve trays if bubble-caps are appropriate (Kohl and Riesenfield, 1987, p.343).

Favor sieve-tray design when there is:1) medium liquid flow rate, or 2) difficult separation requiring many stages (Kohl and Riesenfield, 1987, p.343).

Favor packed columns when there is: 1) high liquid rate, 2) difficult separations requiring many stages, 3) foaming or corrosive liquids, or 4) design requiring maximum flexibility and versatility (Kohl and Riesenfield, 1987, p.343).

Favor spray contacting when there is: 1) high liquid rate, 2) easy separation requiring only one stage, 3) corrosive fluid, 4) solid or viscous fluid, or 5) low ΔP required across the column (Kohl and Riesenfield, 1987, p.344).

Minimize solvent circulation rates by **increasing the number of theoretical stages** (especially if $N < 5$); and by **maintaining the absorber as close to isothermal as possible**, by using a solvent with higher or lower molecular weight, or a lower activity coefficient (Keller,

1982, pp.48,51 and 55).

The column diameter is calculated at the bottom of the column. Because both flows V (vapor) and L (liquid) are larger at the bottom of the column (Wankat, 1988, p.485).

If the gas rate is greater than 14 m^3/min, a nominal packing size smaller than 25 mm (1 in.) would not be practical. For more than 56 m^3/min, the limit is 50 mm (2 in.) (Zenz, 1979, p.3-61).

- Energy-Conservation and Heat-Integration Heuristics:

For isothermal absorbers, targeting high recoveries (>99%), favor an absorption factor (L/KV) between 1 and 2, with L/KV ≈ 1.4 as optimal. *Higher values of L (increased solvent flow rate) raise recovery stripper costs. Lower values of L (reduced solvent flow rate) require more equilibrium stages and increase absorber costs* (Douglas, 1988, p.77; Treybal, 1980, p.291).

Favor low solvent inlet temperature, and minimize the temperature rise in the column through the use of an intercooler. *A temperature rise generates a decrease of the solvent capacity* (Douglas, 1988, p.81).

Place intercoolers in the column to minimize the adiabatic temperature rise (Keller, 1982, p.54).

Avoid the use of adiabatic absorbers, unless there is only a small temperature rise across the absorber. *Cost increases substantially* (Douglas, 1988, p.90; Keller, 1982, p.54).

Favor a staged-blowdown pressure reduction for solvent recovery, particularly when the solute composition in the absorber feed is high (Keller, 1982, p.51).

Use an unabsorbed "product gas", where possible, as the stripping gas for the recovery column (Keller, 1982, p.55).

Favor the pressure-reduction mode of solvent recovery (gas desorption or stripping) if the feed gas is originally supplied at high pressure (Keller, 1982, p.56).

If the vapor feed to the absorption column is at a low pressure, it usually "does not pay" to increase the column pressure. *Compressor capital costs will usually outweigh solvent-recovery savings resulting from reduced liquid flow rate* (Douglas, 1988, p.82).

When heat effects are important, the addition of stages will have no useful effect on the efficiency. Solutions to this difficulty can be sought by increasing the solvent flow, introducing strategically placed coolers, cooling and dehumidifying the inlet gas, and/or raising the tower operating pressure (Perry and Green, 1984, p.14-24)

e) Solvent-Selection Heuristics:

The limiting relative volatilities of solutes and species that are not absorbed are 1.5. *However, it is better to keep them as high as 10* (Keller, 1982, p.45).

High selectivity (above 2.0) for the material to be absorbed. *Low selectivities need additional energy* (Keller, 1982, p.45).

High capacity for absorbed material. *It permits low solvent circulation and lowers the recovery cost* (Keller, 1982, p.46).

Low cost or alternatively extreme stability and simple means to prevent losses. *Cheap or easy to recover* (Keller, 1982, p.46).

Low volatility. *It prevents losses with the unabsorbed gas* (Keller, 1982, p.46).

Low viscosity. *Minimizes pumping problems and promotes high-stage efficiencies* (Keller, 1982, p.46).

Low toxicity, low corrosivity, capability for simple and safe disposal or reclaiming (Keller, 1982, p.46).

Nonflammable, low freezing point. *This conditions should be favored when possible for security reasons* (Treybal, 1980, p.282).

If reactive solvent is used, favor a reaction that is reversible. *To reuse the solvent, we want a reversible reaction* (Treybal,1980, p.281).

Favor solvents that yield reversible chemical absorption rather than those giving irreversible chemical or physical absorption (Douglas,1988, p.81).

2.3 Stripping

2.3.1 Characteristics

The American Petroleum Institute (API, 1967, p.10-3) defines *stripping* as a form of distillation wherein a small amount of a relatively volatile material is removed from a large volume of a less volatile material.

Stripping is the inverse operation of absorption. The petroleum industry often uses it for solvent recovery. A major application of stripping is for refinery waste disposal. Stripping is frequently used for removing small amounts of volatile impurities, e.g., hydrogen sulfides and ammonia from large volumes of waste waters. However, this method is not suitable for low-volatility impurities like phenolic. The most frequent use of stripping operation in refinery waste disposal is to remove hydrogen sulfide and ammonia from sour waters (see section A.1.2).

2.3.2 Use of Kremser Equation for Stripping

The assumptions of section 2.1.4.3 have to be verified. The assumption of a dilute feed mixture is easy to check because the feed is known. Assumptions, such as constant temperature and pressure, can be checked after rigorous simulations.

2.3.2.1 Stripping Factor

The stripping factor for a component i is,

$$S_i = \frac{K_i V}{L} \tag{2.25}$$

where L is the liquid flow rate and V the vapor flow rate. K_i is the K value for component i.

2.3.2.2 Number-of-Stages Evaluation

We choose a component i (Fig. 2.4); if we estimate the vapor flow rate V, we can calculate the stripping factor of the column:

$$S_i = \frac{K_i V}{L} \tag{2.26}$$

K_i is the K value of component i, L is usually known, V is estimated.

We rewrite the Kremser equation for stripping as follows:

$$\frac{x_{i_{in}} - x_{i_{out}}}{x_{i_{in}} - \frac{y_{i_{in}}}{K_i}} = \frac{S_i^{N+1} - S_i}{S_i^{N+1} - 1} \tag{2.27}$$

Here N is the number of stages; S_i is the stripping factor of component i; x_{iin} and x_{iout} are the mole fraction of component i in the inlet and outlet liquid phase (solvent), respectively; y_{iin} is the mole fraction of component i in the inlet gas phase (lean gas).

If $S_i \neq 1$, the number of theoretical stages N is:

$$N = \frac{\ln\left[\dfrac{\dfrac{x_{i_{in}} - x_{i_{out}}}{x_{i_{in}} - \dfrac{y_{i_{in}}}{K_i}} - S_i}{\dfrac{x_{i_{in}} - x_{i_{out}}}{x_{i_{in}} - \dfrac{y_{i_{in}}}{K_i}} - 1}\right]}{\ln S_i} - 1 \tag{2.28}$$

with i=fixed component.

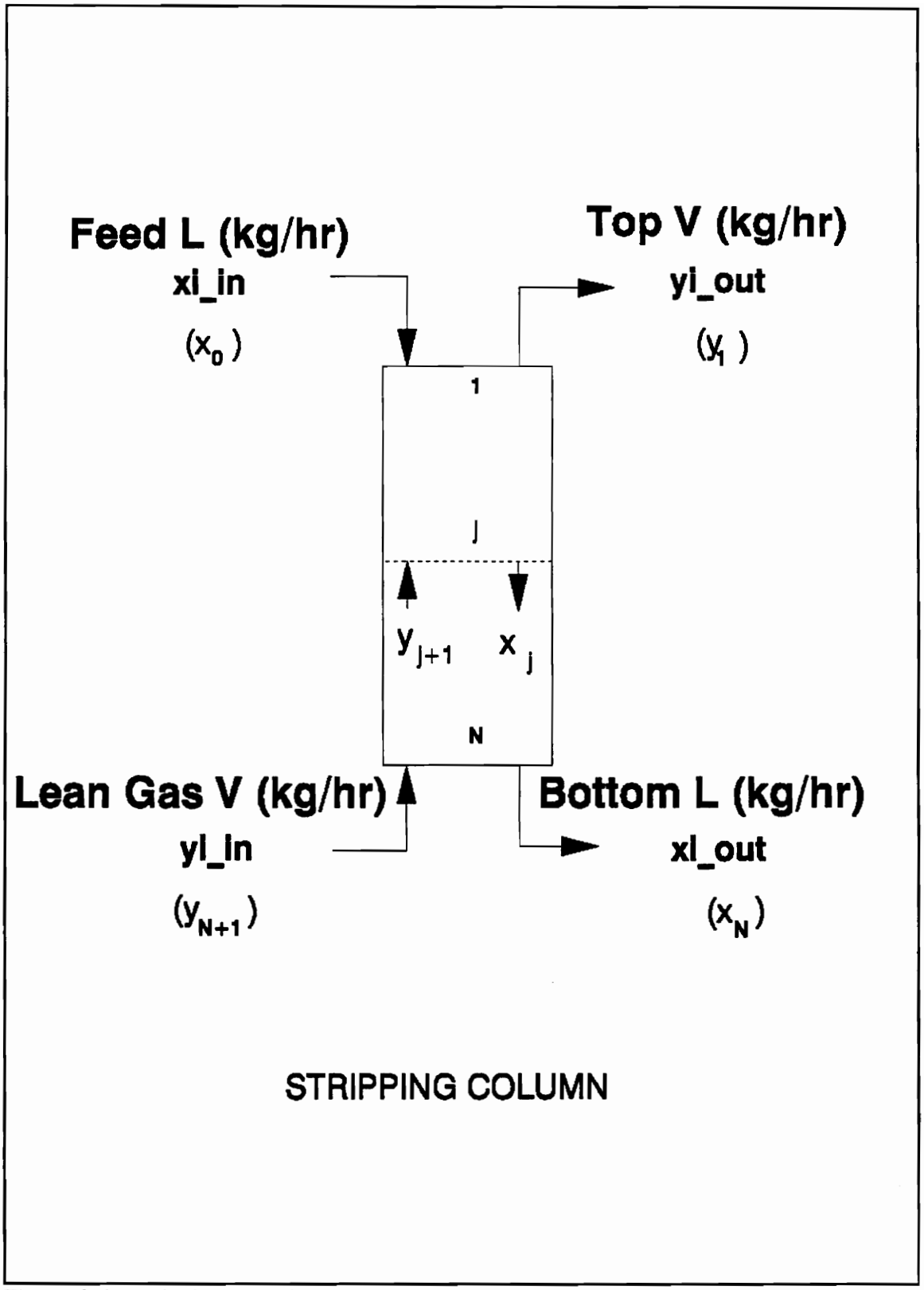


Figure 2.4 : Stripping Flowchart

When $S_i = 1$, N is:

$$N = \frac{x_{i_{in}} - x_{i_{out}}}{x_{i_{in}} - \frac{y_{i_{in}}}{K_i}} \qquad \textit{with } i\textit{=fixed component} \tag{2.29}$$

2.3.2.3 Component Recovery

Usually we know x_{iin} and y_{iin} from the problem statement. The Kremser equation then gives the recovery of component i, i.e., x_{iout}:

$$x_{i_{out}} = x_{i_{in}} - \left(x_{i_{in}} - \frac{y_{i_{in}}}{K_i}\right)\left(\frac{S_i^{N+1} - S_i}{S_i^{N+1} - 1}\right) \tag{2.30}$$

Equation (2.30) is valid when $S_i \neq 1$; when $S_i = 1$, we use:

$$x_{i_{out}} = \frac{x_{i_{in}} + N\,\frac{y_{i_{in}}}{K_i}}{N+1} \tag{2.31}$$

Usually, y_{iin}, the molar fraction of component i in the lean gas, is 0. When y_{iin} is different from zero, it means that a component is present in the lean gas. Thus, Eqs. (2.30) and (2.31) show that the Kremser equation is still valid when the lean gas contains one or several components of the feed.

2.3.3 Heuristics for Stripping

Most of the heuristics that we enumerated for absorption stay valid. However, stripping

has specific applications, which we describe below as the method heuristics. Some heuristics deal specifically with the lean gas.

a) Method Heuristics

Stripping is usually chosen for the recovery of solvents, or for the purification of liquid wastes (API, 1967, p.10-3).

Stripping is not suitable for removing materials of low volatility, e.g., phenolics. *Because heating and equipment requirements make it uneconomical* (API, 1967, p.10-3).

b) Species Heuristics

Oxidation competes with stripping as a means of reducing the sulfide content in refinery effluent (API, 1967, p.10-3).

Do not use stripping for phenolics in sour-water removal. Prefer oxidation (API, 1967, p.10-3).

Stripping is very often used for removal of hydrogen sulfide and ammonia from sour waters (waste water containing usually CO_2 and NH_3) (API, 1967, p.10-3).

c) Composition Heuristics

Same as absorption plus:

Stripping is not always the most economical method to use when large quantities of impurities are to be removed from waste waters unless the impurities are valuable enough to

justify recovery (API, 1967, p.10-3).

Acidify the sour waters when refinery sour waters are to be stripped of sulfur components. *It converts the sulfides to hydrogen sulfides* (API, 1967, p.10-3).

Acidification should not be used if ammonia removal is desired (API, 1967, p.10-3).

d) Design Heuristics

Same heuristics as absorption except:

In the design of stripping towers, the optimum value of the stripping factor will be in the range of 0.5 to 0.8. *This heuristic is consistent with heuristic for absorption factor ($1 < A < 2$ opt. $= 1.4$) given by Douglas (1988, p.77); indeed, $1/2 = 0.5$ and $1/1.4 = 0.71$* (Perry and Green,1986, p.14-29).

It may not be desirable to heat waste water to the temperature required for stripping at atmospheric pressure or greater. *Decomposition of material may occur or the cost of heating may be too high* (API, 1967, p.10-3).

If raschig rings are used, choose them larger than 2 in. (API,1967, p.10-4).

e) Lean-Gas Selection Heuristics

The lean gas is usually an inert gas such as steam or nitrogen (Perry and Green,1986, p.14-17).

The lean gas can be a flue gas (mixture of air and CO_2) (API, 1967, Chap.15).

Inert-gas stripping of an aqueous solution is generally used only when the overhead vapors are incinerated. *When stripping with inert gas, incineration is important because vapor-pressure relationships make it difficult to condense the material stripped from waste waters* (API, 1967, p.10-3).

f) Heuristics for Steam-Stripping

- Favor steam as a lean gas for stripping a pollutant from an aqueous solution. *The advantage is that the steam is used both as a source of heat and as a diluent to reduce the effective pressure* (API, 1967, p.10-3).

- A condenser on the overhead line from the column is desirable to reduce the amount of steam entering the firebox or incinerator in which the stripped gases are burned (API, 1967, p.10-4).

- Van Winkle (1967, p.359) states that for petroleum-oil strippers, the usual steam requirement is proportional to the amount of stripped product. He gives the following table:

Table 2.2 Steam Requirement for Hydrocarbons Stripping

Stripped Product	**kg steam / m^3 Stripped**
Naphtha	23 - 60
Kerosene (or diesel)	23 - 60
Gas Oil	23 - 60
Neutral Oils	48 - 108
Topped Crude	48 - 144
Residual Cylinder Stock	120 - 600

These values are consistent with heuristics on the stripping factor.

Example:

We want to strip a mixture of oil and light hydrocarbons with steam to recover the oil. The feed is 60.49 mol/hr or 9010 kg/hr. With 2191 kg/hr (86.5 mol/hr) stripped, and the overhead product has a density of 870.8 kg/m^3. The amount stripped is then: 2191/870.8=2.52 m^3/hr. The K value of the key component is k=0.75. According to the table the ratio kg steam/m^3 stripped is:

$$48 < V(\text{kg steam})/L(m^3) < 108$$

$$48 \times 0.75 \times 9.5 \times 10^{-3} < kV(\text{mole/hr})/L(\text{mole/hr}) < 108 \times 0.75 \times 9.5 \times 10^{-3}$$

$$\underline{0.35 < S < 0.77}$$

which is close to the heuristic range $0.5 < S < 0.8$.

It is, however, difficult to use Table 2.2, because of the need for the density of the liquid flow.

2.4 Liquid-Liquid Extraction

2.4.1 Characteristics

Othmer (1983, p.xi) defines *solvent extraction* as the separation to obtain, from an initial liquid, a dissolved material such as another liquid or a solid. We limit this study to the use of extraction to obtain another liquid. This limitation restricts the term solvent extraction to liquid-liquid extraction (LLE).

LLE is a process where one or more solutes are removed from a liquid by transferring the solute(s) into a second liquid phase, explains Wankat (1988, p.515). The two liquid phases must either be immiscible (insoluble in each other) or partially miscible. LLE is a common industrial and laboratory operations. It is used not only for pharmaceutical products, such as the production of penicillin, but also in the petroleum industry for aromatic recovery or benzene alkylation (Keller, 1982 p.67). Solvent extraction is a relatively rare operation in the treatment of refinery waste waters. However, it has one important application: the extraction of phenolic materials (API, 1967, p.10-5).

Robbins (1979, section 1-9) explains that LLE systems can be classified into three types: immiscible solvents, partially miscible solvents with a low solute concentration in the extract, and partially miscible solvent with a high solute concentration in the extract.

2.4.2 Kremser Equation Applied to LLE

The assumptions of section 2.1.4.3 have to be verified. The assumption of a dilute feed mixture is easy to check because the feed is known. Assumptions, such as constant temperature and pressure, can be checked after rigorous simulations.

2.4.2.1 Extraction Factor

Like in absorption and stripping, a separation factor can be used in LLE. The extraction factor is defined by the following equation,

$$E_i = \frac{K_i S}{L} \quad or \quad E_i = \frac{K_i V}{L} \tag{2.32}$$

where E_i is the extraction factor; L is the feed molar flow rate (heavy liquid containing the multicomponent mixture); S (or V) is the solvent molar flow rate (light liquid); and K_i is the distribution coefficient of component i.

2.4.2.2 Number-of-Stages Evaluation

We choose a component i (Fig. 2.5); if we estimate the solvent flow rate S, we can calculate the extraction factor of the column:

$$E_i = \frac{K_i S}{L} \tag{2.33}$$

K_i is the K value or distribution coefficient of component i; L is usually known and S is estimated.

We rewrite the Kremser equation for LLE as follows:

$$\frac{x_{i_{in}} - x_{i_{out}}}{x_{i_{in}} - \frac{y_{i_{in}}}{K}} = \frac{E_i^{N+1} - E_i}{E_i^{N+1} - 1} \tag{2.34}$$

where N is the number of stages; E_i is the extraction factor of component i; x_{iin} and x_{iout} are the mole fraction of component i in the inlet and outlet feed L (raffinate), respectively; y_{iin} is the

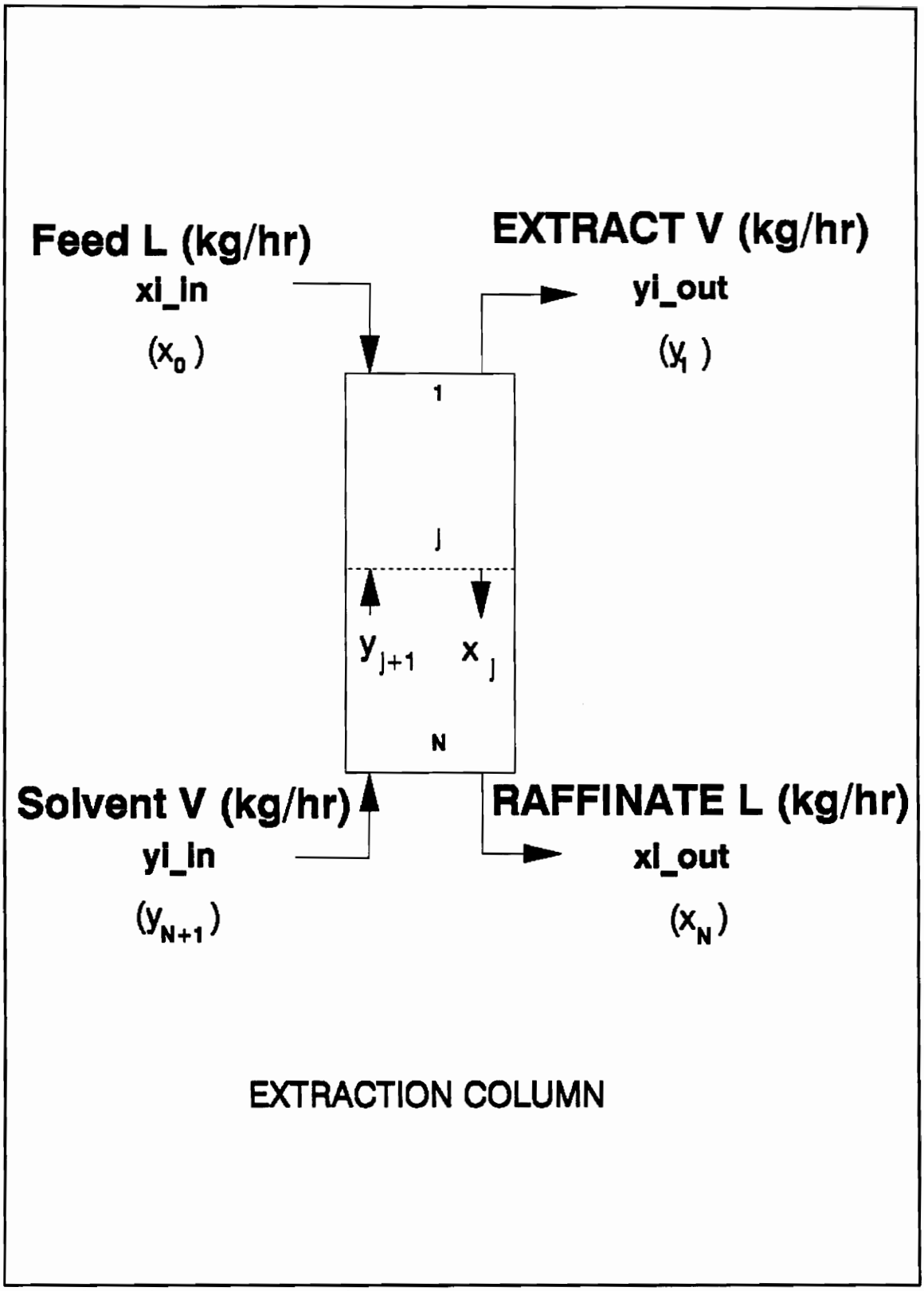


Figure 2.5 : Liquid-Liquid Extraction (LLE) Flowchart

mole fraction of component i in the inlet solvent phase S (solvent or extract).

Then we can find the number of theoretical stages N:

$$N = \frac{\ln\left[\dfrac{\dfrac{x_{i_{in}} - x_{i_{out}}}{x_{i_{in}} - \dfrac{y_{i_{in}}}{K_i}} - E_i}{\dfrac{x_{i_{in}} - x_{i_{out}}}{x_{i_{in}} - \dfrac{y_{i_{in}}}{K_i}} - 1}\right]}{\ln E_i} \qquad (2.35)$$

with i=fixed component.

When $E_i = 1$, N is:

$$N = \frac{x_{i_{in}} - x_{i_{out}}}{x_{i_{out}} - \dfrac{y_{i_{in}}}{K_i}} \qquad \textit{with i=fixed component} \qquad (2.36)$$

2.4.2.3 Component Recovery

In the Kremser equation, usually x_{iin} and y_{iin} are known from the problem statement, then the recovery of component i, i.e., x_{iout}, is given by:

$$x_{i_{out}} = x_{i_{in}} - \left(x_{i_{in}} - \frac{y_{i_{in}}}{K_i}\right)\left(\frac{E_i^{N+1} - E_i}{E_i^{N+1} - 1}\right) \qquad (2.37)$$

Equation (2.37) is valid when $E_i \neq 1$; when $E_i = 1$, we use:

$$x_{i_{out}} = \frac{x_{i_{in}} + N \frac{y_{i_{in}}}{K_i}}{N+1} \qquad (2.38)$$

Usually, y_{iin}, the molar fraction of component i in the solvent, is 0. When y_{iin} is different from zero, it means that a component is present in the solvent. Thus, Eqs. (2.37) and (2.38) show that the Kremser equation is still valid when the solvent contains one or several components of the feed.

2.4.3 Heuristics for Extraction

a) Method Heuristics

Choose LLE for the separation of groups of chemically dissimilar components with overlapping boiling ranges, *for example, benzene, toluene, and xylenes from C_6 to C_8 paraffins* (Keller, 1982, p.66).

The removal of extract from the solvent is often accomplished by distillation (Keller, 1982, p.64).

Solvent losses in the raffinate in waste treatment is the main drawback of extraction (Keller, 1982, p.76).

Extraction must always be considered in the context of the overall process of which it forms a part. *The final decision is based on technical and economic feasibility, and safety considerations* (Blumberg, 1988, p.180).

b) Species Heuristics

Favor LLE for removal of phenolics (100 to 1000 mg/liter). Removal of 90 to 99 % is commonly attained (API, 1967, p.10-5).

Foaming and emulsification characteristics greatly affect the design (API, 1967, p.10-5).

In the refining of kerosene and lubricating oils, LLE is widely used and solvents are usually furfuraldehyde or phenol (Bailes, 1976, p.232).

c) Composition Heuristics

d) Design Heuristics

- Column Heuristics:

Favor a low number-of-stages extractor ($N < 5$). *An extractor with more than 5 stages will be more complex and expensive* (Keller, 1982 p.64).

At least 5 to 10 theoretical stages should be used to attain a reasonably low reboiler duty for solvent recovery. *Decreasing the number of stages increases the amount of solvent needed* (Keller, 1982, p.75).

Use 10 to 15 stages in countercurrent extraction. *Heuristic from the economical analysis of the influence of the extraction factor* E (Jenson and Jeffreys, 1971).

The number of extraction stages appears to have been optimized at five to seven in many petroleum refining operations (Hanson, 1971, p.117).

Mixer-settlers batteries are built with up to five *practical* stages (Reissinger, 1978, p.248).

In a Podbielniak extractor (extractor with concentric sieve trays), three to five theoretical stages can be achieved (Reissinger, 1978, p.250).

A mixer-settler unit is approximately equivalent to one equilibrium stage (Baird, 1991, p.1290).

[We shall take N=5 as the optimal number of stages]

Design (and operate) extraction processes with care. *Otherwise, excessive solvent losses occur* (Keller, 1982, p.64).

A countercurrent column, or a combination of mixer and settler unit, is usually used (API, 1967, p.10-6).

Centrifugal extractors require less space than columns and are easy to clean, but operating costs are generally higher (API, 1967, p.10-5).

As the number of stages required by an extraction process increases, columns tend to be favored over mixer-settlers. *There are exceptions to this rule of thumb. An example is the process*

for rare earth separations that contains several hundred mixer-settler stages (because of the need for the control of pH) (Baird, 1991, p.1290).

Mechanically-agitated columns enable one to increase the number of stages and decrease the solvent circulation, but at the price of mechanical complexity and higher investment (Keller, 1982, p.73).

Sieve-tray and packed columns are more frequently used than spray-columns (Reissinger, 1978, p.251).

A well-designed mixer-settler provides 0.8 to 0.9 theoretical stages per actual stage (Cusack, 1991b, p.133).

- Energy-Conservation and Heat-Integration Heuristics:

For extractors targeting high recoveries (>99%), favor an extraction factor (E=KS/L) between 1 and 1.25. *Higher values of L (increased solvent flow rate) raise recovery costs. Lower values of L (reduced solvent flow rate) require more equilibrium stages, and increase absorber costs* (Pratt, 1983, p.191).

As a general guideline, an extraction process should be run with a minimum extraction factor (E) of 1.3 . *Below this value, the number of stages is excessive and impractical* (Cusack, 1991a, p.70).

e) Solvent-Selection Heuristics

The selectivity should be in excess of 2.0. *It will reduce the number of stages* (Keller, 1982, p.68).

Favor a solvent with high capacity for the extract. *It minimizes solvent circulation and recovery costs* (Keller, 1982, p.68).

Favor easy separability between the feed-raffinate and the solvent phases, i.e., large density differences and no emulsion formation (Keller, 1982, p.68).

Favor low solvent solubility in the feed-raffinate phase. *It reduces solvent losses or recovery costs* (Keller, 1982, p.68).

Favor a high relative volatility (2.0 or higher) between solvent and extract. *It makes the recovery (by distillation usually) easier* (Keller, 1982, p.68).

The boiling point of the solvent should be much higher or much lower than those of the components. *Ease of recovery* (Keller, 1982, p.71).

Favor a solvent that creates moderate interfacial tension. *The lower the interfacial tension, the less energy is needed to create a droplet dispersion. However, a very low value (1 dyne/cm) may create emulsions that are either impossible to settle or take impractically long* (Cusack, 1991a, p.68)

Choose a low-viscosity solvent. *It minimizes pumping problems and promotes high stage*

efficiencies (Keller, 1982, p.69).

Favor a solvent with a phase density as different as possible from the feed and with interfacial tensions as high as possible. *Phase separation, emulsion and process control often present problems* (Keller, 1982, p.64).

Favor a solvent with low cost or great chemical and thermal stability (Keller, 1982, p.69).

Favor a solvent with low toxicity and corrosivity. *Simple disposal* (Keller, 1982, p.69).

Benzene and light petroleum fractions are often used for phenolics removal (API, 1967, p.10-5).

2.5 Summary

In this chapter, we briefly described the thermodynamics of multicomponent separations. We defined the terms: key component and separation factors. These parameters are used in the Kremser equation. The shortcut method developed by Kremser is used in the expert system, EXSEP, to test the feasibility of solvent-based separations.

In sections 2.2 to 2.4, we derived the expressions of the Kremser equation corresponding to absorption, stripping, and extraction. We also gave a non-exhaustive list of heuristics that apply to these three solvent-based separations.

CHAPTER 3

Chemical Engineering Perspective of EXSEP

In this chapter, we discuss the chemical engineering perspective of EXSEP for solving multicomponent separation problems. In section 3.1, we describe the very convenient "component assignment matrix or CAM" that EXSEP uses for the problem representation. To synthesize feasible flowsheets, EXSEP uses chemical engineering knowledge, which includes heuristics and rules of thumb (section 3-2 to 3-4). In section 3.5, we study the validity of the results given by EXSEP. We compare the results to problems found in the literature. We use also rigorous simulations (chemical engineering CAD software). Finally, in section 3.6, we discuss the limitations of EXSEP.

3.1 Component Assignment Matrix (CAM) for Problem Representation

3.1.1 Presence of the Solvent

Table 3.1 defines an example problem of absorption (Nelson, 1969, p.856). In the first column, the components present in the feed mixture are H_2 and C_3H_8 to $C_5H_{12}+$. The aim of this problem is to remove 99 % of i-C_5 and 100 % of C_5+, both of which will be absorbed by an oil (C_8 or C_9). The second and third columns are the flow rates (in mol/hr) of each component in the overhead and bottom products, respectively. The last column is the K value of each component.

Table 3.1: Problem Statement (Example of Absorption)

Components	Vapor Overhead: p1 mol/hr	Liquid Bottom: p2 mol/hr	K-value
H_2	85.59	0.0	50
C_3H_8	5.72	1.43	2.80
C_4H_{10}	0.751	0.639	1.2
n-C_4H_{10}	0.994	1.556	0.9
i-C_5H_{12}	0.013	1.327	0.37
C_5H_{12}+	0.0	1.98	0.24

EXSEP uses a component assignment matrix (CAM) to represent the problem. CAM is a PxC matrix, where P is the number of products and C is the number of components. The ij^{th} element of this matrix corresponds to the molar flow rate of the j^{th} component in the i^{th} product.

CAM 1

	H_2	C_3	iC_4	C_4	iC_5	C_5+
p2	0	1.43	0.639	1.556	1.327	1.98
p1	85.59	5.72	0.751	0.994	0.013	0

The components are in columns and the products are in rows. To state that the flow rate of H_2 to the overhead product p1 is 85.59 mol/hr, we simply write 85.59 in the first column and the second row.

The order of the components and products in the matrix is significant. Indeed, the component columns are sorted from left to right in decreasing order of K values; the product rows are arranged from top to bottom in increasing order of K values.

We define the K value of the j^{th} product as the weighted average shown by Eq. (3.1).

$$\overline{K_j}=\frac{\sum K_i F_{ij}}{\sum F_{ij}} \tag{3.1}$$

where K_i is the K value of component i, and F_{ij} the flow rate of component i in the j^{th} product.

To take into account the presence of the solvent, we add the solvent to the CAM. It comes before product p2 in the first row because the solvent for absorption is usually chosen for having a K value much smaller than those of the products. Thus, all the solvent goes to the bottom of the absorber. Moreover, to use the Kremser equation, we assume that the K value of the solvent is much lower than those of the products and tends to zero. In the same way, if we call "oil" the component of the solvent, "oil" goes after C_5+ in the CAM because it has the smallest K value among all the components.

CAM 2

	H_2	C_3	iC_4	C_4	iC_5	C_5+	oil
solvent	0	0	0	0	0	0	L
p2	0	1.43	0.639	1.556	1.327	1.98	0
p1	85.59	5.72	0.751	0.994	0.013	0	0

3.1.2 Position of Splits

The CAM is very useful to represent the possible splits. In the MSA-modules, we only test the feasibility of the split between the top and bottom products p1 and p2. This separation is represented by the horizontal line in CAM 2. Every product above the split line will go to the

bottom (i.e. p2 and the solvent) and the product(s) below the line goes to the top of the column. In the CAM, an horizontal line represents a sloppy split or non-sharp, product split, whereas a vertical line represents a sharp, component split (a pure component goes in one of the products).

If we consider the submatrix CAM 3, which is the bottom product of the absorption column, the vertical dashed line illustrates the solvent recovery column where the oil is separated from the other components.

CAM 3

	H_2	C_3	iC_4	C_4	iC_5	C_5+	Oil
solvent	0	0	0	0	0	0	L
p2	0	1.43	0.639	1.556	1.327	1.98	0

3.2 Heuristics for Feasibility Analysis

As shown in Figure 3.1, the goal of the MSA-modules (absorption in this case) is to test if the split between the top (p1) and bottom (p2) products is thermodynamically and economically feasible and to find the number of stages and the solvent flow rate. To evaluate the thermodynamical feasibility, EXSEP uses the K values of the components and the Kremser equation to estimate the component-recovery ratios. In addition, EXSEP uses heuristics to find the "optimum" number of stages and solvent flow rates.

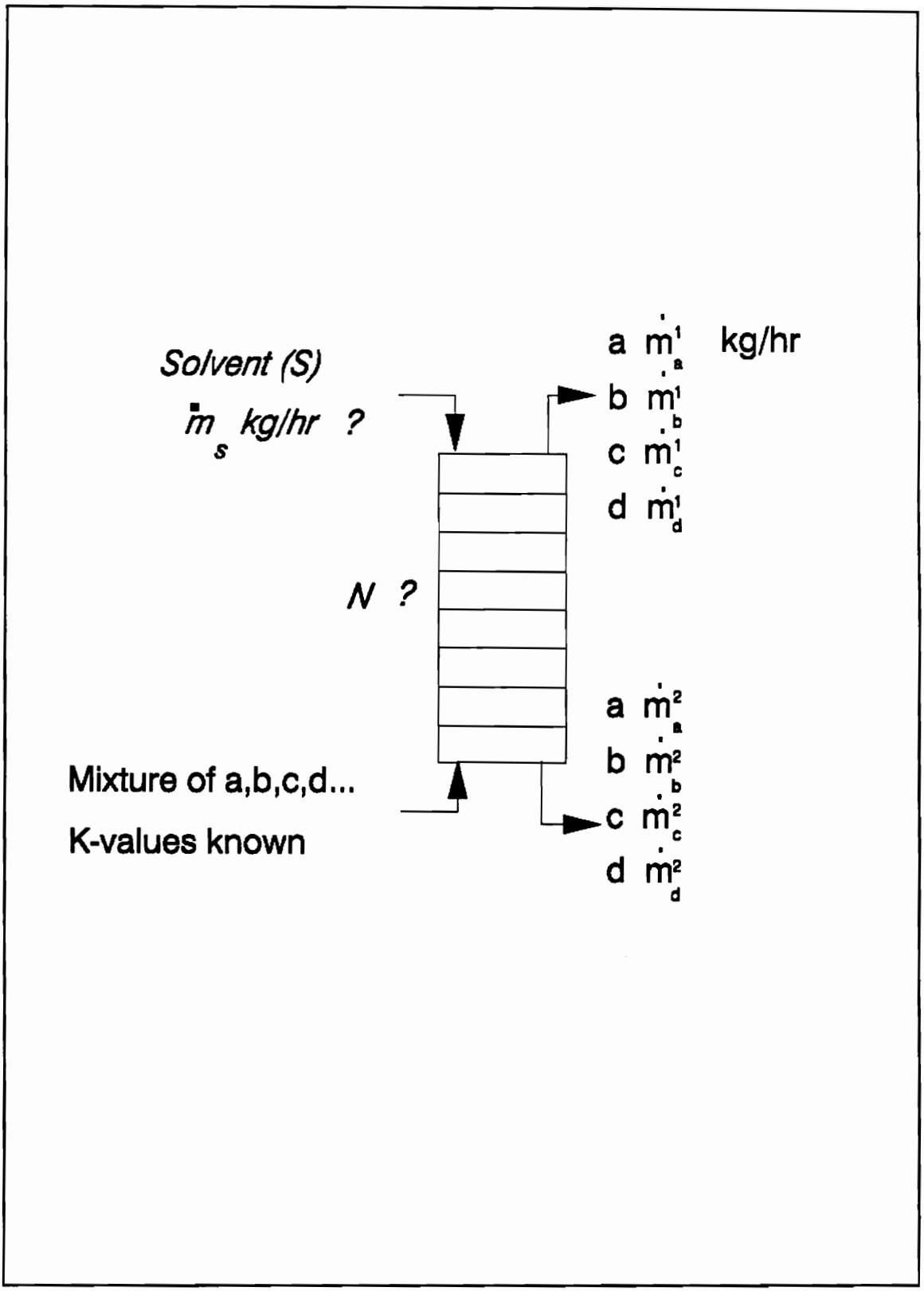


Figure 3.1 : Example of Absorption.

3.2.1 Number of Stages

Keller (1982, p.50) presents a heuristic chart (Fig. 3.2) where the X-axis is the number of theoretical stages of an absorption column. The first Y-axis is the ratio of the solvent flow rate and the quantity of CH_4 recovered in an absorption column. The second Y-axis is the energy required to pump the solvent. For a number of stages, N, of about 5, both the solvent flow rate and the energy required are minimum. This property explains the limitation of the number of stages given by the heuristics (see section 2.2.3). Indeed, the Kremser equation advises one to build a column with a number of stages greater than five.

In addition, it is obvious that the higher the column, the more expensive it is to build. N should be greater than five, but not too much greater. This leads to an optimal value of five for the number of stages of an absorption column. The same conclusion applies also to stripping columns. For extraction, the upper limit seems to be 10 stages, because of high construction cost and operating problems, and the lower limit is about 4 stages.

3.2.2 Separation Factor

Douglas (1988, p.77) and Treybal (1980, p.291) give an optimal value for the absorption factor ($A_{opt} = 1.4$). We use this value in EXSEP to generate initial flowsheets. A high solvent flow rate L ($L = A.k.V$, so L is proportional to A) raises solvent-recovery costs. A low value of L requires more equilibrium stages and increases absorber cost.

We use both heuristics on the separation factor and the number of theoretical stages even if they seem equivalent. Indeed, the two design variables can be optimized separately. For a given number of stages and its corresponding absorption factor A, if A is smaller the A_{opt}, EXSEP tests the feasibility of the separation. When this separation is infeasible, we impose a new value of A being equal to A_{opt}. This leads to better component-recovery ratios (see section 3.2.3). For more

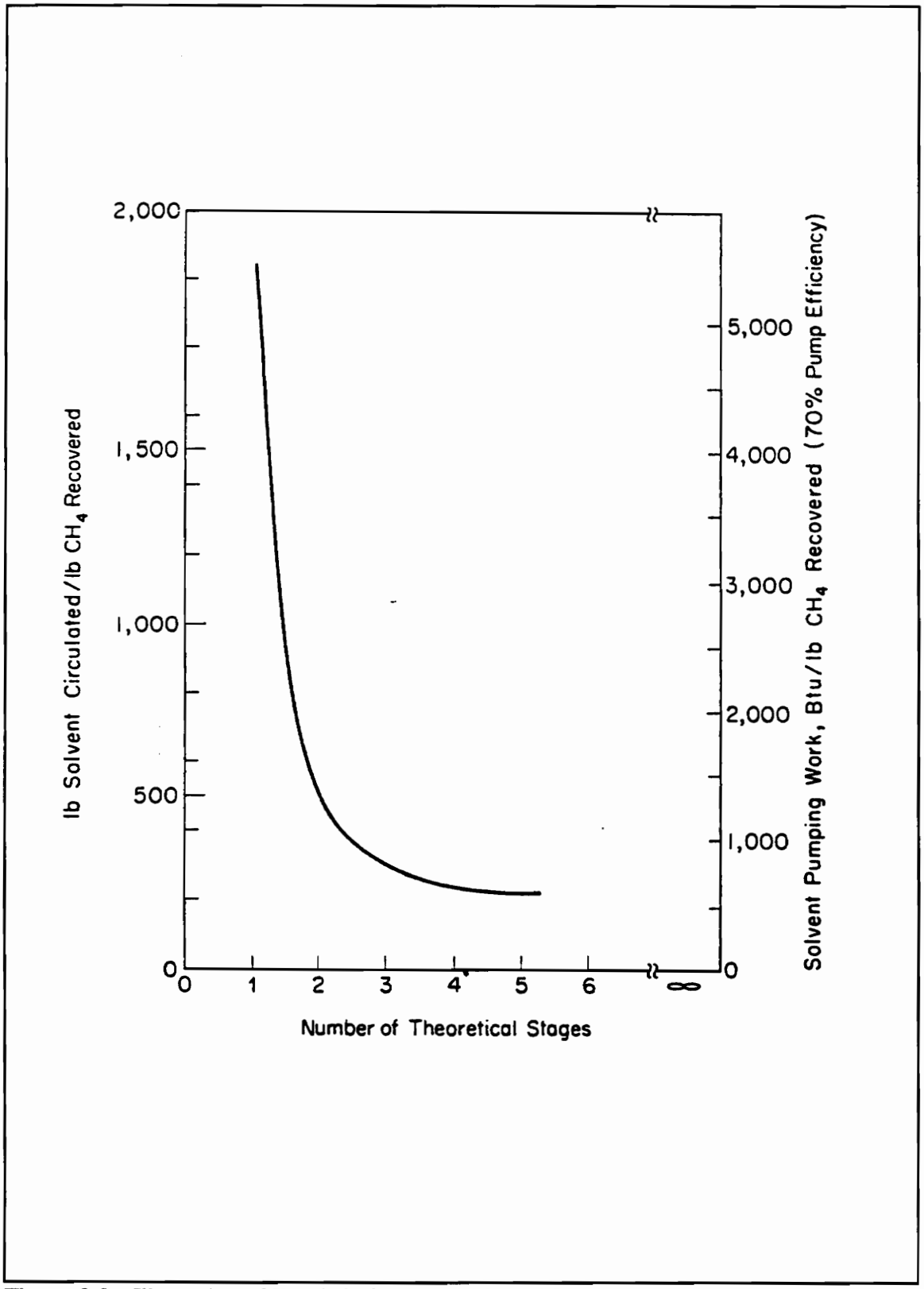


Figure 3.2 : Illustration of Heuristic for the Number of Stages (Keller, 1982, p.50).

details, see section 5.3.2.1.

3.2.3 Component-Recovery Ratios

The component-recovery ratio d/b is defined by the molar fraction of product i going to the top over the molar fraction of product i going to the bottom. For example, the flow rate of i to the top is f_i^t=90 mol/hr, and the flow rate of i to the bottom is f_i^b=10 mol/hr. Then d/b=0.9/0.1:

$$f_i^t=90\ ,\ f_i^b=10\ ,\ f_i^t+f_i^b=100$$

$$d=\frac{90}{100}=0.9$$

$$b=\frac{10}{100}=0.1 \qquad (3.2)$$

$$\frac{d}{b}=\frac{0.9}{0.1}$$

The rules to calculate the component-recovery ratios are the following:

- *If d=0 then d/b=0.02* **We always assume that at least 2% of a component goes to the top.**
- *If b=0 then d/b=49.0* **We always assume that at least 2% of a component goes to the bottom.**
- *If d≠0 and b≠0 then d/b=d÷b*

For a solution to be feasible, we ask the deviation between the specified and actual ratio for every component to be less than $\pm 10\%$. That means the two conditions are:

$$0.9\left(\frac{d}{b}\right)_{specified} < \left(\frac{d}{b}\right)_{actual} < 1.1\left(\frac{d}{b}\right)_{specified} \qquad (3.3)$$

3.3 Rules for Explanation Facility

3.3.1 Four Characteristic Factor Definitions

Figure 3.3 shows the position of a solution in a plot of N (number of stages) versus A (absorption factor).

To find this solution, EXSEP starts the calculations with the absorption factor equal to a minimal (or initial) value called $\mathbf{A_{min}}$. Then, from left to right on Fig. 3.3, EXSEP increments A, and performs the feasibility analysis for each incremental step, until the separation factor reaches the upper limit called $\mathbf{A_{sl}}$ (sl stands for solvent because this upper limit depends on the solvent).

$\mathbf{A_{opt}}$ is the optimal value (=1.4) coming from heuristics. We also define $\mathbf{A_{st}}$ (for stage) which is the value of the absorption factor when the number of stages is exactly equal to the optimal number of stages $\mathbf{N_{opt}}$ (=5, for absorption).

We call these factors (A_{min}, A_{opt}, A_{st}, and A_{sl}) the four characteristic factors. In the same way, we define S_{min}, S_{opt}, S_{st}, and S_{lg} (for lean gas) for stripping, and E_{min}, E_{opt}, E_{st}, and E_{sl} for extraction. These factors, except A_{st}, are normally constant and set according to heuristics in EXSEP. But the flexibility of our expert system enables the user to change these values to suit an exceptional problem statement.

3.3.2 Solution Position

The "quality" of a solution depends on its position among the four characteristic factors. For example, if a solution (black dot in Figure 3.3) has an absorption factor A close to A_{opt}, lower than A_{opt} and close to N_{opt}, as shown in Figure 3.3, then the solution is satisfactory because both A and N are optimum.

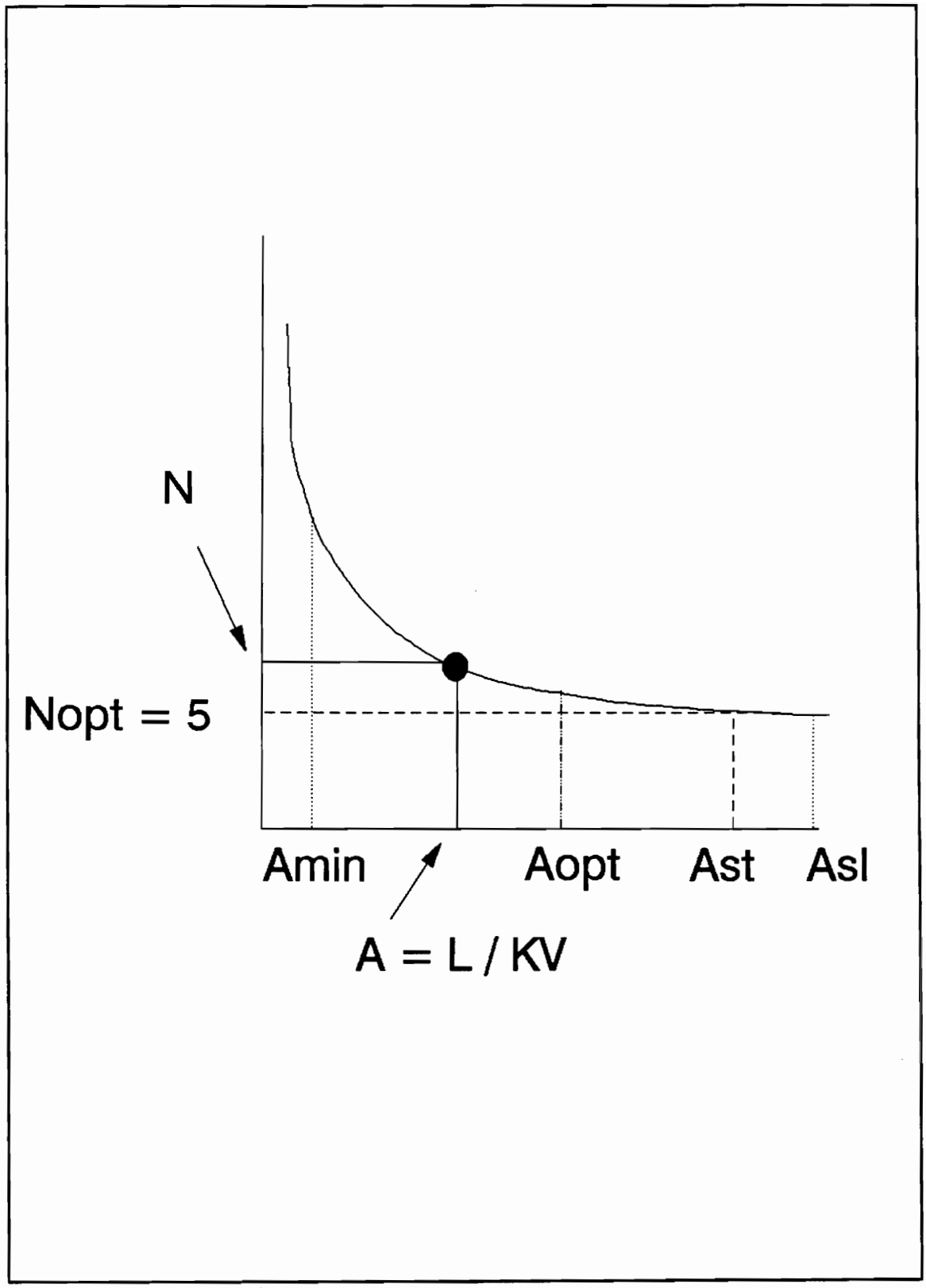


Figure 3.3 : Example of Solution

We summarize the conditions for a satisfactory solution:

if

$A \approx A_{opt}$, $N \approx N_{opt}$ *and* $A \geq A_{st}$

then heuristics satisified:

A is optimum and $N \geq N_{opt}$

For each possible position of a solution, EXSEP will give an explanation or diagnosis (see sections 4.1.5 and 5.4.1.1)

3.3.3 Search-Range Size

The existence of a solution depends not only on the size of the search space (range A_{min}-A_{sl}), but also on the relative positions of the four factors. If A_{sl} is smaller than A_{min}, no calculation will occur as EXSEP starts incrementing from A_{min} to A_{sl}. There are 24 configurations of these four A factors (see section 5.4.1.2). Each of them is taken into account by EXSEP, which gives explanations concerning the locations of possible solutions in the search space.

Let us consider an example. In Figure 3.4, A_{sl} is smaller than A_{st}. This means that EXSEP will stop the search at A_{sl}, where the corresponding number of stages will still be higher than the optimum number of stages. Thus, a solution may exist with N exactly being equal to 5 (optimum) which would be better than another solution inside the search space with N greater than 5 (ignoring the heuristics of the absorption factor).

To summarize, if $A_{sl} < A_{st}$, there may exist feasible solutions with N still higher than 5. In that case, EXSEP advises the user to increase A_{sl}.

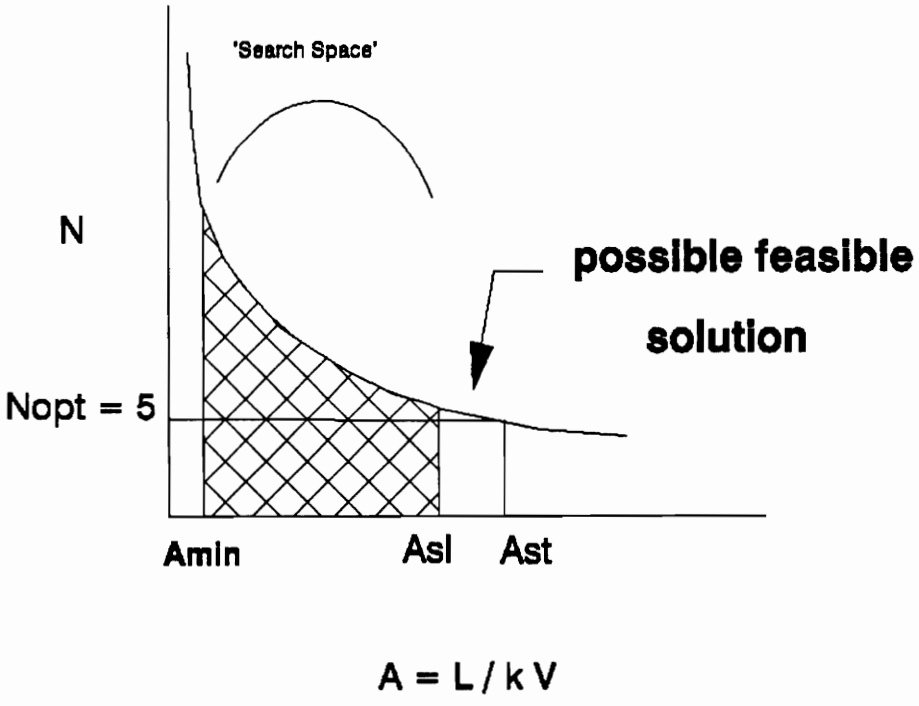


Figure 3.4 : Example of Search-Range Explanation

3.4 Selection of the Best Solution

EXSEP generates several feasible solutions. Each solution has its intrinsic quality, which depends on its separation factor, number of stages, and deviation between the actual and specified component-recovery ratios. We try to find solutions with N as close as possible to N_{opt}, but if not equal to, higher than N_{opt}. We also seek solutions with a separation factor, e.g. A, as close as possible to the optimal value A_{opt} and if not equal to, then smaller than the optimum (to minimize the solvent flow rate). For example, we prefer 6 stages than 4 stages, because the construction cost will not be that different, whereas the economy of solvent and the performance with 6 stages are better. Finally, we want the deviation between the actual and specified component-recovery ratios to be minimum.

3.4.1 Coefficient of Separation

To satisfy these specifications, we developed an empirical function called CS for Coefficient of Separation.

We looked for functions which would satisfy the characteristics shown by Eq. (3.5). We choose to add the properties of three functions, one for each parameter. The presence of the logarithm is the result of this addition of functions. We take the exponential of the all sum to scale the CS factor. For *Dev*, the choice is the inverse function (1/x). For N and S, a function satisfying the properties of having a maximum in the optimum value is a parabola. To satisfy the other property (CS(N+1) greater than CS(N-1)), we used the arc-tangent function.

The derivation of Eq. (3.6) is in section A.3.

$$\left(\frac{\partial CS(Dev,N,S)}{\partial N}\right)_{N=Nopt}=0 \quad and\ CS(N+1)>CS(N-1)$$
$$\left(\frac{\partial CS(Dev,N,S)}{\partial S}\right)_{S=Sopt}=0 \quad and\ CS(S+1)<CS(S-1) \tag{3.5}$$
$$if\ Dev\neq 0$$
$$\left(\frac{\partial CS(Dev,N,S)}{\partial Dev}\right)<0$$

where:

Dev is the average deviation between the actual and expected component-recovery ratios.

N is the number of stages.

$\mathbf{N_{opt}}$ is the optimal number of stages.

S is the stripping factor (KV/L).

$\mathbf{S_{opt}}$ is the optimal stripping factor.

$$CS(Dev,N,S)=\exp[-\ln(Dev) -\ln\{\frac{N^2}{2}-(N_{opt}-2)N+\frac{(N_{opt}-2)^2}{2}+1\}+2\sqrt{2}\arctan\{\frac{N-(N_{opt}-2)}{\sqrt{2}}\} -\ln\{\frac{S^2}{2}-(S_{opt}+2)S+\frac{(S_{opt}+2)^2}{2}+1\}-2\sqrt{2}\arctan\{\frac{S-(S_{opt}+2)}{\sqrt{2}}\}\ ] \tag{3.6}$$

This coefficient has **no physical meaning,** but it gives useful results. Figure 3.5 shows a plot of CS(N). Keeping S=0.71 (optimum) and the deviation Dev=1%. We notice that CS is maximum when $N=N_{opt}$ and that CS(6) > CS(4).

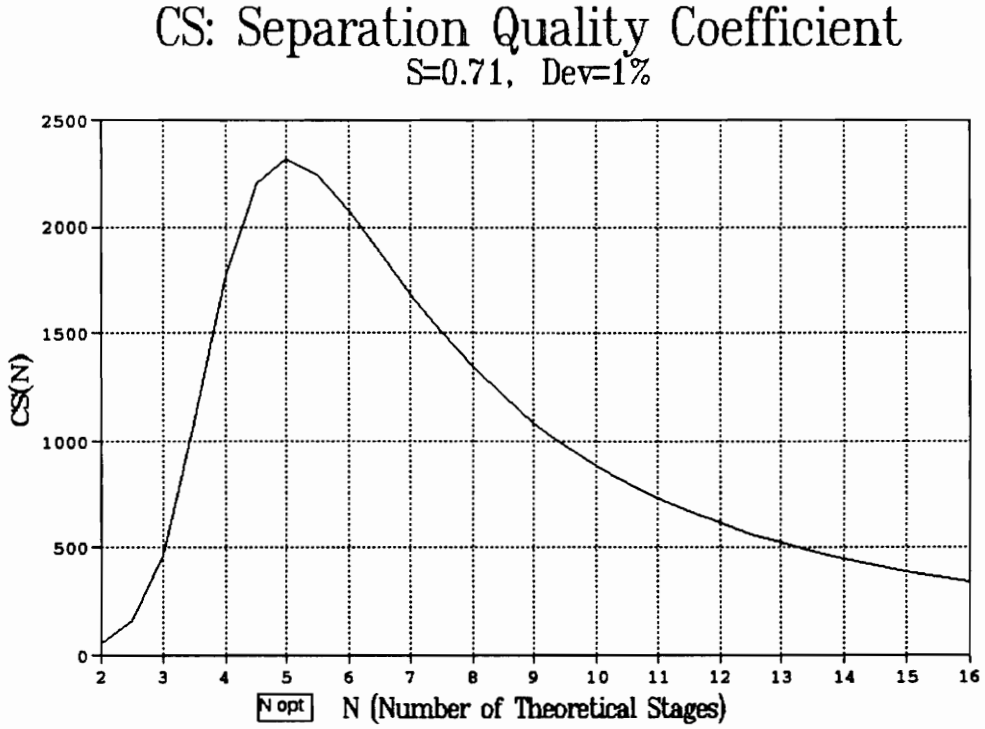


Figure 3.5 : Coefficient of Separation (CS) versus the Number of Stages (N).

3.4.2 Final Selection

When all feasible solutions are known, EXSEP calculates their respective CS values and sorts them in decreasing CS order. Then, EXSEP indicates the best solution according to the CS ranking.

3.5 Validity and Consistency of the Solution

Most of the examples tested by EXSEP come from books where the authors give their own solutions using shortcut methods. Usually, the approach they use is to choose the number of stages and then change the solvent flow rate until the component-recovery ratios converge to the specified ratios. EXSEP has no idea of the number of stages or the solvent flow rate when it starts its search. In most examples tested, the results of EXSEP (number of stages and solvent flow rates) are consistent with the results given in the literature.

From the solutions provided by EXSEP, we use another mean of verification by running rigorous simulations with the CAD software Design II (Chemshare, 1985). These simulations show that the approximate flow rates are close to the actual flow rates, which makes EXSEP reliable for the prediction of the flow rates. In addition, the design variables (N and the solvent flow rate) given by EXSEP provide the user a preliminary flowsheet for rigorous design by CAD simulations.

3.5.1 Comparison with Other Shortcut Methods

3.5.1.1 Absorption

We test an example of absorption (Nelson, 1969, p.856) with EXSEP. The problem is to absorb 99 % of iC5 and 100 % of C5+ with an oil from a mixture of H2 (85.6 %), C3 (7.15

%), iC4 (1.39 %), C4 (2.55 %), iC5 (1.34 %) and C5+ (1.98 %). Nelson uses the Kremser equation as we do in EXSEP. Table 3.2 shows the design variables. Nelson starts by assuming 8 stages and then calculates the absorption factor to match the component flow rates in the top and the bottom products.

In EXSEP, we do not need to assume the number of stages. EXSEP finds both the absorption factor (or solvent flow rate) and the number of stages.

Table 3.2: Design Variable Comparison

	Number of stages N	**Solvent Flow rate L (mol/hr)**	**Absorption Factor A**
EXSEP	8.6	55.5	1.5
NELSON	8	55.5	1.5

The results given by EXSEP are very consistent with the results given by Nelson. In Table 3.3, we compare the flow rates estimated by EXSEP and Nelson.

Table 3.3: Flow Rate Comparison

		EXSEP		NELSON	
Component	Vapor Feed mol/hr	Vapor Product mol/hr	Percent Absorbed %	Vapor Product mol/hr	Percent Absorbed %
H_2	85.59	83.88	2	85.59	0
C_3	7.15	5.73	20	5.73	20
i-C_4	1.39	0.75	46	0.75	46
C_4	2.55	0.99	61	0.99	61
i-C_5	1.34	0.03	98	8.5×10^{-3}	99
C_5+	1.98	0.04	98	1.1×10^{-3}	100
Oil	0	0	---	0	---

The differences come from our assumption that at least 2% of light products goes to the bottom and at least 2% of heavy products goes to the top (see section 3.2.3). This rule, used in EXSEP, makes the approximate simulation closer to reality. We discuss this point more in Table 3.8 in section 3.5.2.

3.5.1.2 Stripping

In Table 3.4, we compare the results of two different shortcut methods for a stripping example (Henley and Seader, 1981, p.470). The problem consists of the separation of butadiene sulfone from a mixture of sulfur dioxide (8.3%), butadiene-3 (7%), butadiene-2 (2%) and butadiene sulfone (84%), the first three being stripped with nitrogen.

For their solution, Henley and Seader did not use the Kremser equation, but rather the Edmister equation. The main characteristic of the Edmister equation is to evaluate the stripping factor at different places in the column, whereas the Kremser equation uses only one stripping factor, which is assumed to be constant throughout a column. We observe in Table 3.4 that the numbers of stages are similar. The stripping factors, and thus the solvent (lean-gas) flow rates from both shortcut methods, are different. The deviation between the stripping factors is 48%, but the deviation between the flow rates is only 19%.

Table 3.4: Design Variable Comparison

Method	N Number of Stages	S Stripping factor	V lean gas flow rate mol/hr
EXSEP	5.3	1.05	54.08
Henley & Seader	5	2.04 (average)	45.4

In Table 3.5, we compare the estimated flow rates given by the two shortcut methods. We observe that even with a higher lean-gas flow rate, EXSEP underestimates the amount stripped.

Table 3.5: Flow Rate Comparison

	FEED	EXSEP		Henley & Seader	
	mol/hr	Top mol/hr	% stripped	Top mol/hr	% stripped
N_2	----	**54.08** (feed)	---	**45.4** (feed)	---
SO_2	10	9.8	98	9.97	99.7
B_3	8	7.75	97	7.86	98.3
B_2	2	1.72	86	1.85	92.5
BS	100	2	2	0.75	0

The discrepancies come from the use of the two different methods. However, we prefer the Kremser method because it is closer to the philosophy of expert systems, which uses systematic knowledge representation, and heuristics search, instead of a more complex, but slowly converging algorithm.

3.5.1.3 Extraction

The example of extraction comes from the "ProII Example Manual" (Simulation Sciences, Inc., 1987, p.C5). It consists of the purification of an aqueous effluent containing traces of benzene (0.06%), benzoic acid (0.0001%) and phenol (1.17%). The extracting solvent is a mixture of benzene (99.499%), phenol(0.001%) and water(0.5%).

In Table 3.6, we do not exactly compare two shortcut methods. Instead, we compare the result from EXSEP with the problem statement of a rigorous simulation, where the number of

stages and solvent flow rate are the input data. To find these data, the author of the manual had to evaluate them like EXSEP does. The two sets are quite different. EXSEP results may be better because for only 2-3 more stages, the flow rate is cut by 42 %. A precise economic analysis would be able to choose between the two.

Table 3.6: Comparison of Design Variables

	Rigorous (ProII Example)	**EXSEP**
N	4	6.6
V	135.25	78.06
E	2.58	2

EXSEP gives other solutions, for example, Table 3.7. This solution is much closer to ProII solution, but it is not considered to be the best one by EXSEP, partly because N equals, but is not greater than, the optimum (4 here) and because E is much higher than the optimum (0.8 for extraction). At least EXSEP is able to find this solution which the user can further evaluate. This example shows how efficient EXSEP is to generate multiple flowsheet solutions, and how flexible it is when the user wants to reject the estimated best solution.

Table 3.7: Design Variables

	EXSEP
N	4
V	136.7
E	3.5

3.5.2 Comparison with Rigorous Design

3.5.2.1 Absorption

Table 3.8 compares the approximate flow rates estimated by EXSEP and the exact flow rates calculated with a rigorous simulation (with Design II). EXSEP tends to overestimate the absorption of the light components and underestimate the absorption of the heavy ones.

Table 3.8: Flow Rate Comparison

		EXSEP		RIGOROUS	
Component	Vapor Feed mol/hr	Vapor Product mol/hr	Percent Absorbed %	Vapor Product mol/hr	Percent Absorbed %
H_2	85.59	83.88	2	85.55	0.1
C_3	7.15	5.73	20	5.87	18
i-C_4	1.39	0.75	46	0.80	42
C_4	2.55	0.99	61	0.93	64
i-C_5	1.34	0.03	98	8.5×10^{-3}	99
C_5+	1.98	0.04	98	1.1×10^{-3}	100
Oil	0	0	---	2.5×10^{-3}	---

We observe that the comparison is done between two different kinds of simulations. Indeed, the results provided by EXSEP, i.e., the number of stages and solvent flow rates, are input data for the rigorous simulation.

3.5.2.2 Stripping

To examine the validity of the STRIPPING module of EXSEP, we use a new example. This stripping problem is the recovery column of the absorption problem previously studied. The

recovery process is inspired from an example where Smith (1963, p.270) uses steam to recover an oil. This oil is the absorption solvent of heavy components of a mixture of methane to pentane. The bottom of the absorber (sections 3.5.1.1 and 3.5.2.1) becomes the inlet stream of a steam stripper. The design variables (number of stages=5, and steam flow rate=80.61 mol/hr) are used to run a rigorous simulation with Design II.

In Figure 3.6 we compare the results. The first set of bars represents the feed to the column, in terms of component-recovery ratios. For example, 100% of each component enters the top of the column, whereas the steam enters the bottom.

The second set of bars is the results given by EXSEP. For example, 84% of the 1.98 mol/hr of C5H12+ entering the column goes to the top of the column, and the remaining 16% (0.32 mol/hr) goes to the bottom. The third set is the result from rigorous simulation (Design II).

We can compare the approximate design and rigorous simulation by looking at the size of the bars, the feed in both methods being the same. EXSEP tends to overpredict the amount of light key going to the bottom. If we look closely to the first bar of the third series, we observe that some steam goes to the bottom in the rigorous simulation and not in EXSEP. The reason is that we assume the K value of steam to be infinite when using EXSEP. In reality, it is not.

EXSEP gives a very good approximation of the actual flow rates. It provides realistic design variables. Indeed, the rigorous simulation proves that the separation given by EXSEP is feasible. Moreover, this preliminary design is necessary, because the user of any CAD software (e.g., DesignII, PROII, or ASPEN PLUS) must have an initial flowsheet prior to its rigorous simulation with the software.

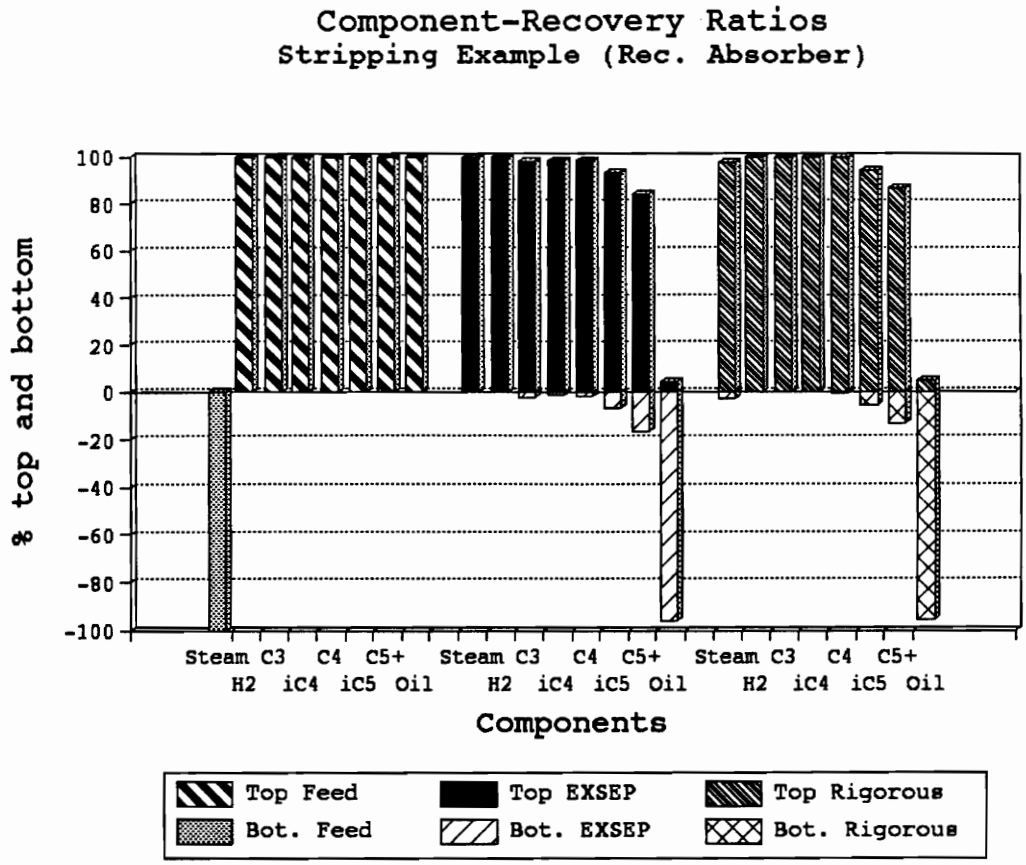


Figure 3.6: Example of Stripping (Recovery of Oil)

3.5.2.3 Extraction

Figure 3.7 compares the feed, and EXSEP and Design II component-recovery ratios. Again, EXSEP overestimates the flow rates of the light components. In this example, the cut is really sharp. The overestimation from EXSEP comes from the assumption we made in section 3.2.3. We always assume that at least 2% of a component goes to the bottom of the separator.

For extraction, the approximate flow rates predicted by EXSEP are reliable, as they were in absorption and stripping.

3.5.3 Verification of the Assumptions Used in the Kremser Equation

As explained in section 2.1.4.3, many assumptions concerning the Kremser equation must be satisfied to be able to use this shortcut method. Using the stage by stage provided by rigorous simulation (e.g. temperature and K-values profiles) for several examples, we could verify the assumptions necessary to run EXSEP simulations (use of the Kremser equation).

- We do not have exactly *isothermal and isobaric conditions* (Fig. 3.8).
- *Negligible heat of absorption (or stripping)* is not satisfied. Otherwise, the conditions would be isothermal and isobaric (Fig. 3.8 and Fig. 3.9a). Usually, this assumption is satisfied for extraction (In Fig. 3.9b, the variation is only 0.1 °C).
- *The absorbed (or stripped or extracted) components are in low concentration (solution must be dilute).*
- *Henry's law constants are constant throughout the column,* is not often satisfied.

In Figure 3.10, we show the variation of the K values for the absorption example. We

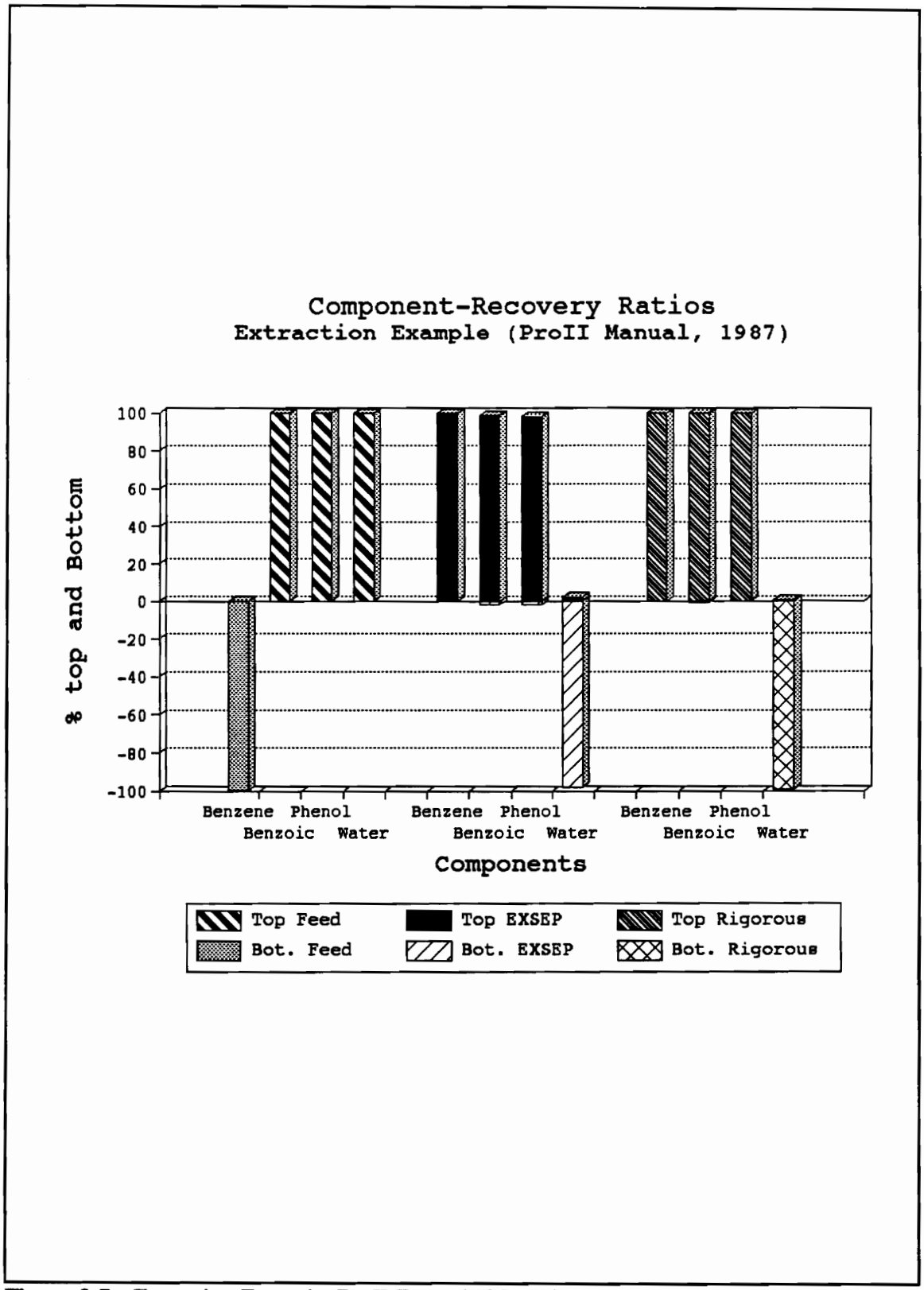


Figure 3.7 : Extraction Example (ProII Example Manual, 1987).

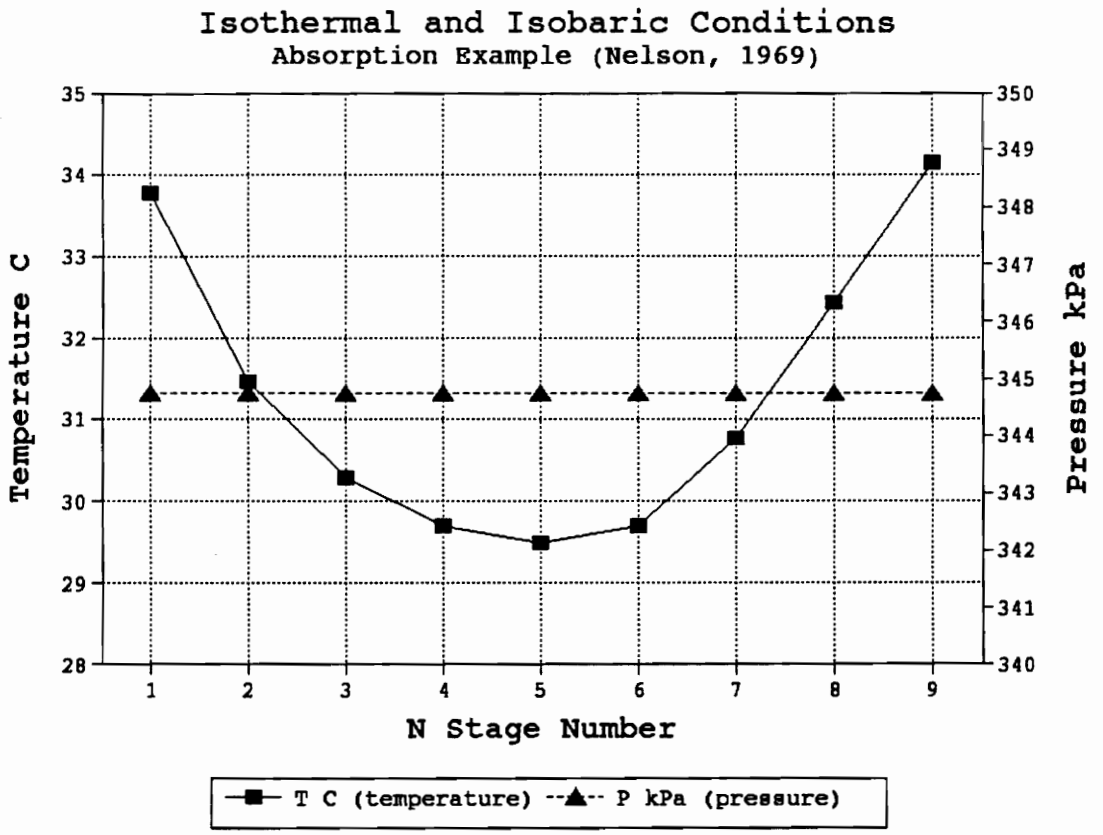
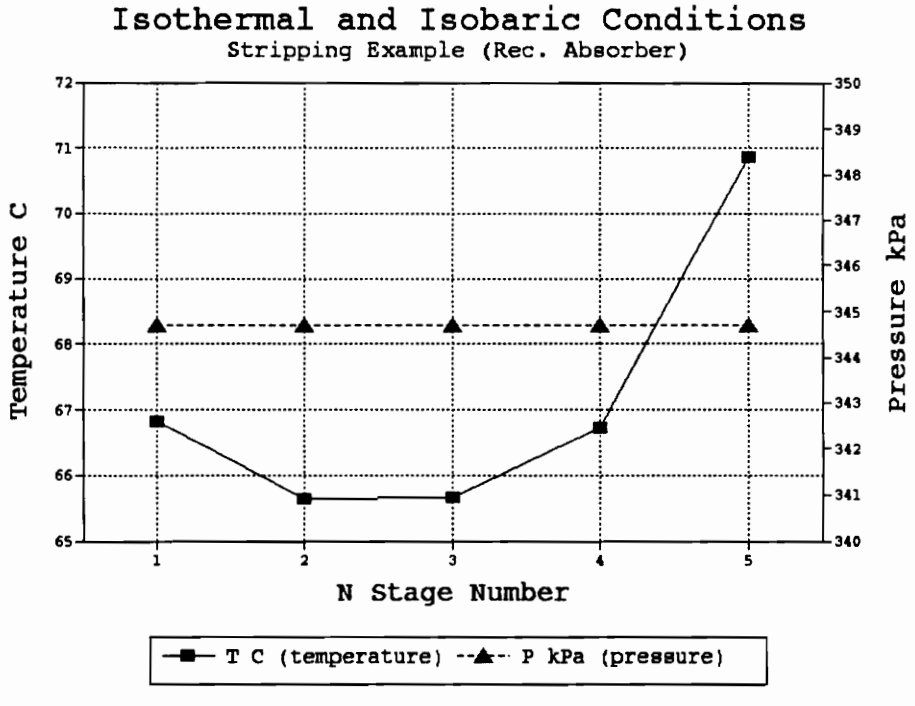


Figure 3.8 : Isothermal and Isobaric Conditions (Absorption)



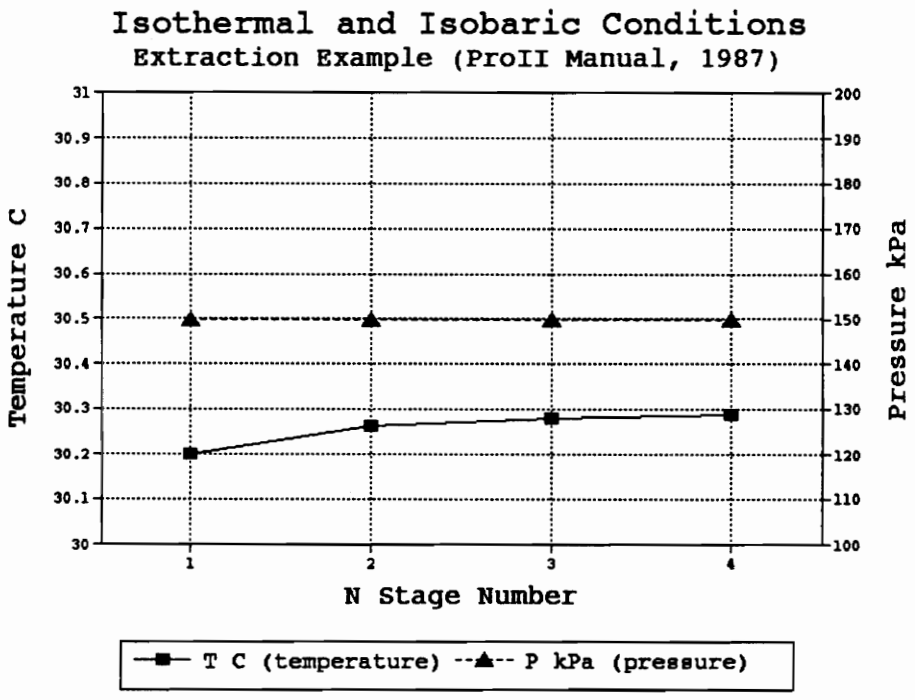
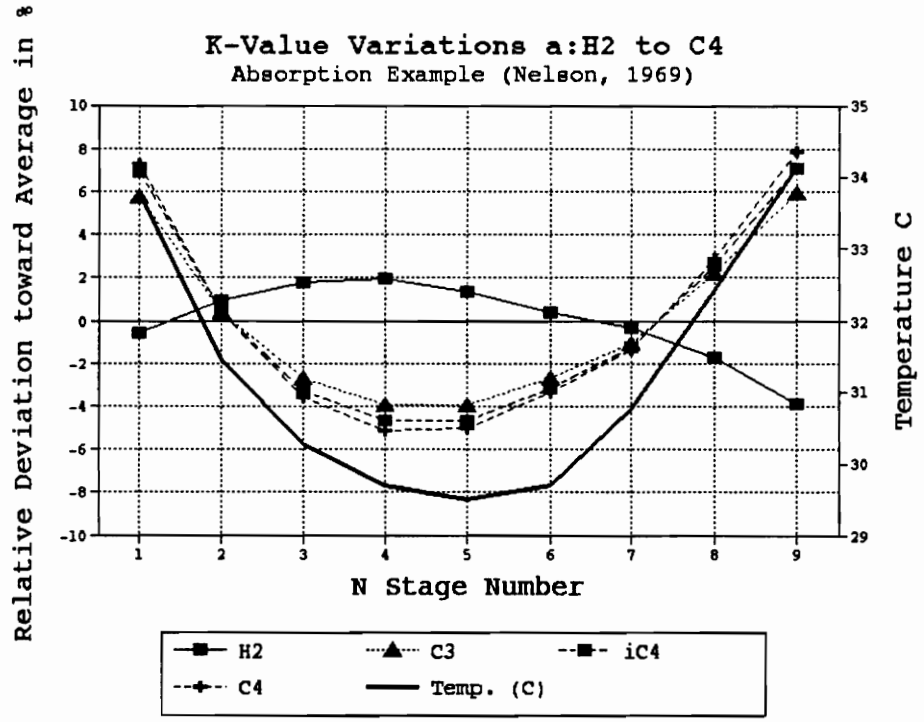


Figure 3.9a-b : Isothermal and Isobaric Conditions. (a) Stripping, (b) Extraction

use a relative deviation to compare the data, because the K values range from 10^3 to 10^{-3}. We calculate the ratio of the actual K value at one stage to the average value for the 9 stages and plot the deviation toward this average. For example, if the K value of H_2 at stage 1 is 1200, the average K value is 1121, then the deviation is:

$$\frac{(1200-1121)}{1121}=7\% \tag{3.7}$$

Fig. 3.11 and 3.12 show the same kind of plots for stripping and extraction, respectively.



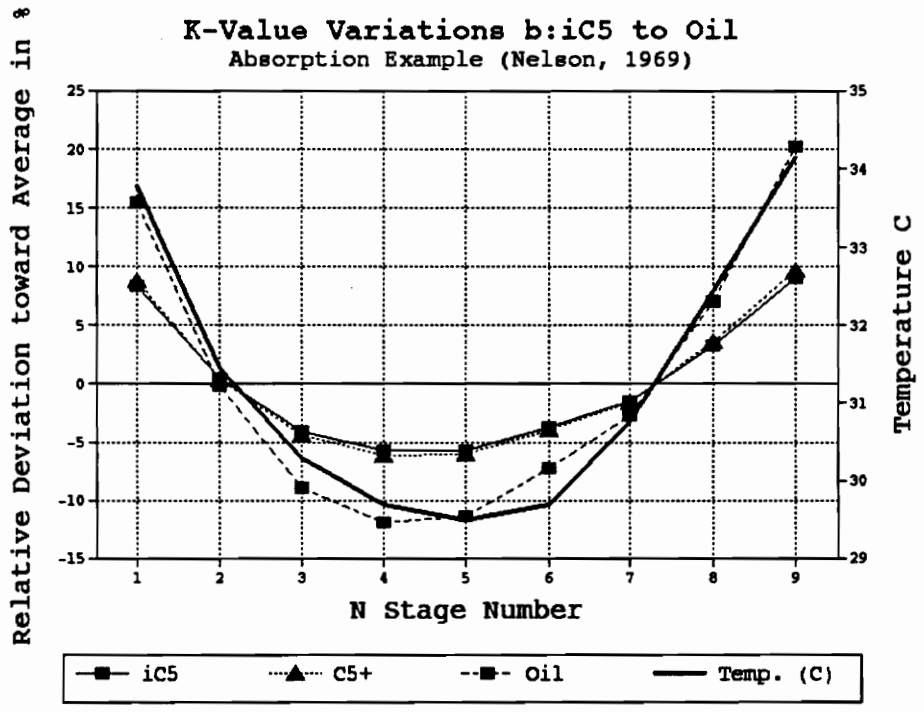
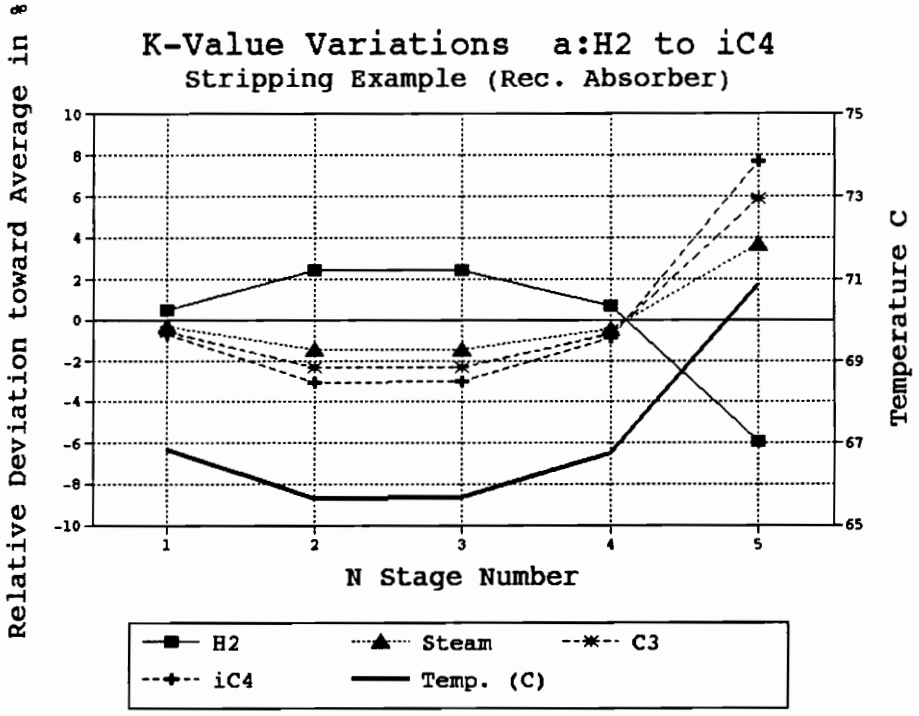


Figure 3.10a-b : Constant K Values Assumption. Absorption Example.



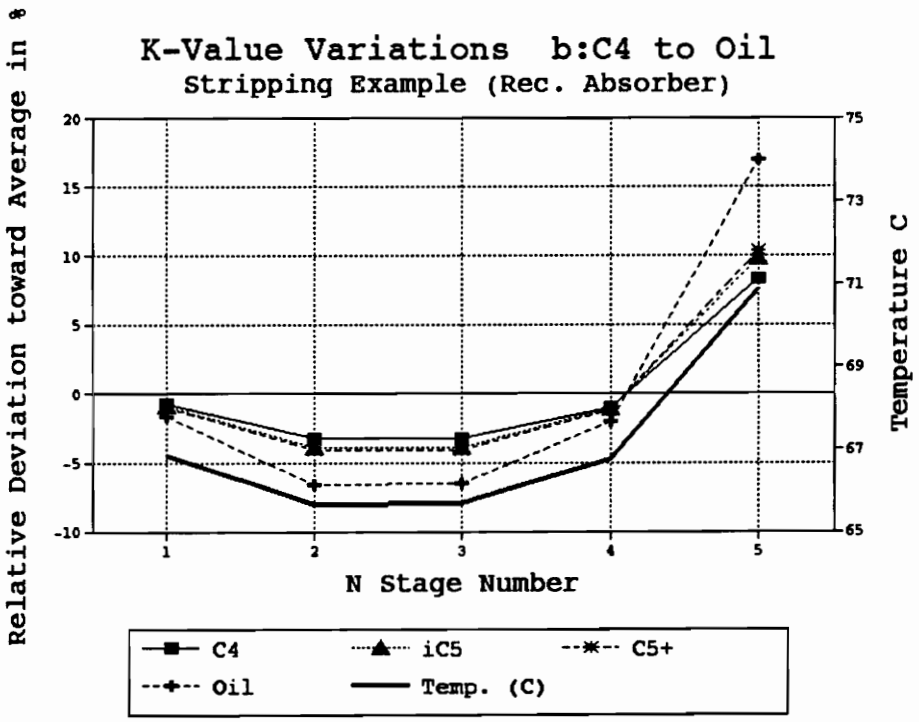


Figure 3.11a-b : Constant K-Value Assumption. Stripping Example.

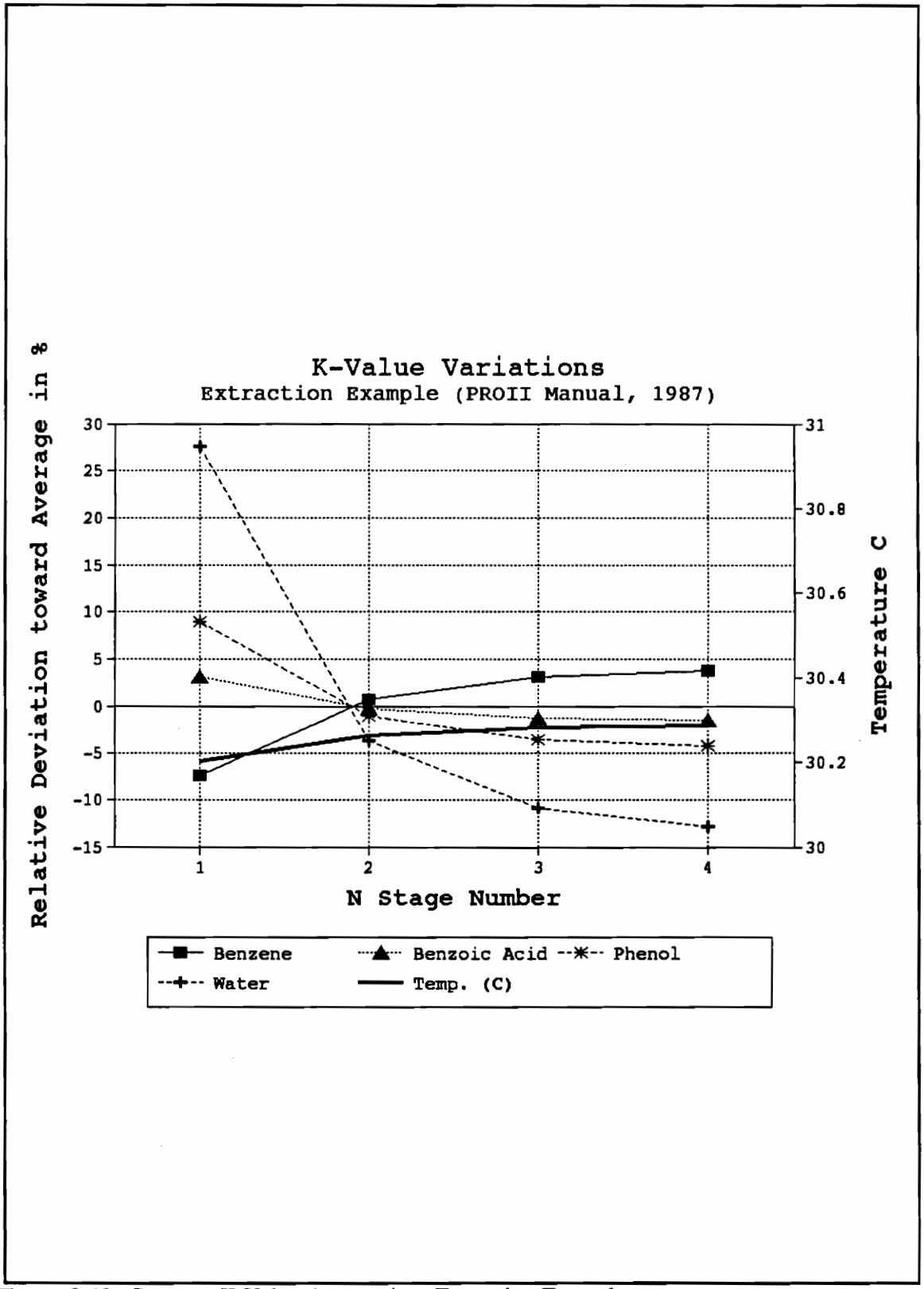


Figure 3.12: Constant K-Value Assumption. Extraction Example.

• *If the molal overflow (L/V) is constant, then the separation factors are constant (the K value being constant) throughout the column, because the amount transferred from one phase to the other is small.*

Fig. 3.13 and 3.14 contradict this assumption for absorption, and for stripping and extraction, respectively.

The favorable results, when compared to those obtained from rigorous CAD simulation in the previous sections, clearly suggest that EXSEP is able to give acceptable preliminary-design results even when the assumptions required to use the Kremser equation are not exactly satisfied. **But experience shows that if the "dilute solution" assumption is not satisfied, EXSEP is unable to provide any result (see an example in section 8.1.3.2).**

3.6 Limitations of EXSEP

EXSEP cannot solve all MSA-based separation problems.

There exist two main limitations:

• The components to be separated **must be in dilute concentrations** (rule of thumb: concentration < 10%).

• The **MSA cannot be a dominating component**. For instance, steam stripping of sour water is difficult to handle by EXSEP, because the lean gas is water and the main component of the feed (often more than 90 %) is also water. In section A.1.2.4, we describe an example.

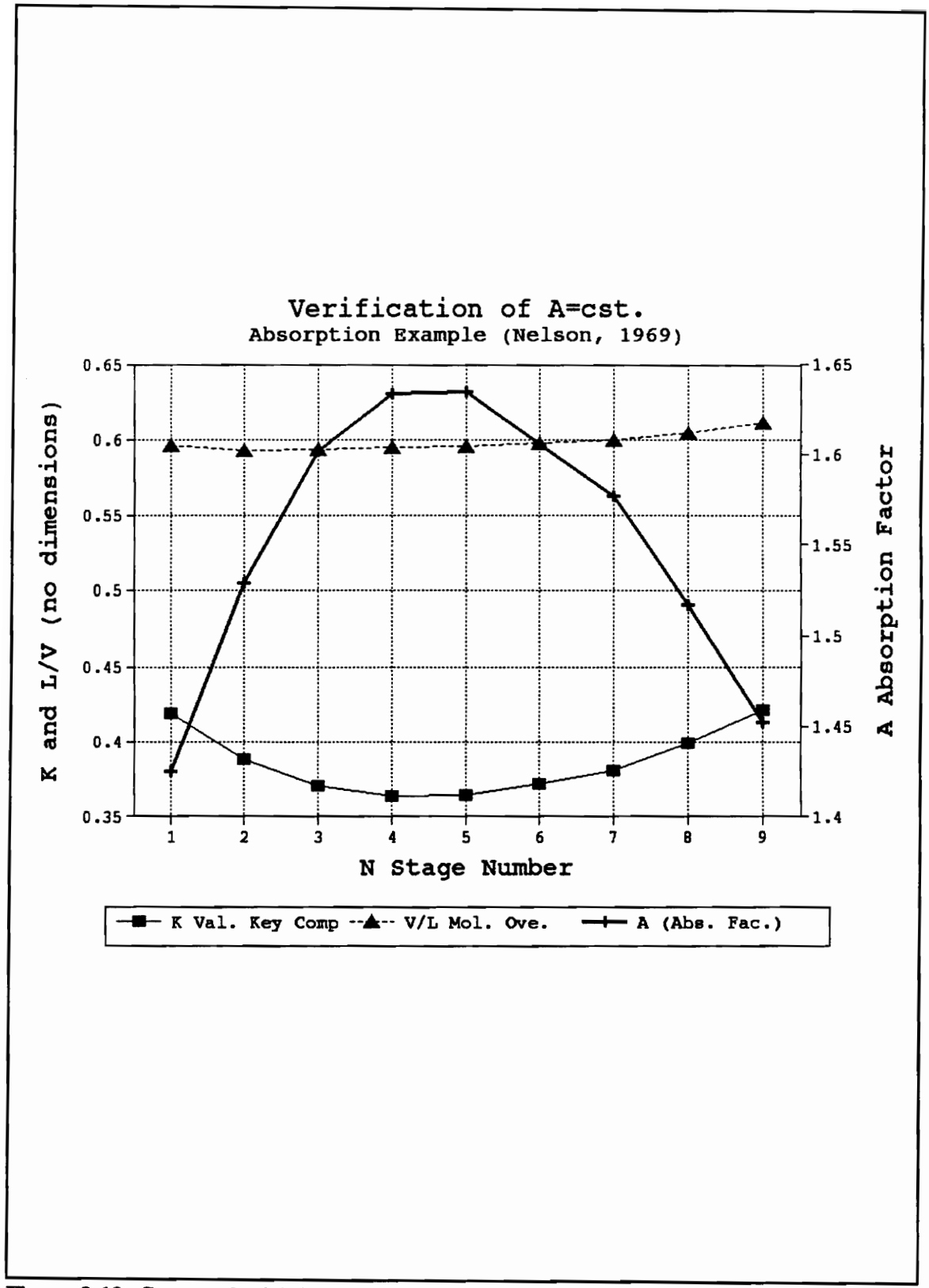


Figure 3.13: Constant Molal Overflow. Absorption Example.

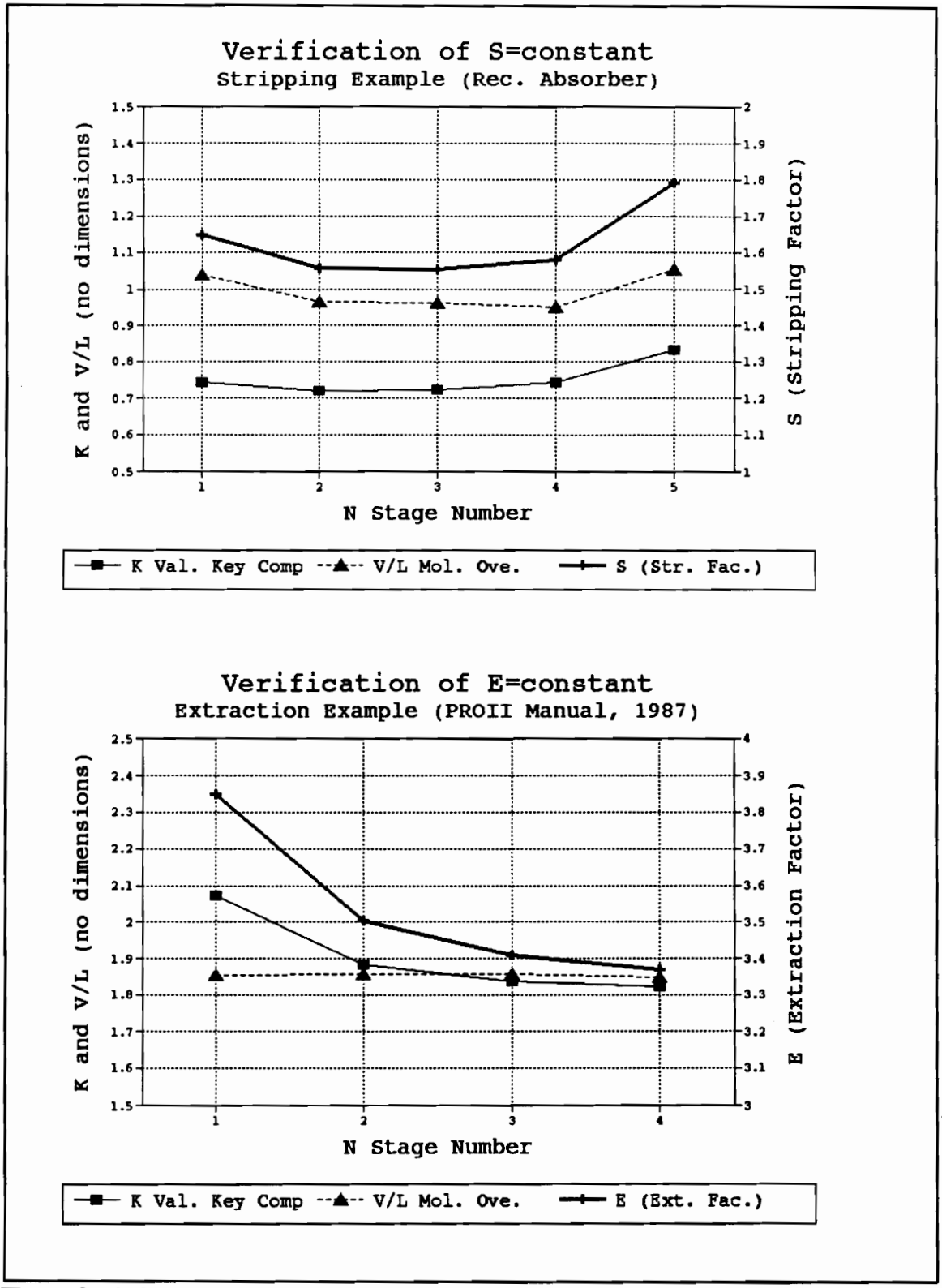


Figure 3.14a-b : Constant Molal Overflow. (a) Stripping Example, (b) Extraction Example.

3.7 Summary

In this chapter, we described how EXSEP represents a given problem by using the component assignment matrix (CAM). From the large list of heuristics presented in Chapter 2, we retained the main ones used in the MSA-modules; they concerned the number of stages and the separation factor. EXSEP uses these heuristics to generate feasible flowsheet solutions. It also has an explanation facility that clarifies the numerical results and predicts the possible existence of other solutions.

Finally, we commented on the validity and consistency of the solutions that EXSEP gives when compared with results from rigorous CAD simulations. We discuss this topic further in section A.1 with several other examples.

CHAPTER 4

User's Perspective of EXSEP

The three MSA-modules, Absorb, Stripping and LLE, have much in common. Most of the menus and windows that the user goes through are the same. In sections 4.2 and 4.3, we shall focus on the particularity of the Stripping and Extraction modules, but first we shall look at the common procedures through the use of the Absorb module.

4.1 System and File Requirements

EXSEP can run on almost any IBM-PC (AT,XT or PS/2) using MS DOS. It was developed for VGA monitor and works better with this type, but it can also work on CGA, EGA, and even Hercules Monochrome monitors. The executable version EXSEP.EXE is in *compiled* Prolog. Its relatively small size (280 Kbytes) makes it usable from floppy disk. It does not require an extended memory. One Mbyte of RAM is entirely sufficient. It solves the example presented in this thesis in less than 1 second CPU with a 286, 8Mhz processor. This makes EXSEP an efficient expert system for IBM-PC. To run, EXSEP needs five type of files:

- EXSEP.EXE (282 Kb)
- MAIN.IDB (65.5 Kb)
- MENU.ARI (2.6 Kb)
- *.DAT (0.5 Kb) [Data Files]
- DEFAULT.* (0.1 Kb) [Default Data Files]

4.2 Common Procedures: An Example of Absorption

4.2.1 Program Initiation

From the DOS prompt C:\> (or A:\>), the only thing to do is to type **EXSEP** and strike [enter]. The first window shows the name of the program and the title of the main menu (Fig. 4.1a).

4.2.2 Separation Choice

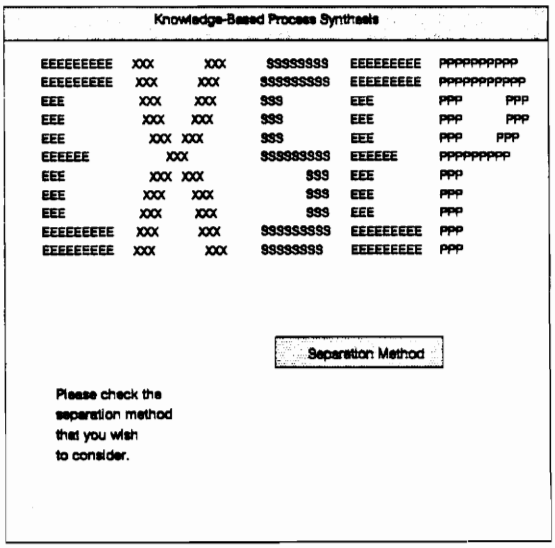
The first step is to select a separation method. We start by pressing the NumLock key to **turn NumLock off**. Then we strike the down-arrow key which reveals the main menu (Fig. 4.1b). We can select one separation method among the four available ones, which are ordinary distillation, absorption, stripping and liquid-liquid extraction. We choose absorption by pressing the down-arrow key once (absorption is highlighted), and also by pressing [enter].

EXSEP then asks if we want to select another separation method (Fig. 4.1c). For example, we could select ordinary distillation and absorption, and EXSEP would try distillation **first**. If distillation is not feasible, EXSEP would enter the absorption module automatically.

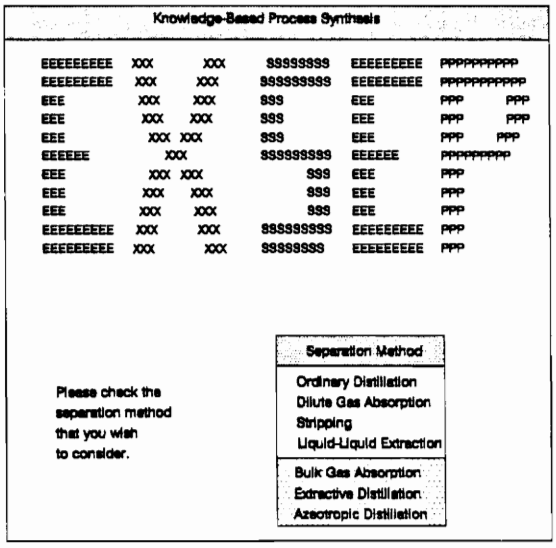
Usually, we answer 'No' (Fig. 4.1d) by typing [enter] twice (The first strike reveals the two alternatives 'Yes' or 'No', and the second validates 'No'). **We are now in the Absorb module.**

4.2.3 Data Loading

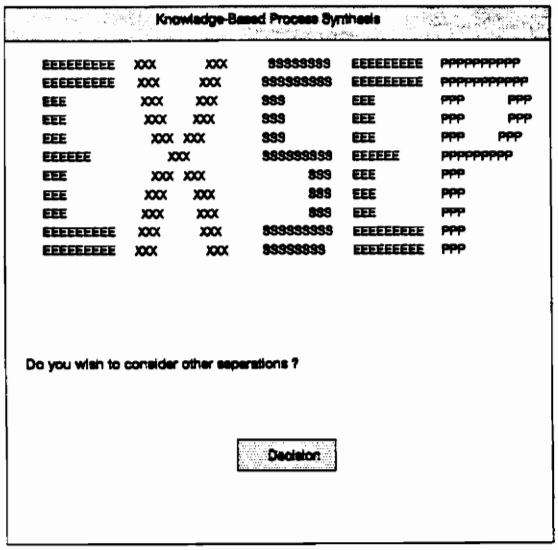
Between the actual problem statement shown in Table 4.1, and the CAM representation explained in section 3.1, we need to enter the data that will enable EXSEP to solve the synthesis problem. When we first work on a new problem, we must enter the data manually and then save a file containing what we entered. We can also write the file with an editor or a word processor,



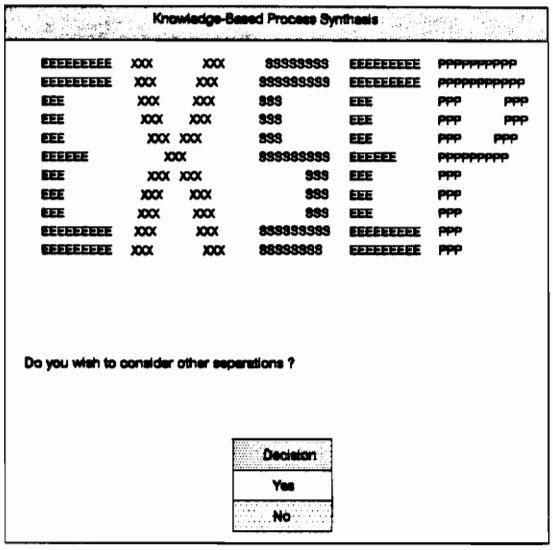
a



b



c



d

Figure 4.1a-c : First Windows Before Entering a Module

or simply modify a file written for the same kind of problem.

Table 4.1: Problem Statement, Example of Absorption.

Components	Vapor Overhead: p1 mol/hr	Liquid Bottom: p2 mol/hr	K-value
H_2	85.59	0.0	50
C_3H_8	5.72	1.43	2.80
iC_4H_{10}	0.751	0.639	1.2
C_4H_{10}	0.994	1.556	0.9
$i\text{-}C_5H_{12}$	0.013	1.327	0.37
$C_5H_{12}+$	0.0	1.98	0.24

4.2.3.1 Manual Loading

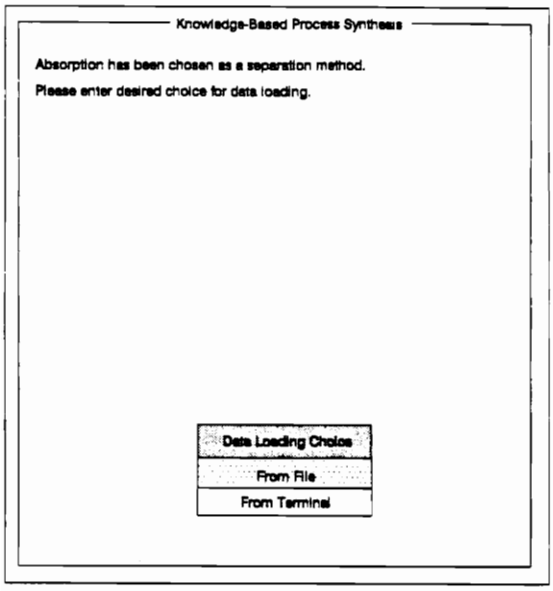
Figure 4.2a shows the next window. We need to select a data-loading method. If we choose "From Terminal" in the menu (Fig. 4.2a), we need to type the data set. The procedure to enter the data is simple. EXSEP asks the questions (Fig. 4.2b-c), and we just provide the answers.

Below, we show the actual interactive question/answer session that we have to follow to enter the data contained in Table 4.1. The answers we give are shown in bold. The presence of the full stops (.) or the apostrophes (') is compulsory.

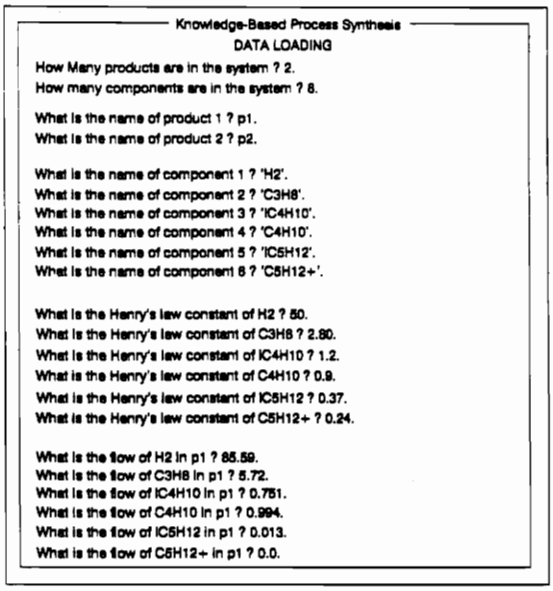
How many products are in the system ? ***2.***
How many components are in the system ? ***6.***

What is the name of product 1 ? ***p1.***
What is the name of product 2 ? ***p2.***

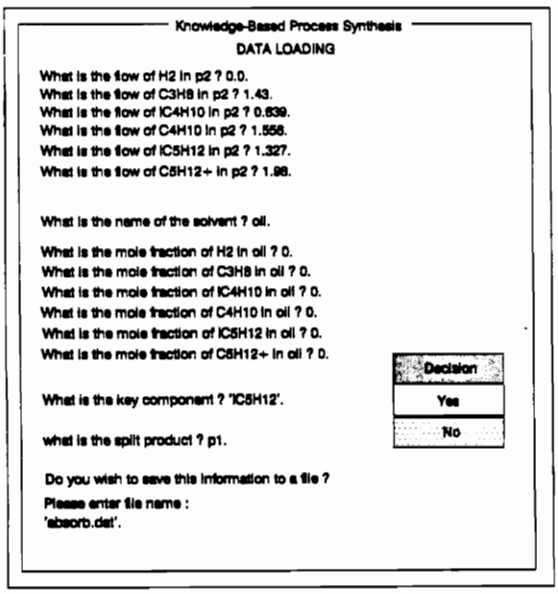
What is the name of component 1 ? ***'H2'.***
What is the name of component 2 ? ***'C3H8'.***
What is the name of component 3 ? ***'IC4H10'.***
What is the name of component 4 ? ***'C4H10'.***



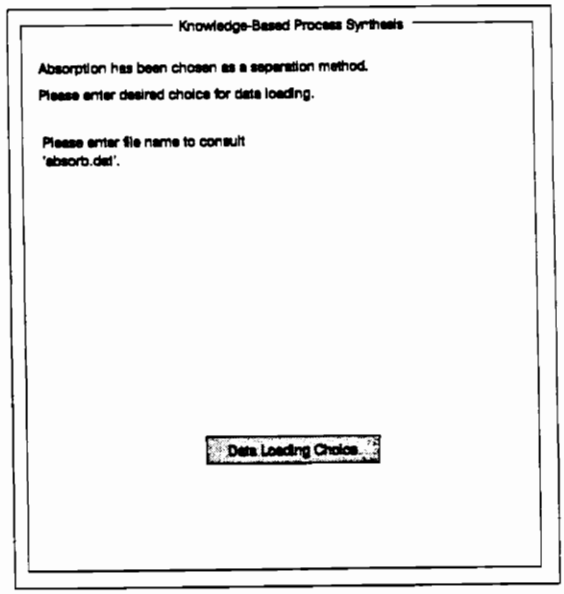
a



b



c



d

Figure 4.2a-d : Data-Loading Options

*What is the name of component 5 ? **'IC5H12'.***
*What is the name of component 6 ? **'C5H12+'.***

*What is the Henry's law constant of component H2 ? **50.***
*What is the Henry's law constant of component C3H8? **2.80.***
*What is the Henry's law constant of component IC4H10 ? **1.2.***
*What is the Henry's law constant of component C4H10 ? **0.9.***
*What is the Henry's law constant of component IC5H12 ? **0.37.***
*What is the Henry's law constant of component C5H12+ ? **0.24.***

*What is the flow of component H2 in p1 ? **85.59.***
*What is the flow of component C3H8 in p1 ? **5.72.***
*What is the flow of component IC4H10 in p1 ? **0.751.***
*What is the flow of component NC4H10 in p1 ? **0.994.***
*What is the flow of component IC5H12 in p1 ? **0.013.***
*What is the flow of component C5H12+ in p1 ? **0.0.***

*What is the flow of component H2 in p2 ? **0.0.***
*What is the flow of component C3H8 in p2 ? **1.43.***
*What is the flow of component IC4H10 in p2 ? **0.639.***
*What is the flow of component C4H10 in p2 ? **1.556.***
*What is the flow of component IC5H12 in p2 ? **1.327.***
*What is the flow of component C5H12+ in p2 ? **1.98.***

*What is the name of the solvent ? **oil.***

*What is the mole fraction of H2 in oil ? **0.***
*What is the mole fraction of C3H8 in oil ? **0.***
*What is the mole fraction of IC4H10 in oil ? **0.***
*What is the mole fraction of C4H10 in oil ? **0.***
*What is the mole fraction of IC5H12 in oil ? **0.***
*What is the mole fraction of C5H12+ in oil ? **0.***

*What is the key component ? **'IC5H12'.***

*What is the split product ? **p1.***

After we enter the data, EXSEP asks us if we want to save the data (Fig. 4.2c). We answer 'yes'. The file saved is the equivalent to that shown in section 4.1.3.2.

4.2.3.2 File Loading

The other option we can choose is the "From File" option.

- If no file exists, the first step is to type it with an editor or a word processor (saved in ASCII or DOS text mode). Written in Prolog, the file is:

```
henry('H2',50.0).
henry('C3H8',2.8).
henry('IC4H10',1.2).
henry('C4H10',0.9).
henry('IC5H12',0.37).
```

```
henry('C5H12+',0.24).
flow(p1,'H2',85.59).
flow(p1,'C3H8',5.72).
flow(p1,'IC4H10',0.751).
flow(p1,'C4H10',0.994).
flow(p1,'IC5H12',0.013).
flow(p1,'C5H12+',0).
flow(p2,'H2',0).
flow(p2,'C3H8',1.43).
flow(p2,'IC4H10',0.639).
flow(p2,'C4H10',1.556).
flow(p2,'IC5H12',1.327).
flow(p2,'C5H12+',1.98).
mole_fraction(solvent,'H2',0).
mole_fraction(solvent,'C3H8',0).
mole_fraction(solvent,'IC4H10',0).
mole_fraction(solvent,'C4H10',0).
mole_fraction(solvent,'IC5H12',0).
mole_fraction(solvent,'C5H12+',0).
initial_set([p1,p2]).
initial_components(['H2','C3H8','IC4H10',
        'C4H10','IC5H12','C5H12+']).
solvent(oil).
key_component(dga,'IC5H12'). % dga = dilute gas absorption
split_product(p1).
```

- If the file exists, we just need to type its name according to the following syntax:

'filename.ext'. [enter]

or

filename [enter] if its extension is .ARI

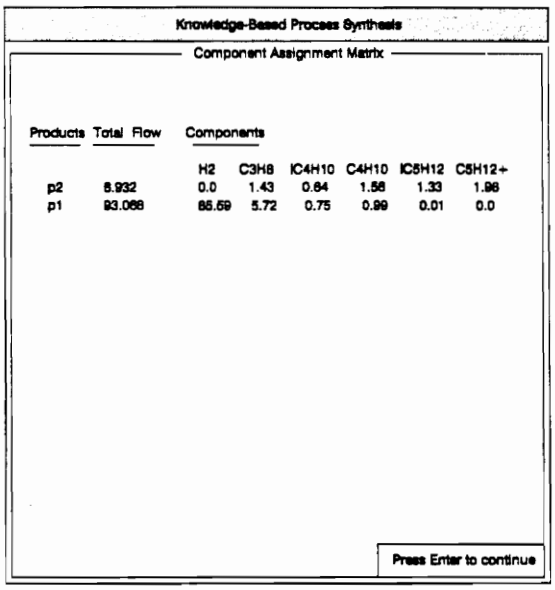
We type (Fig. 4.2d) : **'absorb.dat'.**

4.2.4 Component Assignment Matrix (CAM)

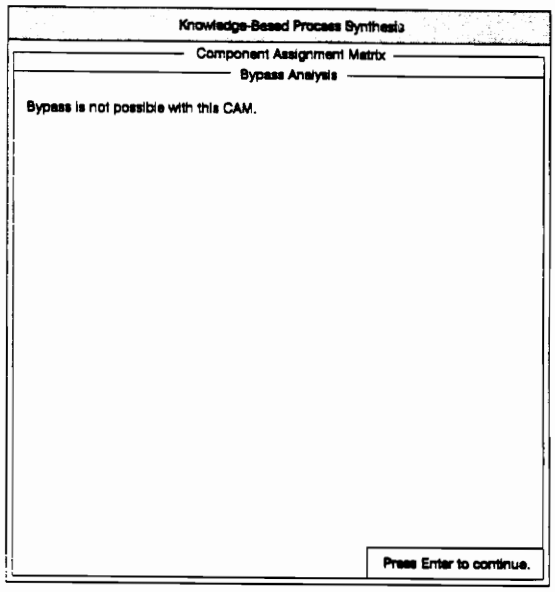
After EXSEP has read the data file, it proposes to display the CAM. We choose 'yes' to display the CAM (Fig. 4.3a).

4.2.5 Bypass

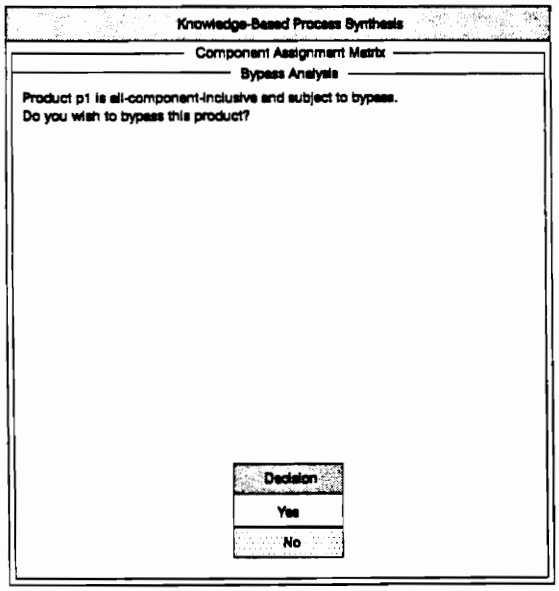
The Bypass module (Quantrille and Liu, 1991, p.304) is used to identify if bypassing a portion of the feed is possible or not. Figure 4.3b shows that for the absorption example, feed



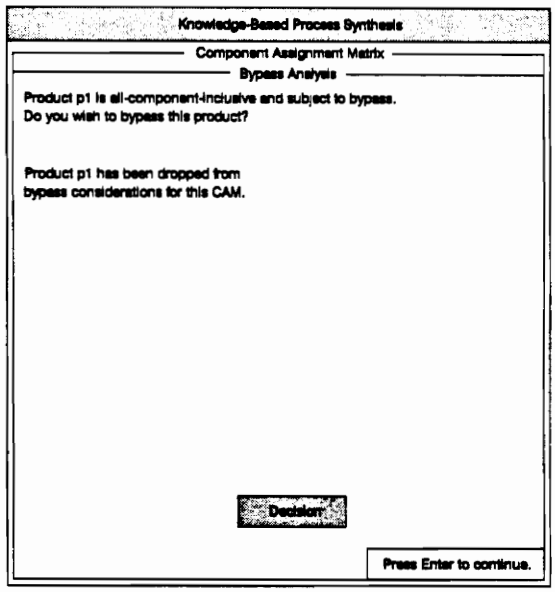
a



b



c



d

Figure 4.3a-d : Component Assignment Matrix (CAM) and Bypass Procedure.

bypass is not possible. When EXSEP finds a possible bypass (Fig. 4.3c), it displays the message:

Product p1 is all-component-inclusive and subject to bypass. Do you wish to bypass the product ?

- If we answer 'No', the message is:

Product p1 has been dropped from bypass considerations for this CAM. (Fig. 4.3d).

- If we answer 'Yes', EXSEP asks us to choose the percentage of the feed to bypass (Fig. 4.4a). We can choose one of the three options.

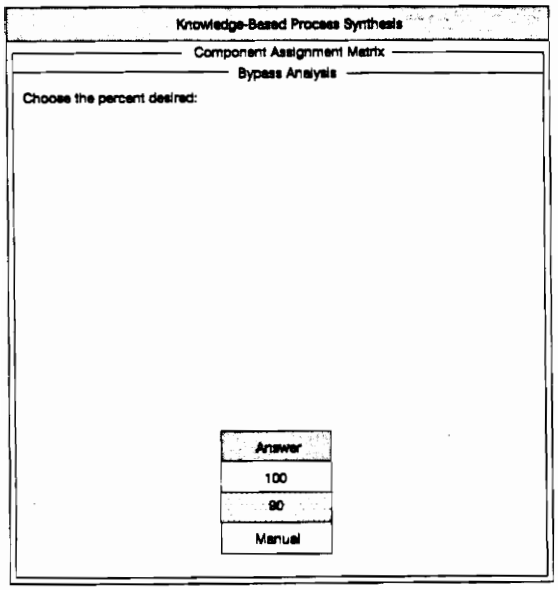
- if we choose "Manually", the question/answer is:

Enter the percent bypass you desire ***50.***

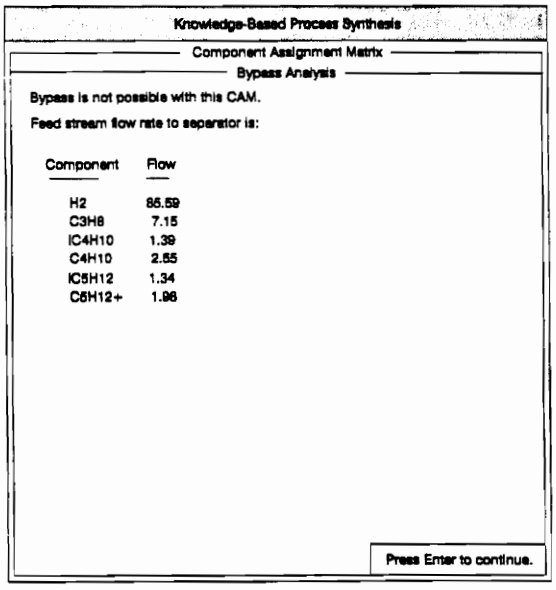
After the bypass is calculated, EXSEP displays the amount actually fed to the column (Fig. 4.4b) and proposes to redisplay the CAM recalculated after bypassing and taking into account the presence of the solvent (Fig. 4.4c-d). The same procedure is performed for the bypass analysis of product p2. EXSEP then continues with the search for solutions.

4.2.6 Flowsheet Solutions

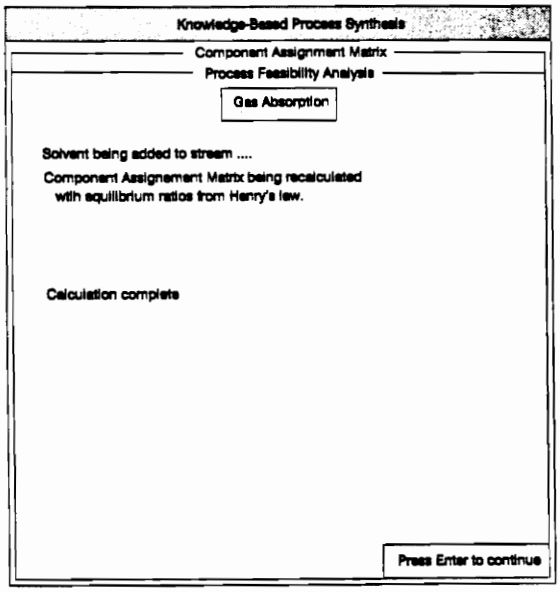
There are two kinds of windows to display solutions. The first displays a table that compares the specified and actual component-recovery ratios (Fig. 4.5a), and the second (Fig. 4.5b) displays the design variables (separation factor and number of stages). In the second window, EXSEP also provides explanations for the solution.



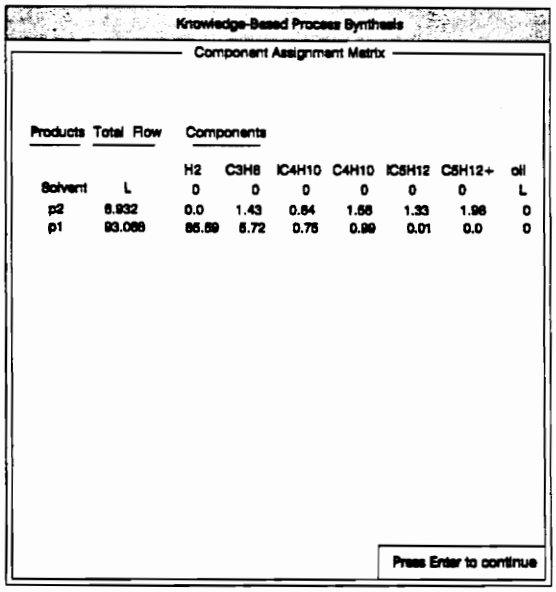
a



b

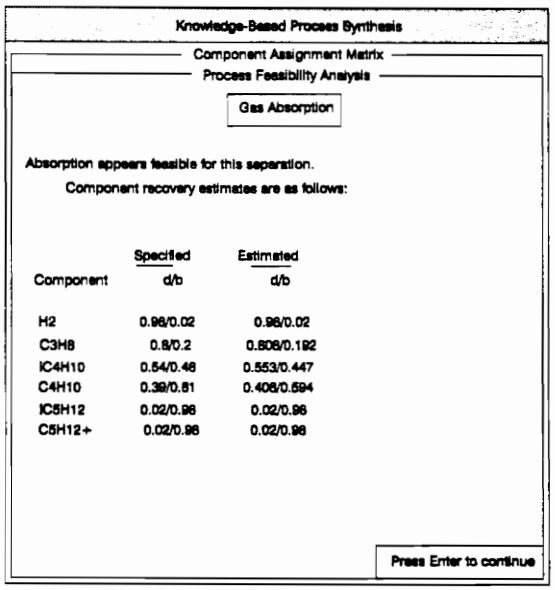


c

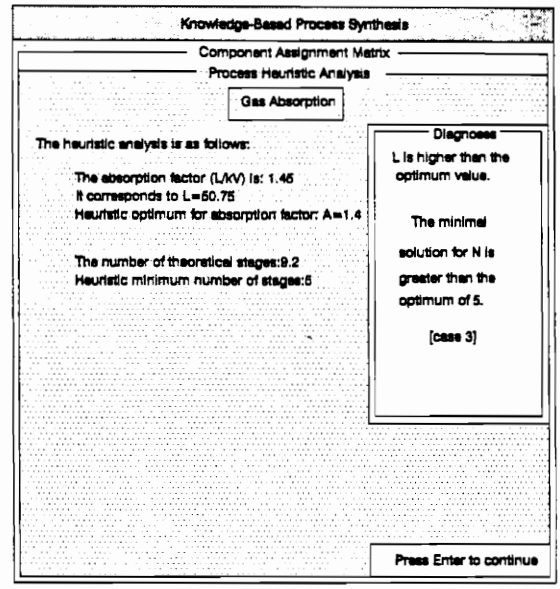


d

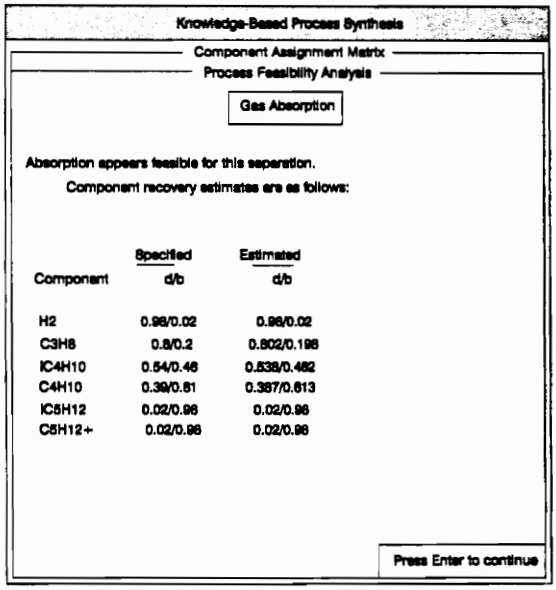
Figure 4.4a-d : Bypass Procedure and Recalculated Component Assignment Matrix (CAM).



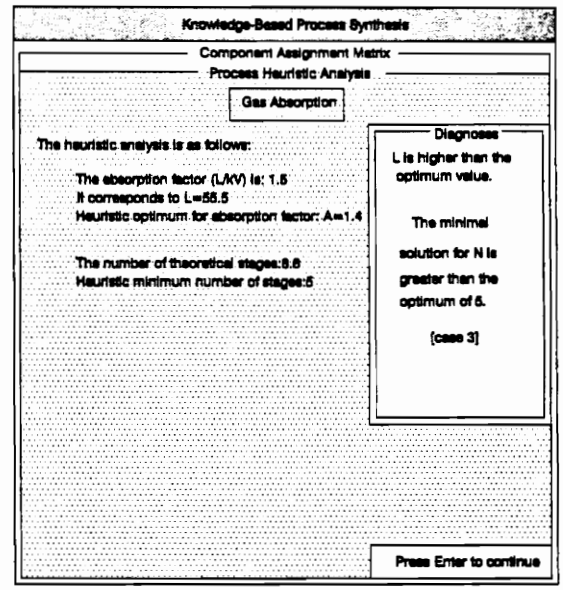
a



b



c



d

Figure 4.5a-d : Solution Windows

4.2.6.1 Component-Recovery Ratios

Fig. 4.5a shows the window that presents the component-recovery ratios. When we see this table, the solution is feasible and the deviation between the specified and actual ratios of component recovery in the overhead product to that in the bottom product (d/d's) is less than 10%. If we then press [enter], the window showing the design variables appears.

4.2.6.2 Design Variables

This window (Fig. 4.5b) shows the design variables and the corresponding heuristics. We shall explain the small window in the right of the screen in section 4.1.6.3.

The first line in Fig. 4.5b gives the actual value of the separation factor (here, the absorption factor), and the second line indicates the optimal value given by heurisitics. The separation factor A is proportional to the liquid-solvent flow rate (L=A*kV) and kV is constant. The next two lines give the actual number of theoretical stages and the heuristic.

4.2.6.3 Position Diagnosis

The blinking window of the right (Fig. 4.5b) explains the design variables found. As we discussed in section 3.3.2, the quality of a solution depends on its "position" compared with the optimal value. For this absorption example, the first solution has the following explanation:

N=8.6 and A=1.5

The diagnosis is:

"L is higher than the optimum value. The minimal solution for N is greater than the optimum of 5." [case 3]

This diagnosis is more understandable if we look at Figure 4.6a. The solution is A > 1.4,

so L= A*kV > 1.4*kV, which explains the first sentence: *L is higher than the optimum value.* N > Nopt=5 explains the second sentence: *The minimal solution for N is greater than the optimum of 5.*

By pressing [enter], we reveal additional solutions (Fig. 4.5c-d and Fig. 4.7a-b) with appropriate explanations. When EXSEP terminates the search (the separation factor has been incremented to the upper limit), it displays the range of the search together with corresponding explanations (Fig. 4.6a).

4.2.7 Search-Range Diagnosis

Fig. 4.7a shows the explanation in the right blinking window. The studied range for this example is: **1.35-1.6**. The diagnosis the EXSEP gives here is: *" Solution may exist for A > 1,6, with a number of stages still higher than 5. You should increase Asl".*

Looking at Figure 4.6b, we notice that Ast is effectively greater than Asl=1.6 (upper limit of the search). As we discussed in section 3.3.3, there may exist a feasible solution for A greater than Asl. EXSEP advises to increase Asl such that Ast is smaller than Asl. The problem is: EXSEP does not know Ast. So we should increase Asl, but we do not know how much.

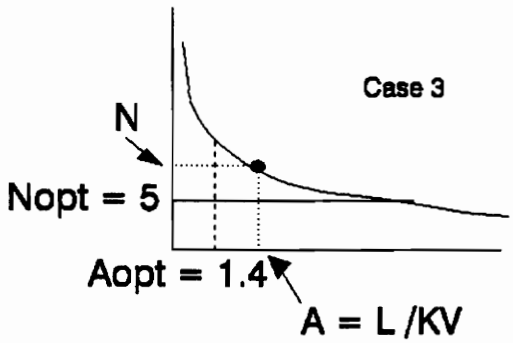
The characteristic factors A_{min}, A_{opt}, A_{st}, and A_{sl} (see section 3.3.1), including Asl, are contained in a **separate file called DEFAULT.FAC**. This file also contains the optimal value of the number of stages, the incremental "step-size" of the increment for the separation factor, and the acceptable deviation between the actual and specified component-recovery ratios:

a)

- **Solution: N=8.6 A=1.5**

"L is higher than the optimum value. The minimal solution for N is greater than the optimum of 5."

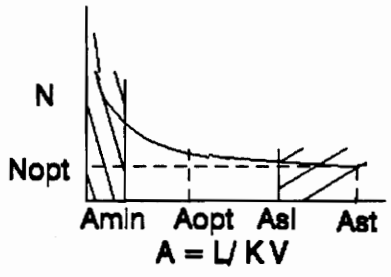
[case 3]



b)

- **Range: 1.35 - 1.6**

"Solution may exist for A>1.6, with a number of stages still higher than 5. You should increase Asl."



Amin=1.35

Aopt=1.4

Ast>1.6

Asl=1.6

Figure 4.6a-b : Explanation Facility a) Solution b) Range

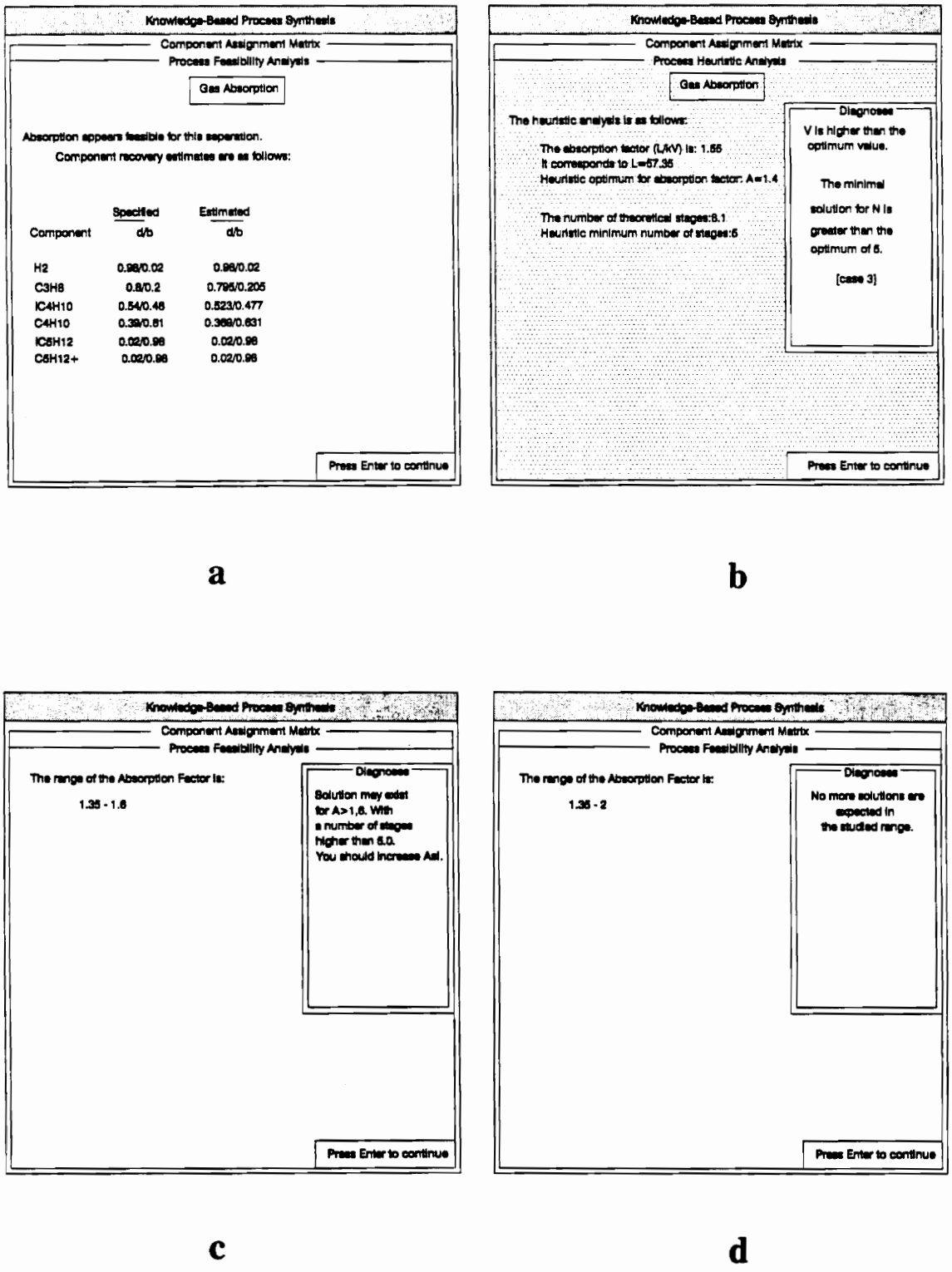


Figure 4.7a-d : Third Solution and Range Explanations

DEFAULT.FAC

optimal_A(1.4). %optimal abs. factor A_{opt}
minimum_A(1.35). %minimum abs. factor A_{min}
limit_A(1.6). %maximum abs. factor A_{sl}
optimal_N(5). %opt. numb. stages
step(0.05). %increment size
error(0.1). %error on d/b's

To change Asl (limit_A(1.6), here), we need to edit the default.fac file with a word processor or an editor to change 1.6 to 2, for example. Then EXSEP will provide the following diagnosis (Fig. 4.7d),

"No more solutions expected in the studied range".

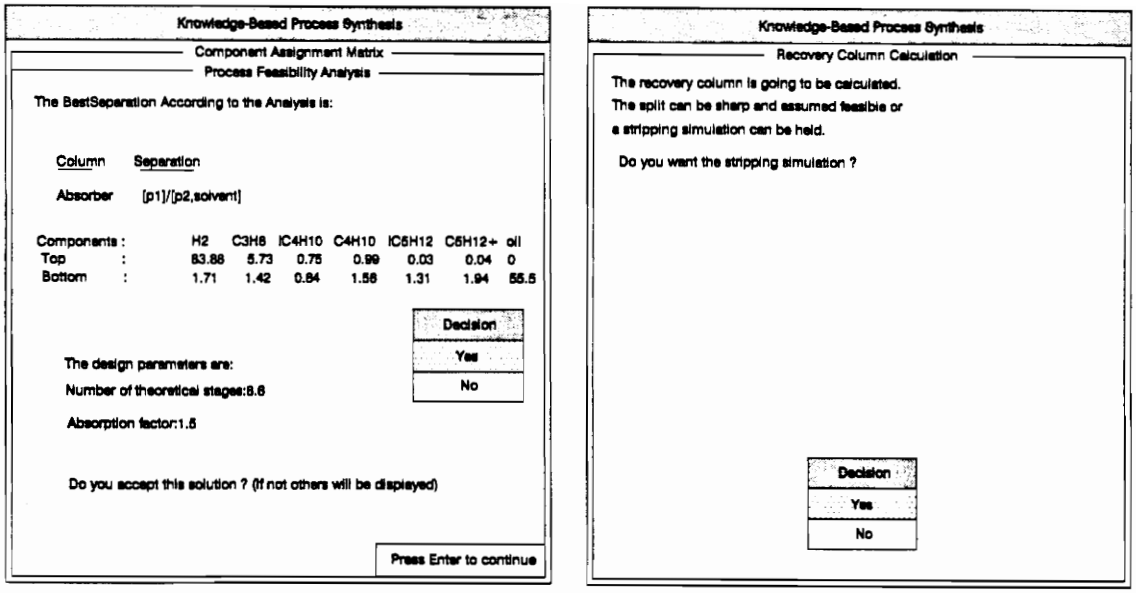
Because Ast will be smaller than Asl.

4.2.8 Choice of the Best Solution

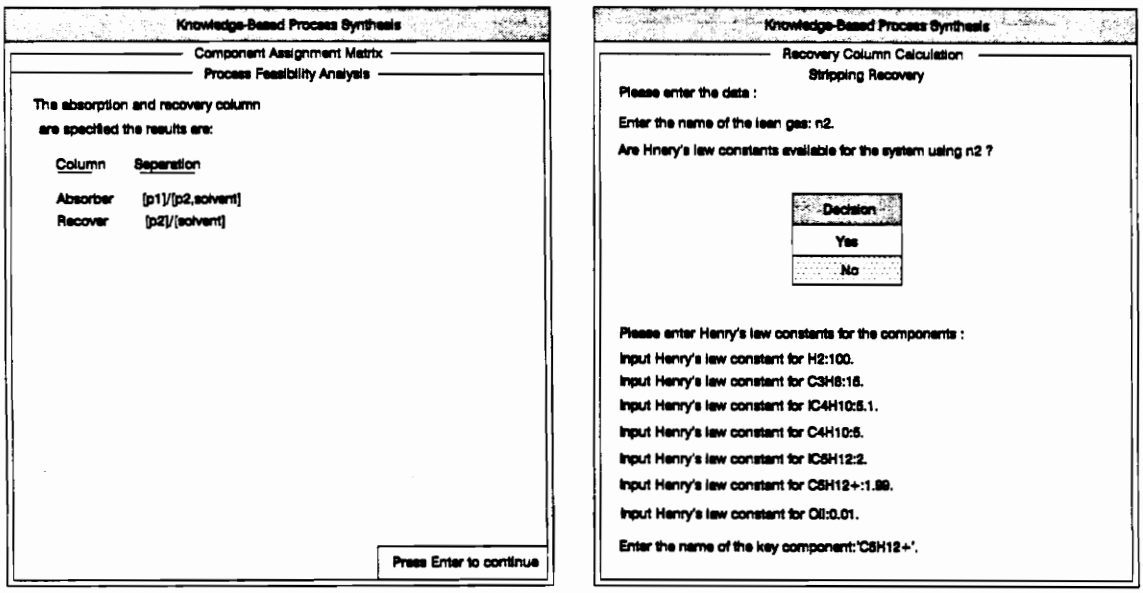
The next window EXSEP displays concerns the choice of the best solution. As we explained in section 3.4, EXSEP ranks the solutions according to the coefficient-of-separation (CS) factor. Figure 4.8a shows the best solution that EXSEP gives for the absorption example.

The first two lines show the name of the separator and the split performed. We have the list of the components and the solvent, and their respective actual flow rates to the top and the bottom. These flow rates can differ from the specified values because of the 10% deviation accepted for the component-recovery ratios. Fig. 4.8a also shows the design parameters for this solution.

EXSEP asks us if we accept this separation. By pressing the down-arrow key, we see the decision menu. If we select 'No', EXSEP displays the next feasible solution and asks again if we accept it. We choose 'Yes' and enter the recovery-column calculation.



a b



c d

Figure 4.8a-d : Best Separation and Recovery Procedure

4.2.9 Recovery Column

At this point (Fig. 4.8b), we have two options:

- Assume the solvent recovery to be feasible, or
- Use the stripping module.

Should we select the first option (answer 'No'), EXSEP specifies that *the recovery of the solvent is supposed to be feasible and sharp*, and it displays where the cut is done (Fig. 4.8c). If we select the second option (answer 'yes'), EXSEP goes to the stripping module. When we use the distillation module for the stripping operation, the best way is to finish the absorption (first option) and then run a distillation simulation with EXSEP according to the procedure described by Quantrille and Liu (1991, p.320).

When we enter the stripping module, EXSEP has already estimated the expected flow rates (**it specifies a sharp separation**) to the top and bottom of the stripper, but it needs other information. The procedure (Fig. 4.8d) is the same as described in section 4.2.1.

A third alternative to run a stripping recovery is to choose the first option, finish the absorption simulation, and run a stripping simulation with the bottom of the absorber being the inlet of the stripper. This alternative enables us to choose a nonsharp solvent recovery. Indeed, the direct method (second option) specifies a sharp separation; if it is not feasible, we need to guess the flow rates in the products and test the feasibility of the split with EXSEP.

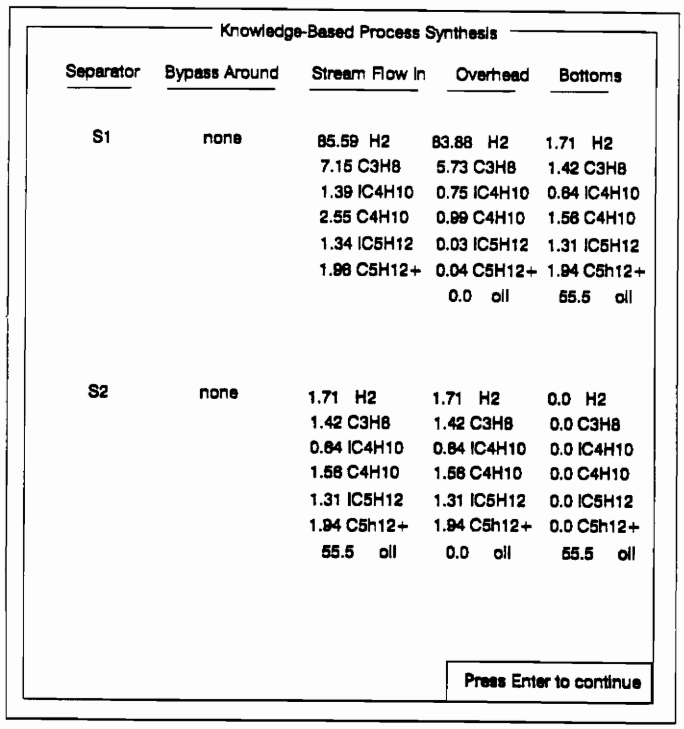
4.2.10 Final Flowsheet

Next, we enter the flowsheet summary window (Fig. 4.9a). The first column shows the names of the separators. The second column gives the amount of product bypassed when the

BYPASS module is used. The column "Stream Flow In" is the inlet flow of the recovery column, and the last two columns are the flow rates of the components and the solvent in the overhead and bottom products. In this example, the last column for separator S1 becomes the inlet stream of the solvent-recovery unit. As we chose the first option for a sharp split for the recovery column, we observe that the stripper split is sharp, because the flow rate of each component to the bottom of the recovery column is zero.

4.2.11 Program Ending

By pressing [enter], we end the simulation with the last window (Fig. 4.9b) which offers to "terminate" the program or "synthesize another" process. If we select "terminate", we go back to DOS. Let us select "synthesize another" process. We come back to the first window, where we shall try a stripping example (see section 4.2).



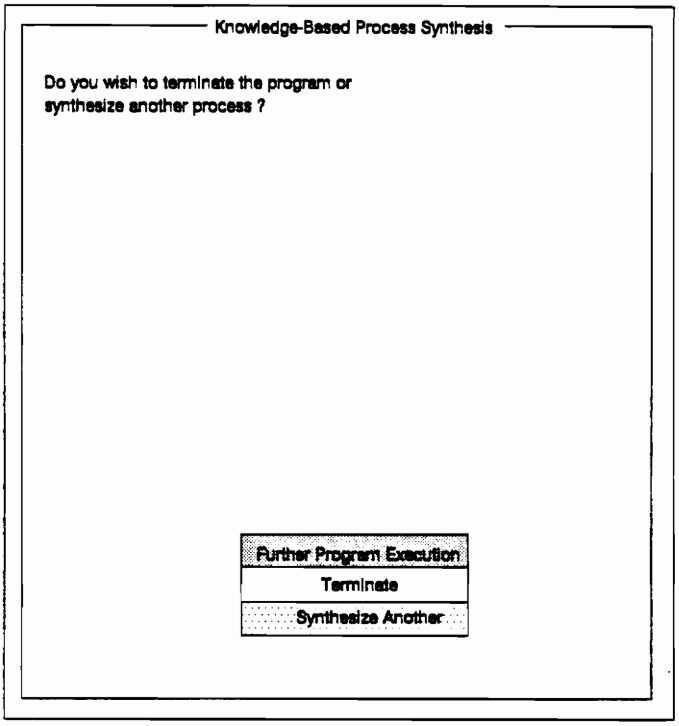


Figure 4.9a-b : (a) Flowsheet Summary, and (b) Program Ending.

4.3 An Example of Stripping

This example was presented in Chapter 3 (section 3.5.1.2). We want to remove the butadiene sulfone contained in a mixture of sulfur dioxide (SO2), butadienes 2 (B2) and 3 (B3), and butadiene sulfone (BS) by stripping SO2, B2 and B3 with nitrogen.

4.3.1 Data Loading

Figure 4.10a-d show the procedure to load the data. The question/answer session of the option "From Terminal" appears as follows:

How many products are in the system ? ***2.***
How many components are in the system ? ***4.***

What is the name of product 1 ? ***p1.***
What is the name of product 2 ? ***p2.***
What is the name of component 1 ? ***'B2'.***
What is the name of component 2 ? ***'B3'.***
What is the name of component 3 ? ***'SO2'.***
What is the name of component 4 ? ***'BS'.***

What is the Henry's law constant of component B2 ? ***2.33.***
What is the Henry's law constant of component B3? ***3.58.***
What is the Henry's law constant of component SO2 ? ***5.39.***
What is the Henry's law constant of component BS ? ***0.019.***

What is the flow of component B2 in p1 ? ***0.23.***
What is the flow of component B3 in p1 ? ***0.23.***
What is the flow of component SO2 in p1 ? ***0.05.***
What is the flow of component BS in p1 ? ***99.4.***
What is the flow of component B2 in p2 ? ***1.77.***
What is the flow of component B3 in p2 ? ***7.77.***
What is the flow of component SO2 in p2 ? ***9.95.***
What is the flow of component BS in p2 ? ***0.6.***

What is the name of the lean gas ? ***nitrogen.***

What is the mole fraction of B2 in nitrogen ? ***0.***
What is the mole fraction of B3 in nitrogen ? ***0.***
What is the mole fraction of SO2 in nitrogen ? ***0.***
What is the mole fraction of BS in nitrogen ? ***0.***

What is the key component ? ***'B2'.***

What is the split product ? ***p2.***

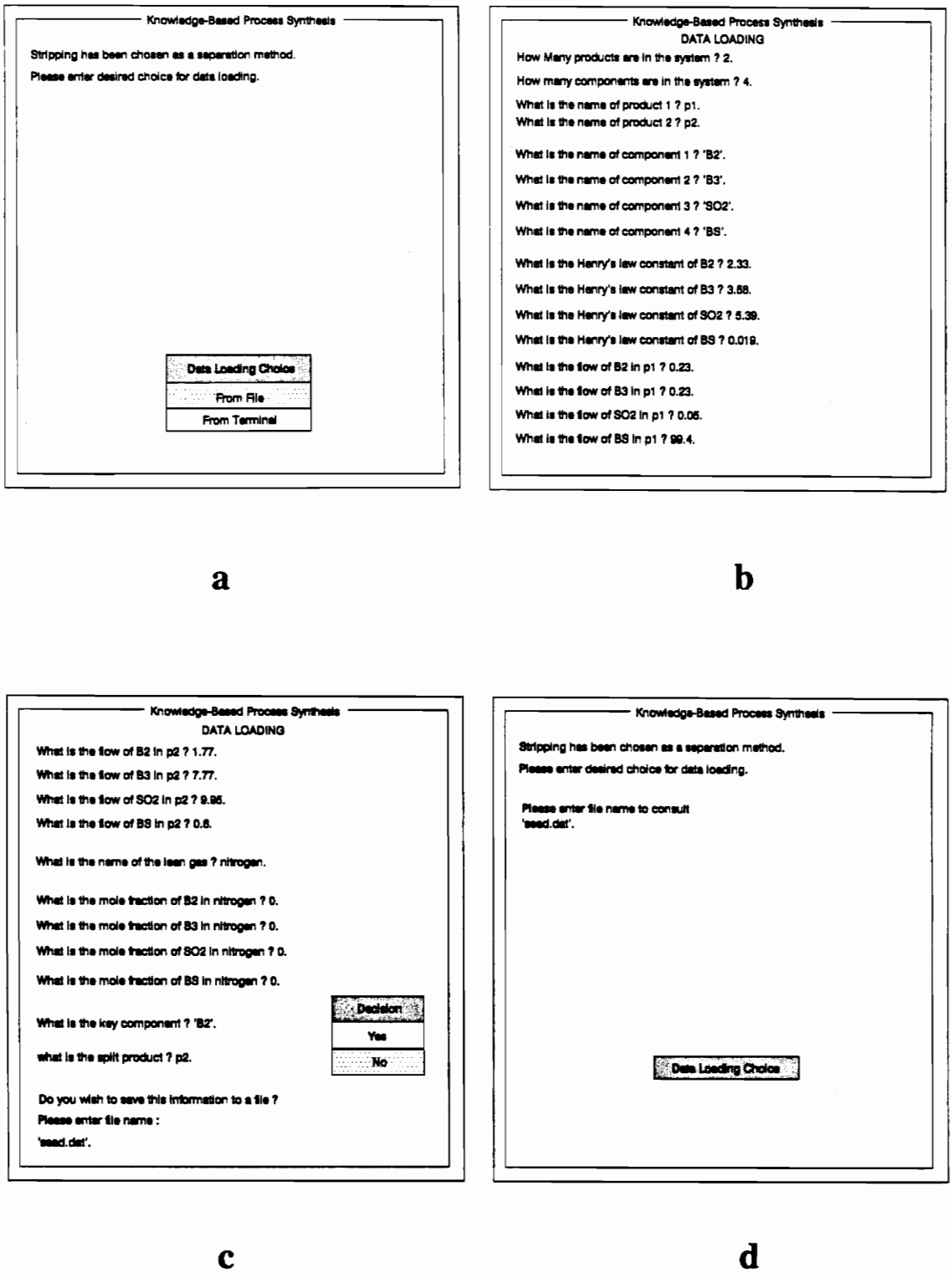


Figure 4.10a-d : Data Loading For the Stripping Example.

It leads to the following file called **sead.dat**

```
henry('SO2',5.39).
henry('B3',3.58).
henry('B2',2.33).
henry('BS',0.019).
flow(p1,'SO2',0.05).
flow(p1,'B3',0.23).
flow(p1,'B2',0.23).
flow(p1,'BS',99.4).
flow(p2,'SO2',9.95).
flow(p2,'B3',7.77).
flow(p2,'B2',1.77).
flow(p2,'BS',0.6).
mole_fraction(lean_gas,'SO2',0).
mole_fraction(lean_gas,'B3',0).
mole_fraction(lean_gas,'B2',0).
mole_fraction(lean_gas,'BS',0).
initial_set([p1,p2]).
initial_components(['SO2','B3','B2','BS']).
lean_gas(nitrogen).
key_component(s,'B2'). % s = stripping
split_product(p2).
```

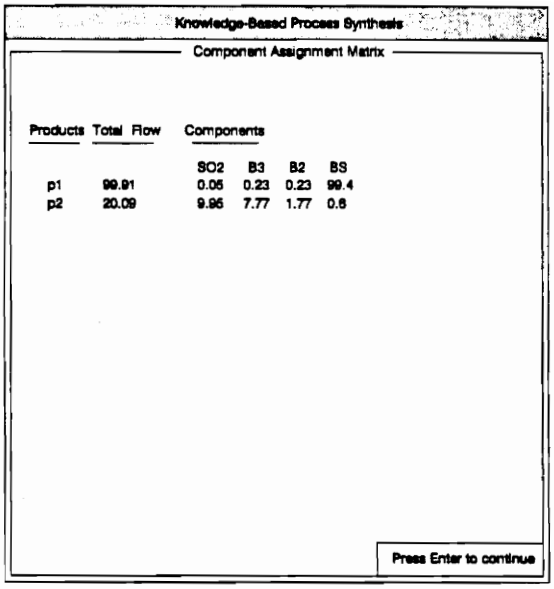
4.3.2 Component Assignment Matrix (CAM)

Fig. 4.11a shows the CAM for the stripping example.

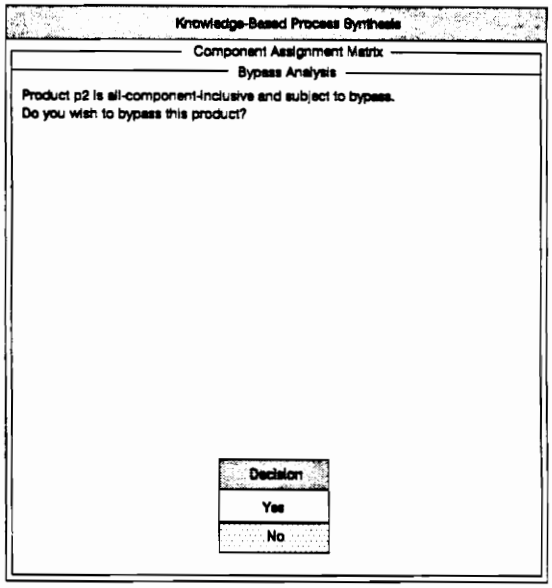
4.3.3 Bypass

In this example, the bypass of the products is possible (Fig. 4.11b-d). To the question, "*Do you wish to bypass this product ?*", we answer 'Yes' for both products. We also choose 100% of bypass.

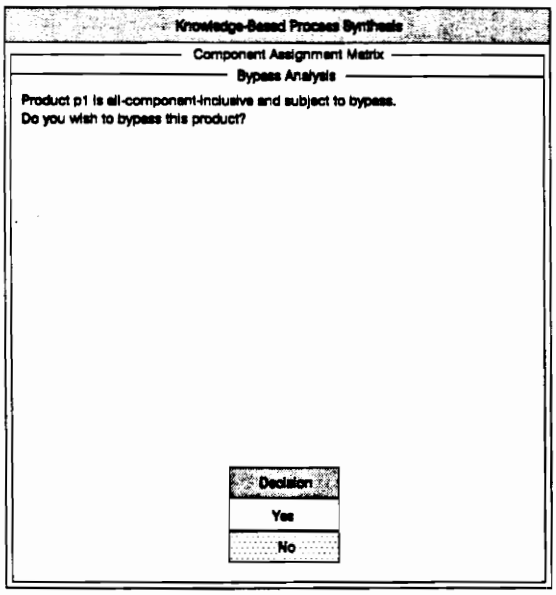
EXSEP displays the amounts actually fed to the column and offers to display the new CAM (Fig. 4.12a-b). We choose 'Yes' to see the recalculated CAM. In Fig. 4.12c-d, we compare the CAMs with and without bypass. In the next section, we shall consider both solutions, with and without bypass.



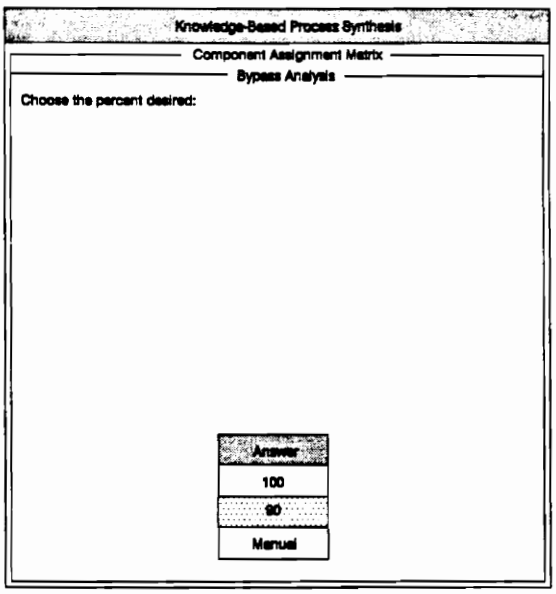
a



b

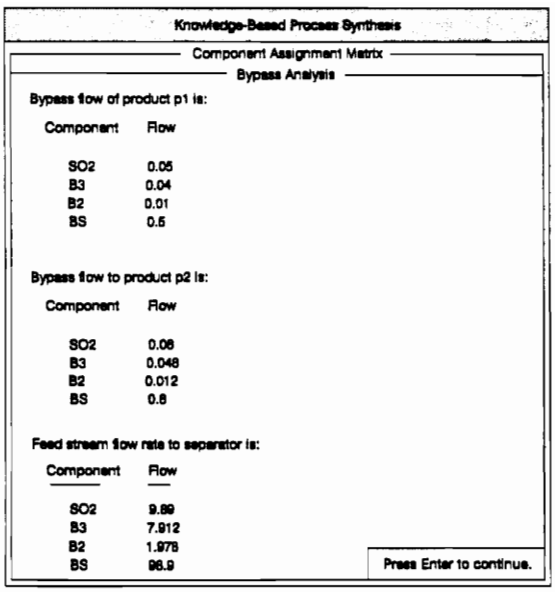


c

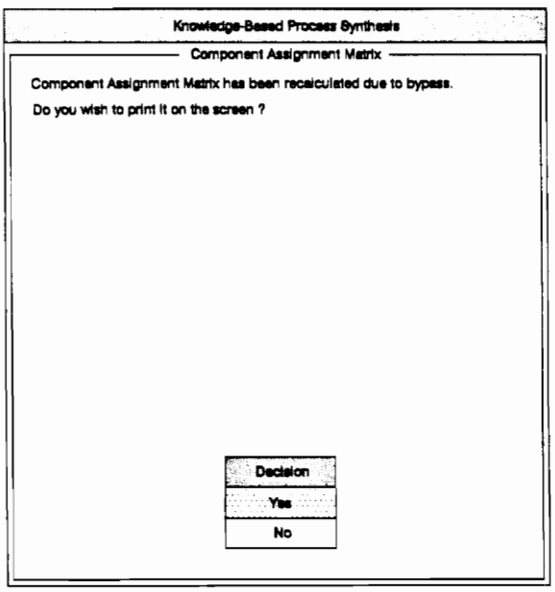


d

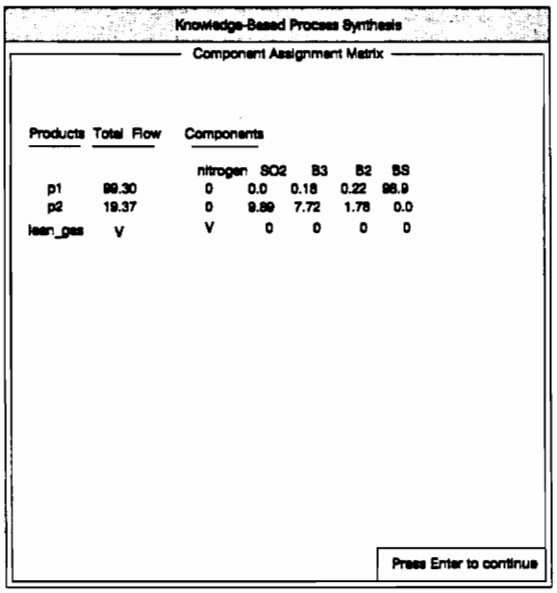
Figure 4.11a-d : (a) CAM, (b-d) : Bypass Procedure



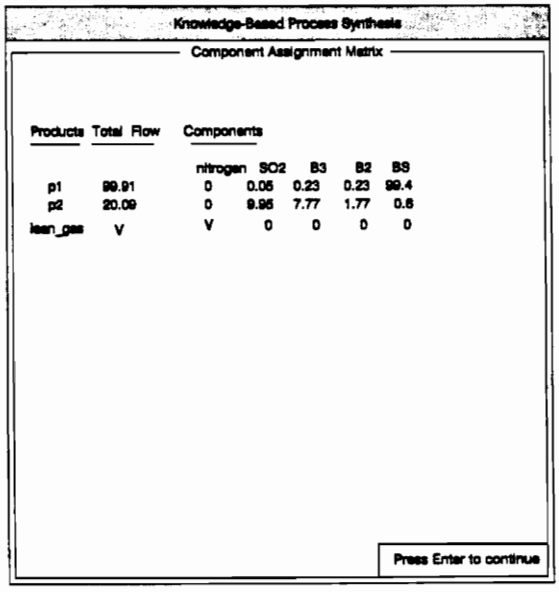
a



b



c



d

Figure 4.12a-d : (a) Feed of the Column, (b) CAM display question, (c) CAM With Bypass, (d) CAM Without Bypass.

4.3.4 Solution

Figure 4.13a-d present the two solutions that EXSEP finds for this problem with bypass, and Figure 4.14a-d shows the solutions without bypass. The incremental steps to go from Smin=0.6 to Slg=1.3 is 0.05. Thus, EXSEP tests 14 different stripping factors and finds only two solutions. If we change the incremental step from 0.05 to 0.025, EXSEP would test twice as many factors and probably would find more than 2 feasible solutions. The explanations are the same as those discussed in section 4.1.6.3.

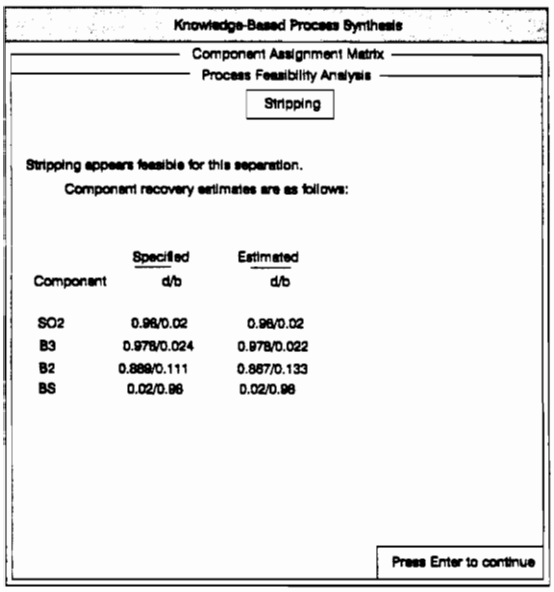
4.3.5 Search-Range Diagnosis

The range of the stripping factor is 0.6-1.3 . EXSEP diagnoses that "*No more solutions are expected in the studied range*". This sentence means that for a stripping factor smaller than 0.6, the number of stages is too high, or impossible to evaluate. Therefore, the separation is not economical. For S greater than 1.3, the number of stages is too small (smaller than the optimum), or the lean gas flow rate is too high. EXSEP infers by this sentence (in italic) that the optimal stripping factor S_{opt} and the stripping factor S_{st} corresponding to an optimal number of stages are included in the interval [0.6;1.3]. This represents the ideal case (see section 5.4.1.2).

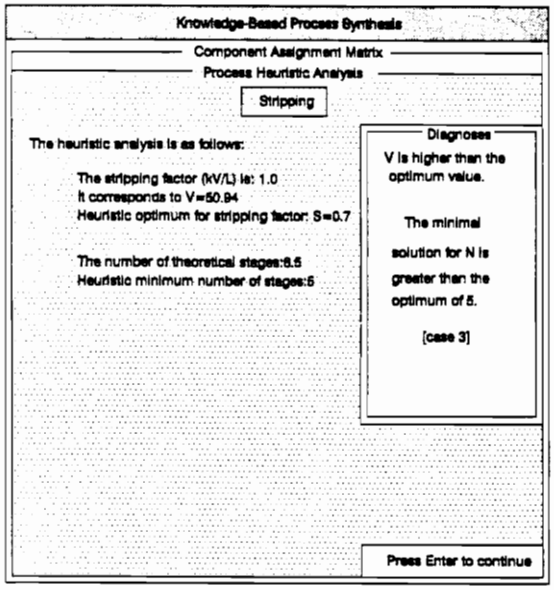
However, there may exist feasible solutions with stripping factors between 0.6 and 1.3, in the way described in the previous paragraph (by changing the incremental step). Figure 4.15a shows the corresponding window (same one without bypass).

4.3.6 Choice of the Best Solution

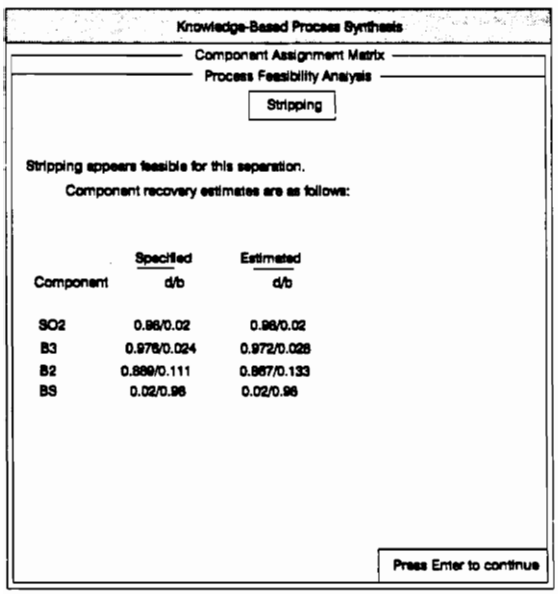
In Figure 4.15b-c, we present the best solutions proposed by EXSEP with and without bypass, respectively. The bypass of 100 % of both products generates only a 1.2 % decrease of the total feed. It decreases the required lean gas flow rate by 6%, but increases the number of



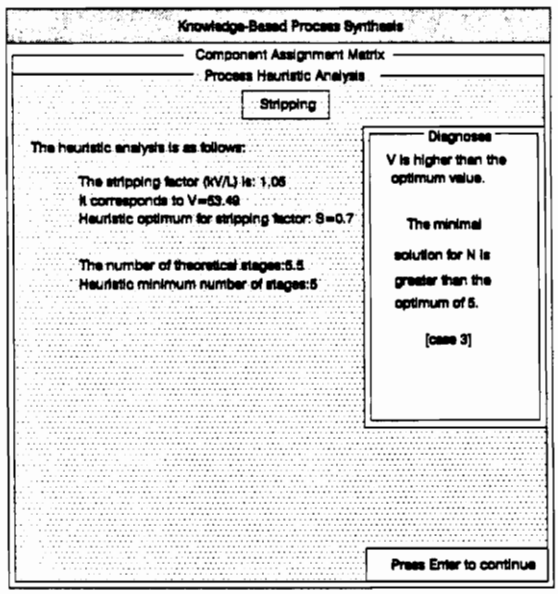
a



b

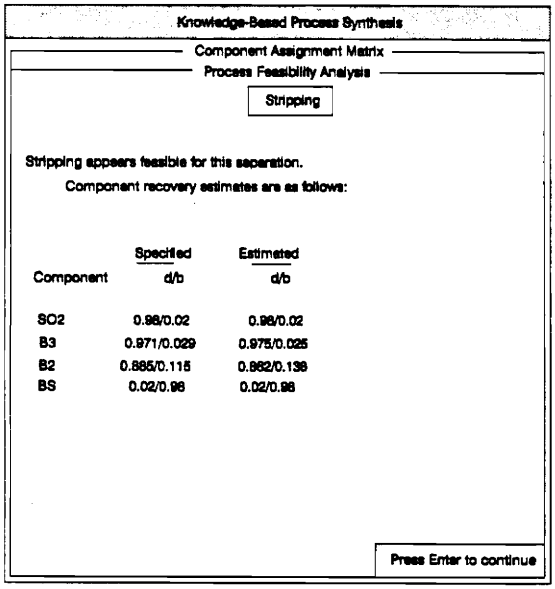


c

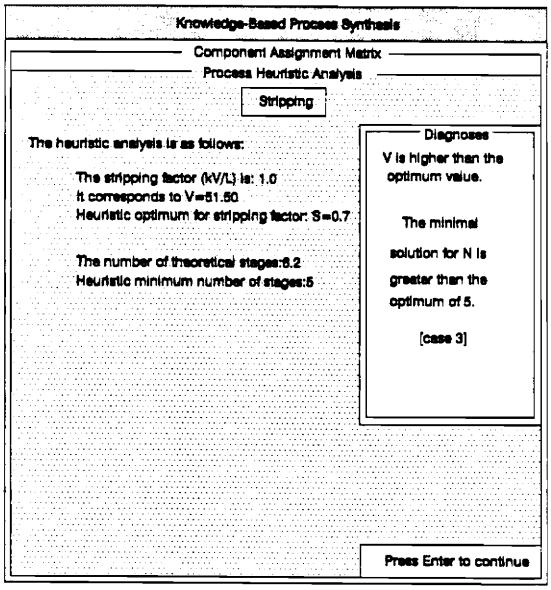


d

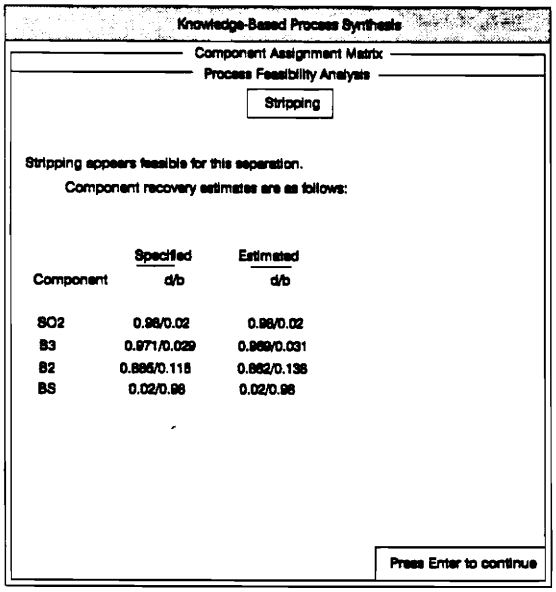
Figure 4.13a-d : Solutions with Bypass.



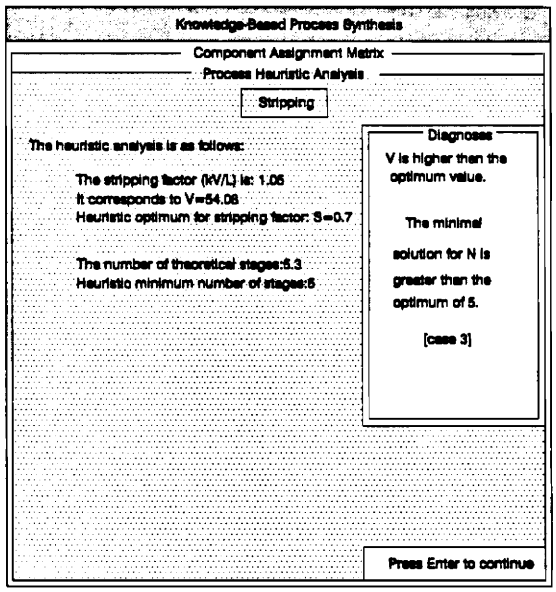
a



b

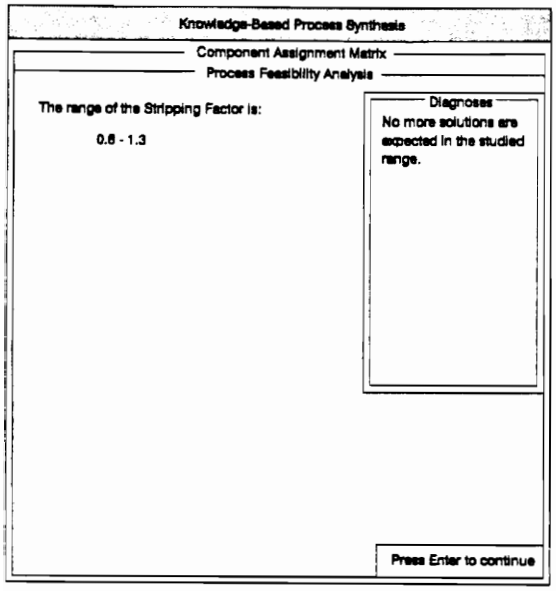


c

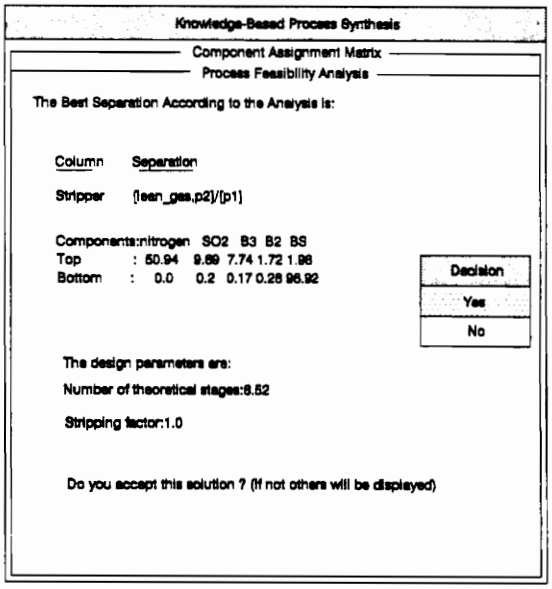


d

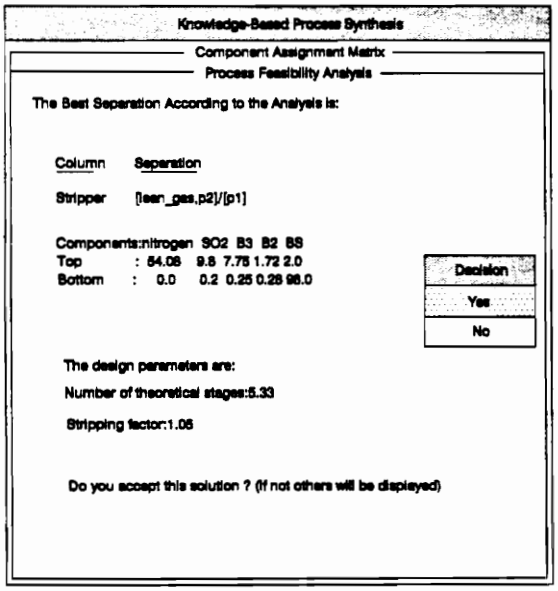
Figure 4.14a-d : Solutions without Bypass.



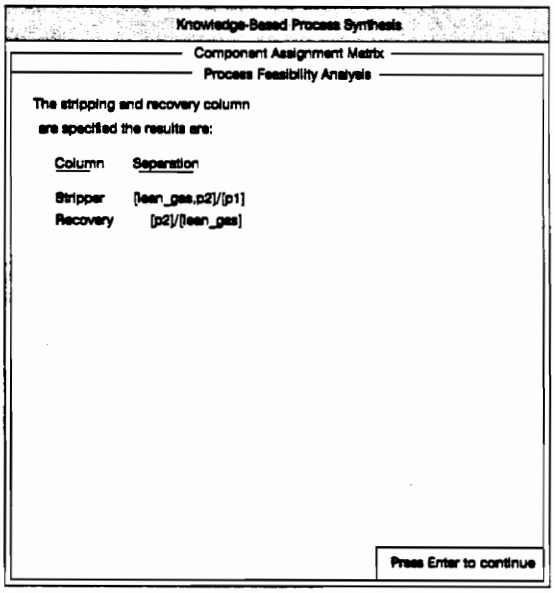
a



b



c



d

Figure 4.15a-d : (a) Range Explanation, (b-c) Best Separations with and without Bypass, (d) Recovery Procedure.

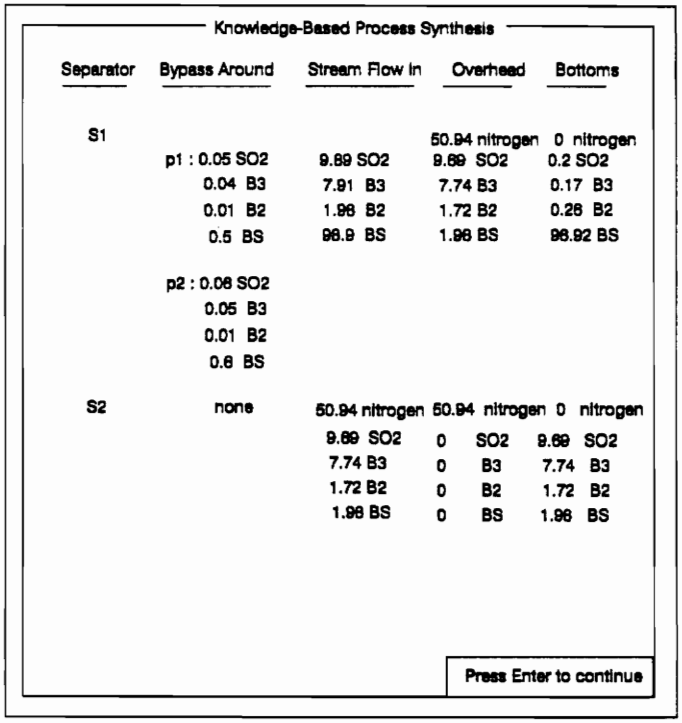
stages by 18 %. We note that the **bypass is not very efficient** for this example. For MSA-based separations, it is not possible to bypass a large amount of product, because the solutions are very dilute and the split must usually be as sharp as possible.

4.3.7 Recovery Column

For the recovery column, we have the choice (Fig. 4.15d) between running an absorber simulation, or finishing the stripping and restarting EXSEP. For the first option, we need to enter data, as explained previously in section 4.1.9. We choose 'No' and enter the last window of the recovery procedure.

4.3.8 Final Flowsheet

In Figure 4.16a-b, we compare the final flowsheet for this problem, with and without bypass, respectively. In the second column, EXSEP specifies the amounts bypassed. We observe how small they are (Fig. 4.16a), which explain why the number of stages and lean gas flow rate do not decrease much when bypass is used (see section 4.2.6). To terminate the program, we use the procedure as explained in section 4.1.11.



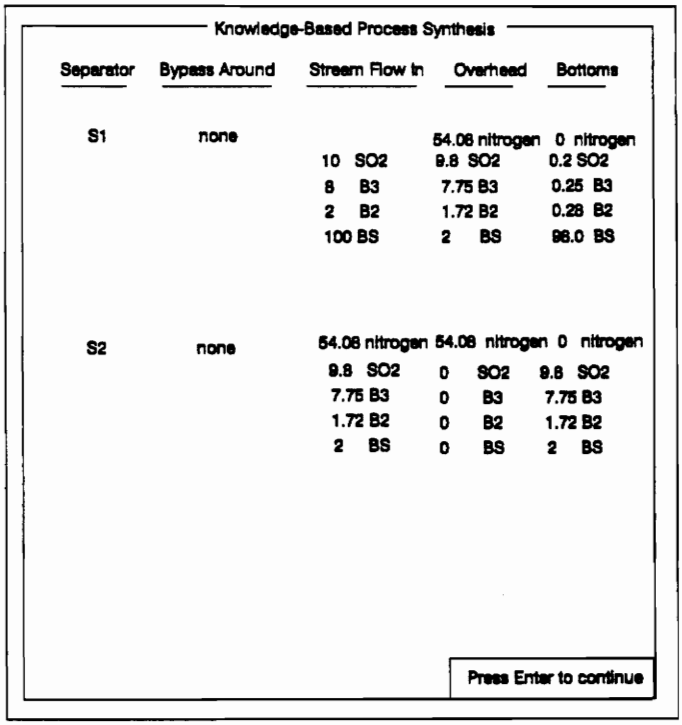


Figure 4.16a-b : (a) Final Flowsheet with Bypass; (b) without Bypass.

4.4 An Example of Extraction

4.4.1 Data Loading

The data file is *'lleproex.dat'*. We show it in the following lines:

```
henry('Bec',99.37).
henry('Phe',1.91).
henry('H2O',0.0031).
flow(p1,'Bec',0.000).
flow(p1,'Phe',0.0043).
flow(p1,'H2O',73.1399).
flow(p2,'Bec',0.0001).
flow(p2,'Phe',0.8633).
flow(p2,'H2O',0.5377).
mole_fraction(solvent,'Bec',0).
mole_fraction(solvent,'Phe',0).
mole_fraction(solvent,'H2O',0).
initial_set([p1,p2]).
initial_components(['Bec','Phe','H2O']).
solvent(benzene).
key_component(lle,'Phe'). % lle = liquid liquid extraction
```

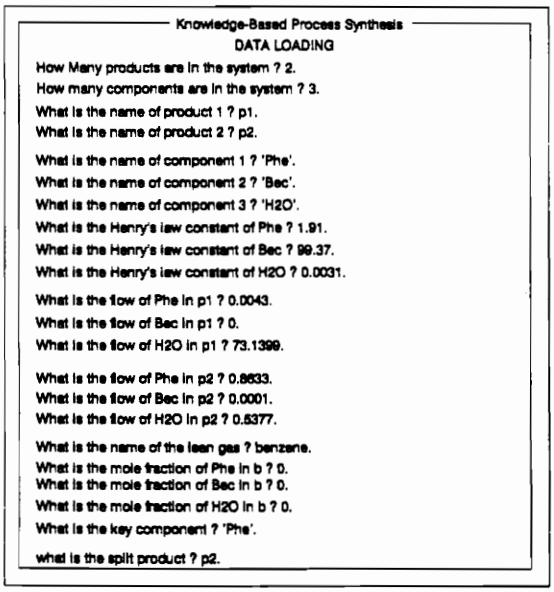
Figure 4.17a shows the corresponding question/answer session.

4.4.2 Component Assignment Matrix (CAM)

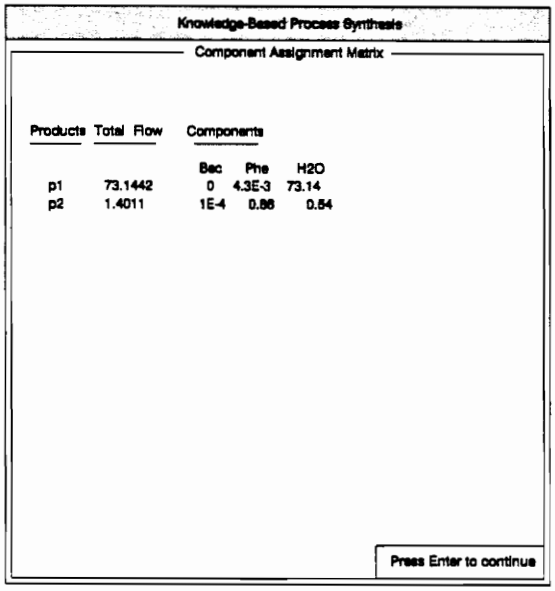
Fig. 4.17b shows the CAM for the extraction example.

4.4.3 Bypass

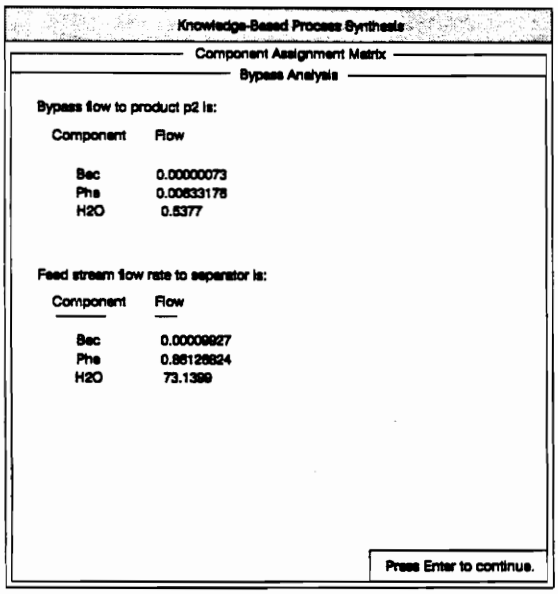
We choose to bypass 100 % of both products. We describe below the flowsheet solutions with and without bypass. Figure 4.17c-d compare the feed to the column with and without bypass.



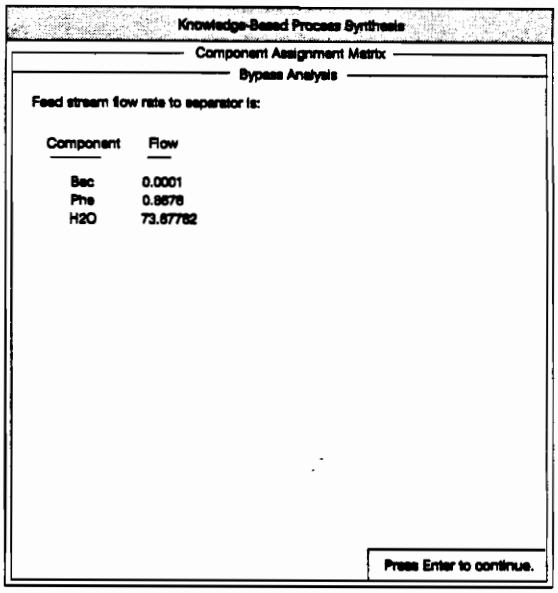
a



b



c

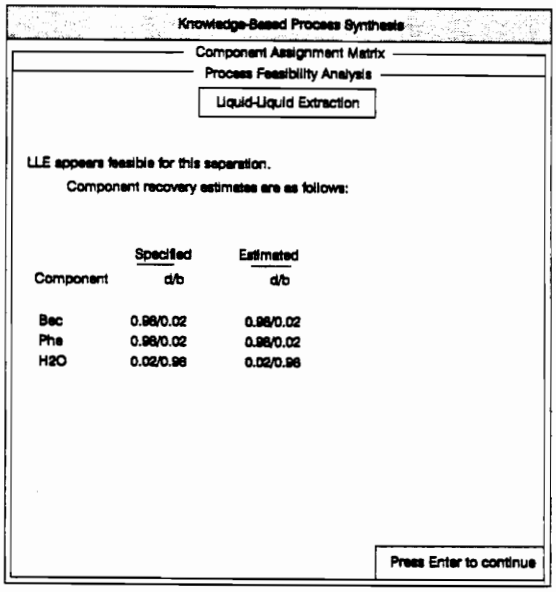


d

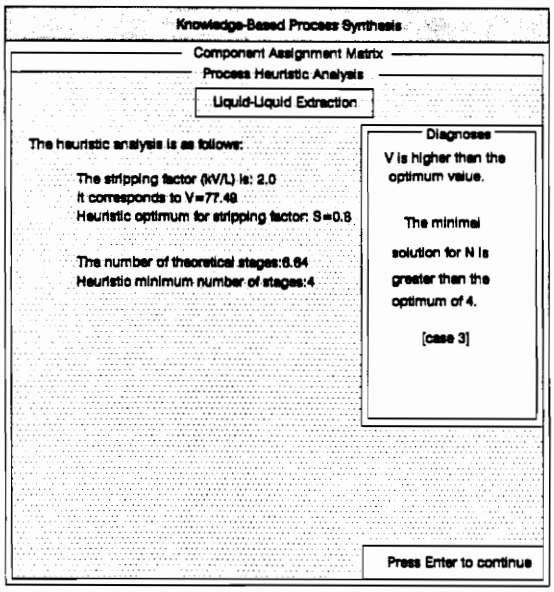
Figure 4.17a-d : (a) Data Loading; (b) CAM; (c-d) Column Feed with and without bypass.

4.4.4 Solution

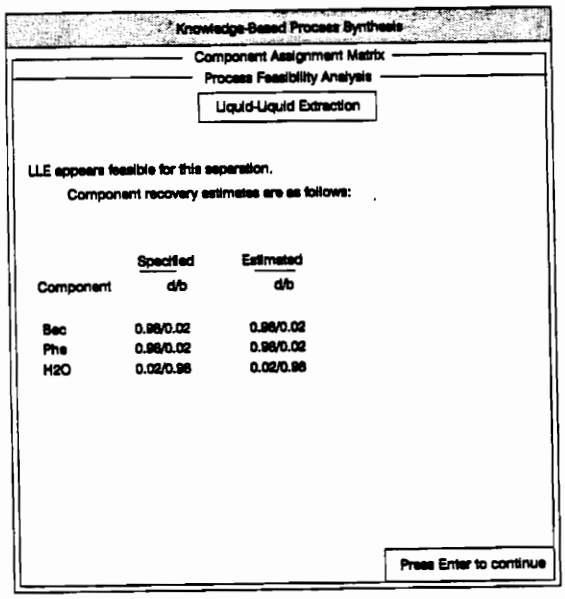
We present in Figure 4.18a-d part of the solutions with bypass. Figure 4.19a-d shows part of the solutions without bypass. We observe that for the same extraction factor, the number of stages systematically decreases by as much as 30% when using bypass. In this example, bypass has a very favorable effect (it does not have much effect for the stripping example).



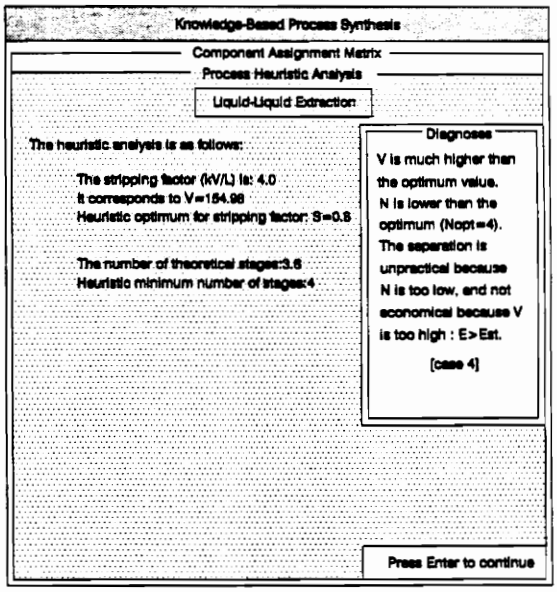
a



b

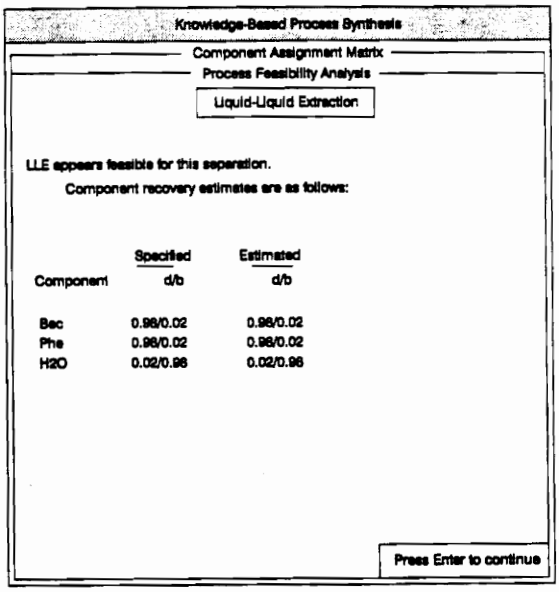


c

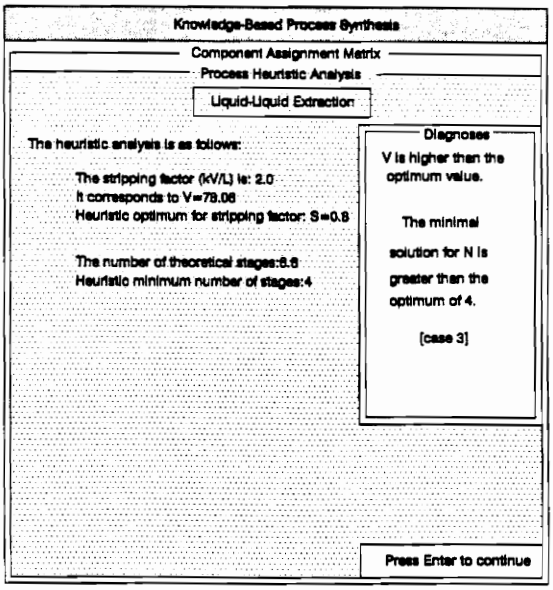


d

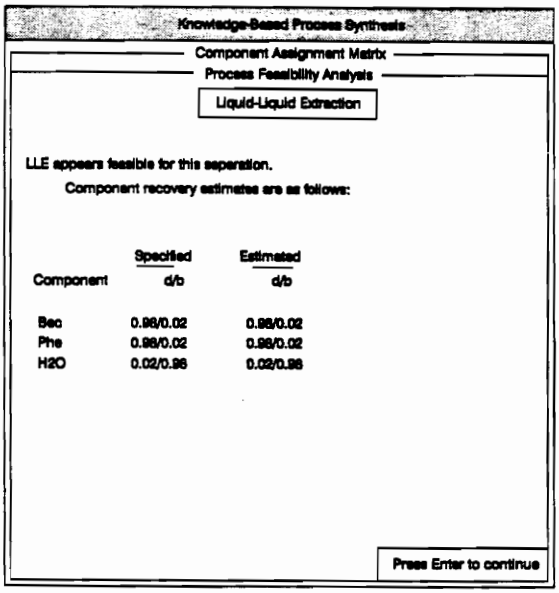
Figure 4.18a-d : Two Solutions with Bypass, (a-c) Component-Recovery Ratios, (b-d) Explanation.



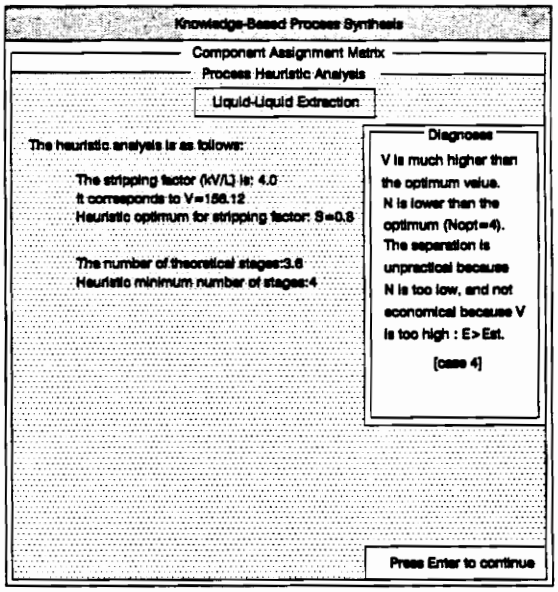
a



b



c



d

Figure 4.19a-d : Two Solutions without Bypass, (a-c) Component-Recovery Ratios, (b-d) Explanation.

4.4.5 Search-Range Diagnosis

In Figure 4.20a, we observe that the explanation for the range of the extraction factor is different form those for absorption and stripping examples that we saw previously. In this case, we have $E_{opt}=0.8 < E_{min}=2 < E < E_{sl}=4$. Thus, the search will not take into account the optimal factor, which is 0.8 according to heuristics.

EXSEP's explanation illustrates a user mistake. Indeed, we set E_{min} too high. It should be smaller than 0.8, and that is exactly what EXSEP advises. If we set E_{min} to 0.6, the solution is actually the same, because **for this example**, the optimal value is not a feasible one.

4.4.6 Choice of the Best Solution

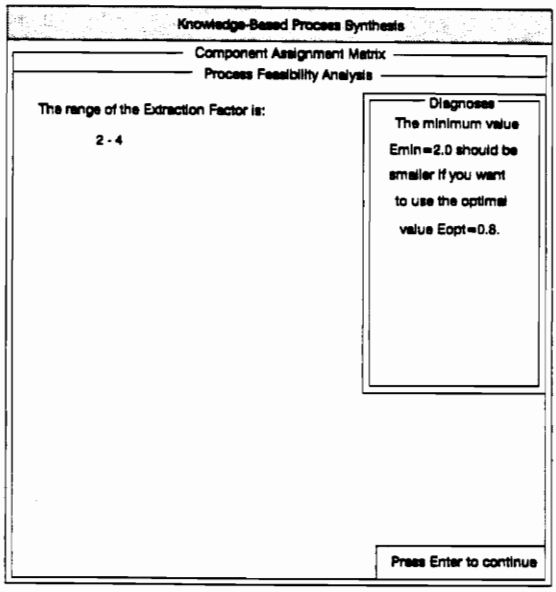
We compare in Figure 4.20c-d, the best solutions proposed by EXSEP for the two methods with and without bypass, respectively. For the same extraction factor, the solvent flow rate V is reduced by 1.2% (the liquid feed flow rate L is also reduced, so E stays constant). The number of stages is reduced by 30 %.

4.4.7 Recovery Column

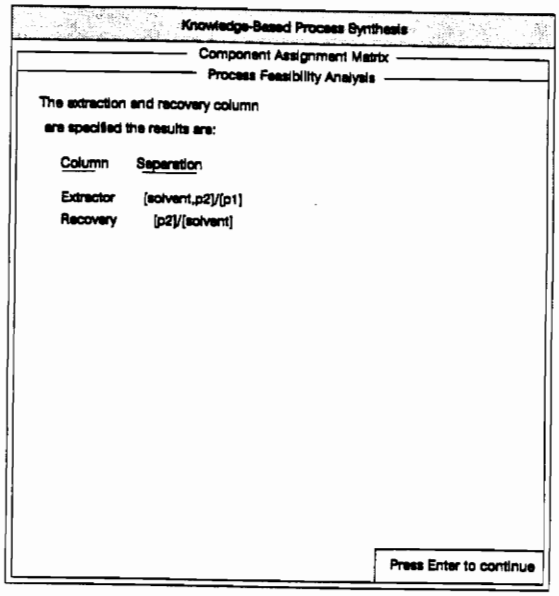
For extraction, the recovery column is not directly calculable. If a distillation must be used for the solvent recovery, we finish the extraction simulation, and choose "Synthesize Another" in the last menu. Figure 4.20b shows the recovery procedure.

4.4.8 Final Flowsheet

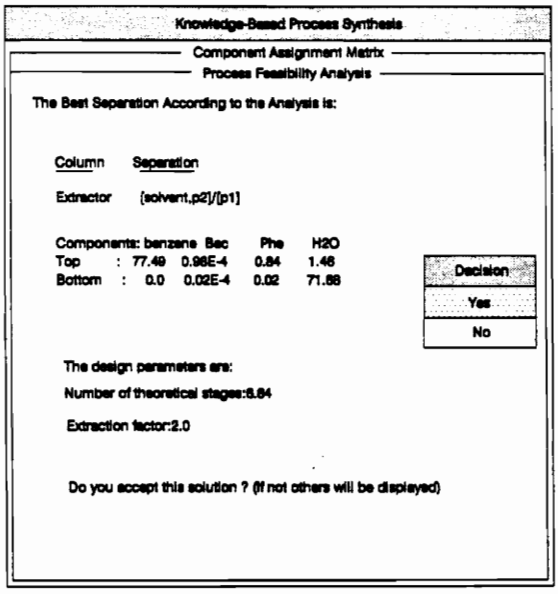
Figure 4.21a-b compare the final flowsheets with bypass and without bypass.



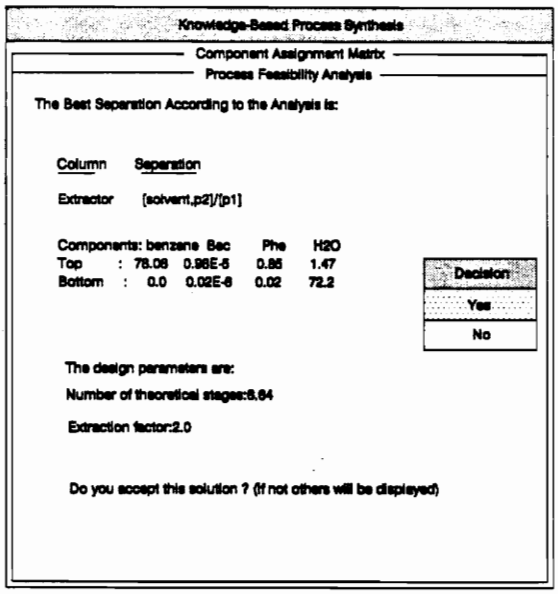
a



b

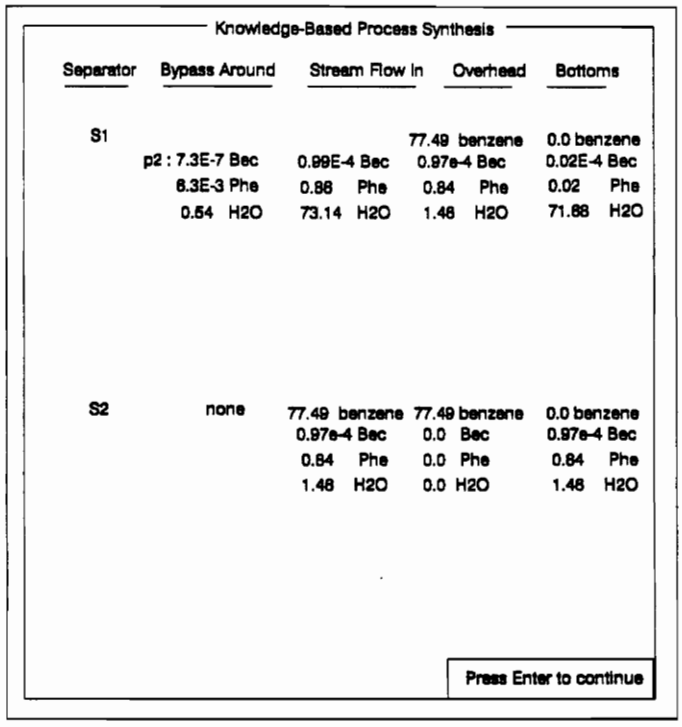


c



d

Figure 4.20a-d : (a) Range Explanation, (b) Recovery Procedure, (c-d) Best Separation with and without Bypass.



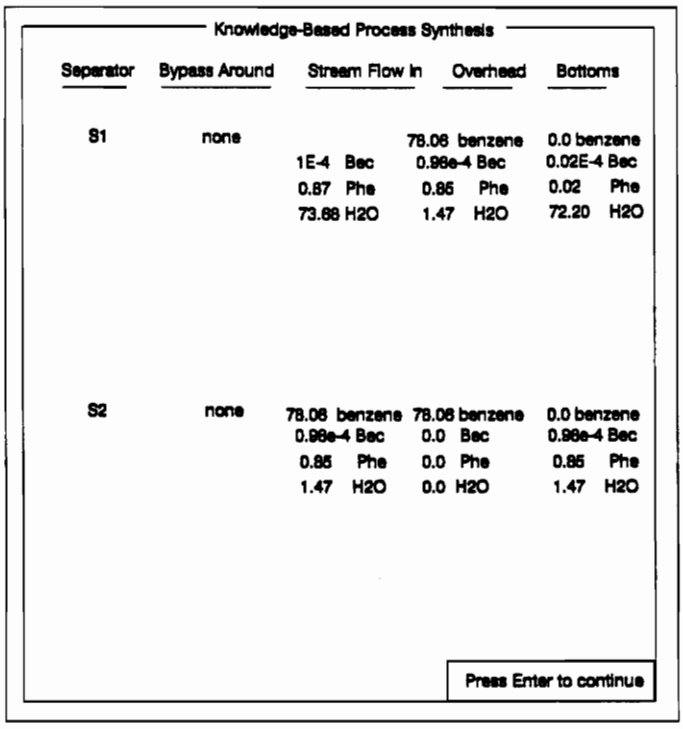


Figure 4.21a-b : Final Flowsheet (a) with Bypass, (b) without Bypass

4.5 Summary

In this chapter, we demonstrated, with one example for each separation (absorption, stripping, and extraction), how simple it is to use EXSEP with its menu-driven decision tools. The user can run the compiled program from DOS in a matter of seconds on almost any IBM-PC. The user enters problem data directly from the keyboard, or else types the data in a file. The user can verify the entry when EXSEP displays the matrix representation (CAM) of the problem statement.

EXSEP performs the bypass and feasibility analysis to offer the best thermodynamically and economically feasible solution according to its knowledge base and inference mechanism. But the user can override EXSEP's choice in order to generate as many solutions as desired. EXSEP also incorporates an explanation facility to help the user in understanding its reasoning.

CHAPTER 5

AI Perspective of EXSEP

EXSEP is written in Prolog, an artificial inteligence (AI) language. In this chapter, we describe EXSEP from the AI perspective. We discuss the search strategy (method used to find flowsheet solutions). We describe the knowledge representation and the program structure in each of the three parts of the search strategy (plan-generate-test). We also discuss the advantages of Arity/Prolog compared to Turbo-Prolog. Finally, we describe the method used to make a compiled version of EXSEP.

5.1 Introduction

5.1.1 Search Strategy

EXSEP uses the plan-generate-test search strategy.

- **Plan**: Data acquisition and knowledge representation.

EXSEP obtains data from a file. These data are facts, rules and heuristics (called 'knowledge' in AI). They are added to the database, and the problem statement is converted to a component assignment matrix (CAM) to facilitate list processing by Prolog.

- **Generate**: Feasible solution generations.

A solution is the number of stages, the solvent flow rate (or separation factor) and the actual component- recovery ratios. The "generate" stage is mostly performed by the *Kremser* clause (see section 5.3.2).

- **Test**: Feasible solution ranking and best solution selection.

In the "test" stage, EXSEP ranks the solutions according to heuristics and chooses the economical ones. It displays the best one according to the heuristic analysis. The user can override this decision and choose another feasible solutions.

The "generate" stage actually performs some of the testing to increase the efficiency of the search. Thus, there is no clear difference between the generator and the tester.

5.1.2 Overall Module Programming

EXSEP has 8 different modules. Some modules (Main, Bypass and Utility) control the others and some modules are specialized. The SST and Split modules are used only for distillation; the Absorb, Stripping, and LLE modules are the MSA-based modules for absorption, stripping and liquid-liquid extraction, respectively.

- MAIN MODULE: Shown in Figure 5.1, it contains program control, interface driver [1.1] and data-acquisition driver [1.4] (note: in the following discussion, we label the branches and steps in Fig. 5.1 to 5.8 by numbers within brackets, e.g., [1.1] in Fig. 5.1).

- BYPASS MODULE: It performs the complete bypass analysis [2.2].

- SST MODULE: It performs the feasibility analysis for ordinary distillation and passes the feasible split to the split module [3.1].

- SPLIT MODULE: It heuristically ranks the feasible splits for ordinary distillation [3.2].

- ABSORB MODULE: It performs every stage of the search for absorption [4.1]. We

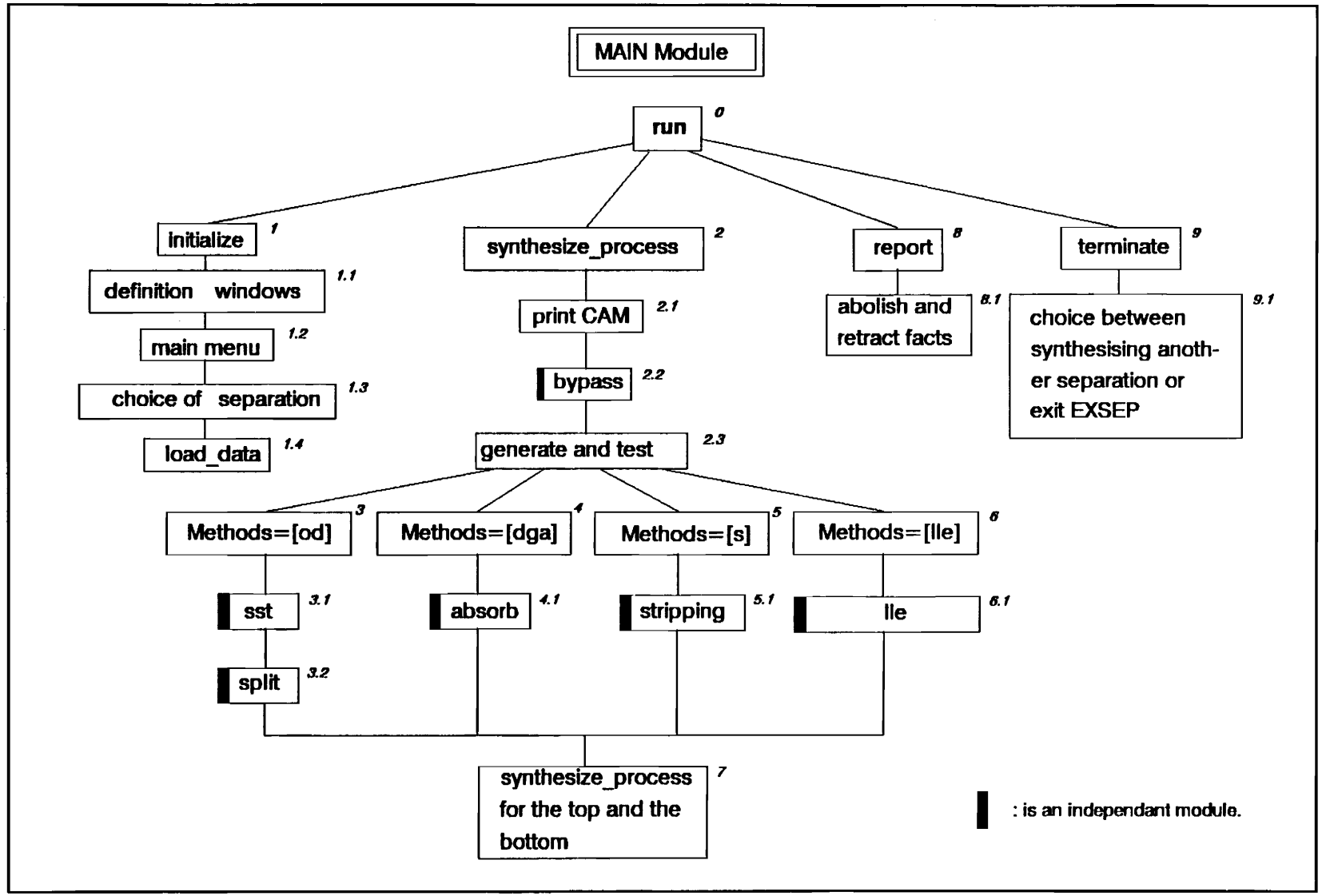


Figure 5.1 : Link Between Modules

detail and explain it in Figure 5.2 and the following paragraphs. It contains some of the utility functions used by the other two MSA-modules.

• STRIPPING MODULE: It has the same structure and characteristics as the ABSORB module, but for stripping [5.1].

• LLE MODULE: It also has the same structure and characteristics as the ABSORB module, but for extraction [6.1].

• UTILITY MODULE: It performs the list processing, and contains other utilities used in all the other modules.

The Main Module (Fig. 5.1) also reports the solutions [8] and manages the program continuation [9]. In the following, we shall describe in greater detail how the MSA-modules work, and what their common structure is.

5.2 Plan Stage

Figure 5.2 shows the overall structure of the MSA-modules. This structure is common to the three modules. We use X as a generic term for the separation factor. It can be A (absorption factor), S (stripping factor), or E (extraction factor).

5.2.1 Component Assignment Matrix Representation

In branch [2], EXSEP performs the list processing to convert the problem data to a CAM (component assignment matrix). The user enters the flow rates and K values in Prolog syntax, for

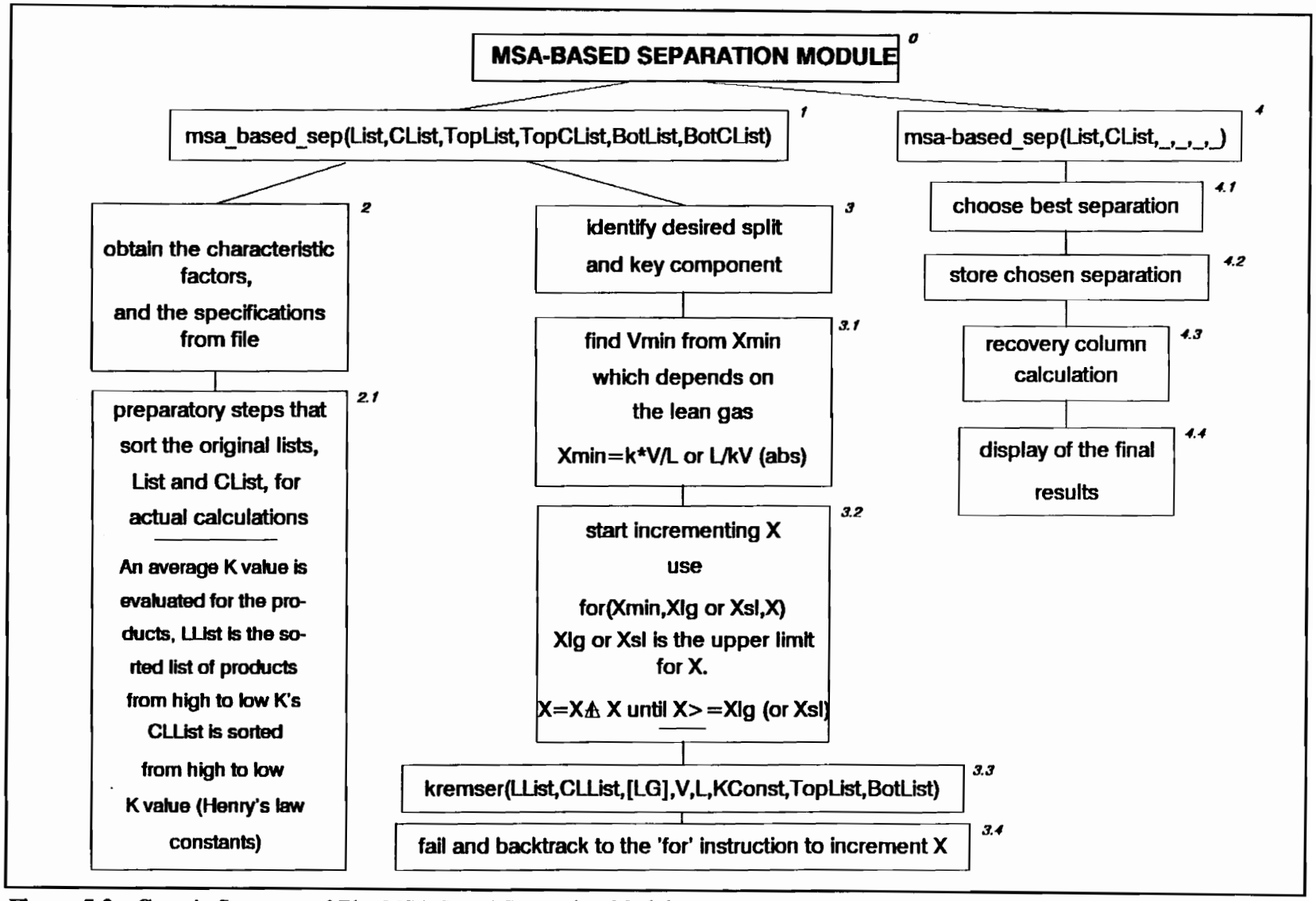


Figure 5.2 : Generic Structure of The MSA-Based Separation Modules.

example: *flow(p1,'CH4',10).* . EXSEP makes lists of products, components and flow rates, and sorts these lists according to the CAM characteristics. As an example, the Absorb module of EXSEP sorts the list of the components in decreasing values of their Henry's law constants.

5.2.2 Split Determination

In the same way, EXSEP identifies the desired split [3] and creates the top and bottom lists (TopList,TopCList, BotList, and BotCList).

5.3 Generate Stage

5.3.1 Multiple Solution Generations

To generate multiple flowsheet solutions, EXSEP uses an incremental procedure ([3.2] in Fig. 5.2). The starting point of the procedure depends on the characteristic minimal separation factor, Xmin, [3.1], given by heuristics. The end point of the procedure is given by Xsl (called Xlg for stripping). This upper limit is also given by heuristics, and it depends on the solvent. EXSEP obtains Xmin, Xsl, and ΔX from the data file [2].

Each incremental step corresponds to a separation factor X. With the feed flow rate and the K value of the key component, the solvent flow rate is easy to find:

$$L_{solvent}=K_{keycomp.}\ V_{feed}\ X \quad (case\ of\ absorption) \tag{5.1}$$

With the solvent flow rate (L), the feed flow rate (V) and the K value of the key component (KConst), together with the sorted list of products (LList), components (CLList), top and bottom products, (TopList) and (BotList), EXSEP evaluates the feasibility of the separation characterized

by all these variables [3.3]. To generate several solutions, the program always fails after the *Kremser* clause [3.4]. Then it backtracks to the *for* predicate [3.2]. This backtracking continues until the *for* predicate itself fails. Then the simulation is finished.

5.3.2 *Kremser* Clause

To evaluate the split feasibility, EXSEP calls the *Kremser* clause. Figure 5.3 shows the logic structure of the *Kremser* clause. The list of the real functors corresponding to the indices appears in appendix A.4. The first instruction [1] of the Kremser clause is to calculate the number of stages N (see Fig. 5.4). If EXSEP cannot evaluate N (when: ln(R) with $R<0$), the system backtracks to increment X [2]. When EXSEP can evaluate N, the first heuristic is tested, [3] or [12].

EXSEP enters branch [3] if heuristics concerning the number of stages N, are satisfied - the number of stages N must be greater than a given value Nopt. EXSEP tests if X is smaller than the optimal value (e.g. Aopt=1.4 for absorption; see section 2.2.2.1). If X satisfies this condition [4], EXSEP calculates the component-recovery ratios [4.1] and checks if they correspond to the specifications (an error of 10 % is allowed). If they do [5], EXSEP explains the situation [5.1] - see section 5.4.1.1, case #1. Then the program stores the separation in the data base [5.2], and increments the X factor to try to find another solution.

If the component-recovery ratios are not satisfactory [6], and the separation is not feasible, then EXSEP **forces** X to be greater than the actual value and sets it equal to the optimum separation factor Xopt [6.1]. This corresponds to an increase of the solvent or lean-gas flow rate. Doing so can enable the component-recovery ratios to meet the specifications. But it "artificially" increases the solvent flow rate which is contrary to the optimization of this parameter. Then, if

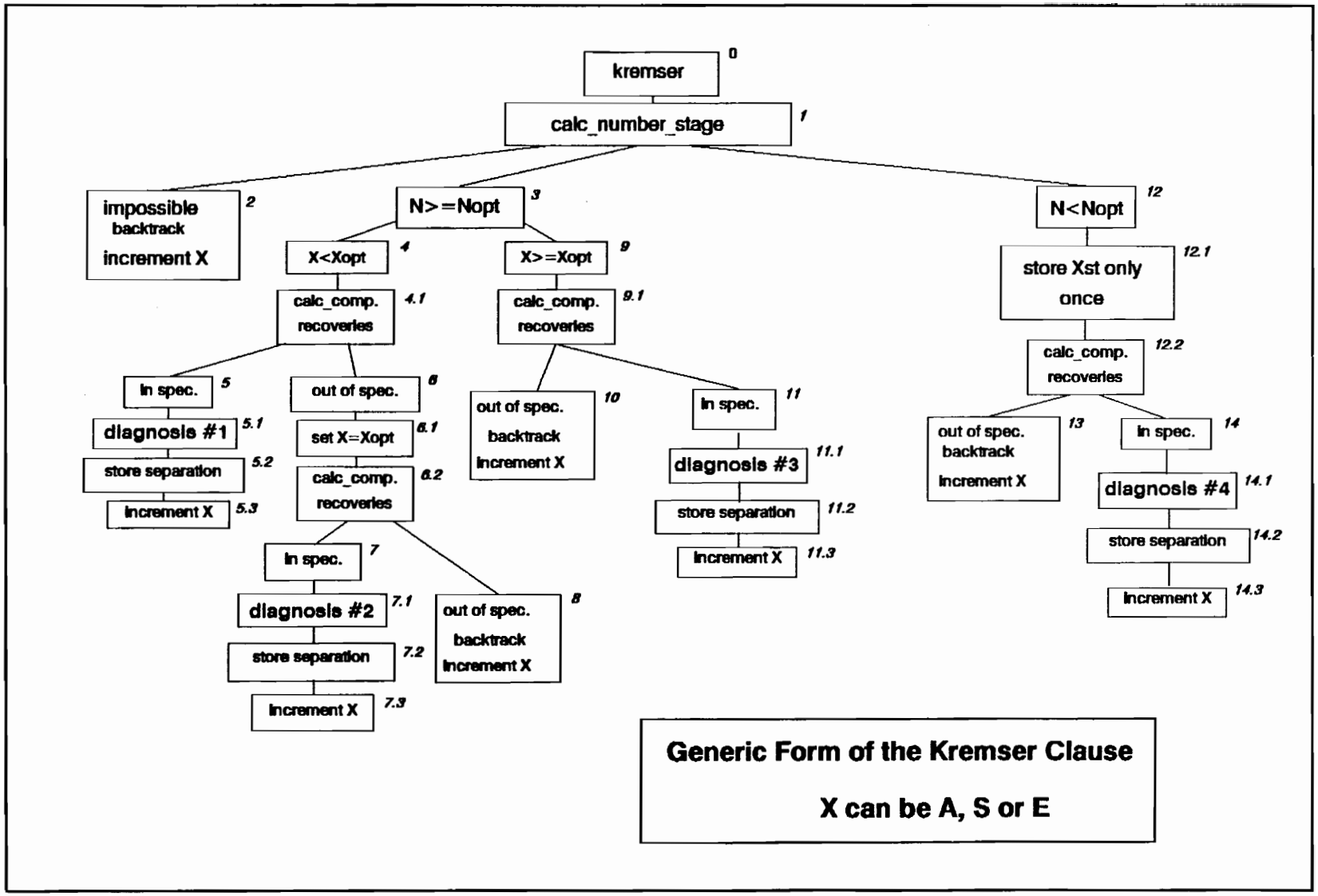


Figure 5.3 : Generic Structure of The Kremser Clause

a solution exists for this imposed solvent flow rate, it will not be an optimal solution. When component-recovery ratios do meet the specifications, the same procedure as [5] to [5.3], is applied in [7] to [7.3]. If they do not, EXSEP increments X [8].

Returning to [3], we note that if X is greater than Xopt [9], then in [9.1], and [10] to [11.3], the same process is applied as in [6.2], and [7] to [8]. Returning to [1], we see that if N is smaller than Nopt [12], the design of the column does not satisfy the first of the design heuristic (see section 2.2.3.d). However, EXSEP keeps trying to find a solution [12.2]. But when EXSEP finds a "solution" [14], the program diagnoses it to be infeasible [14.1]. In [12.1], EXSEP stores the value of X when the branch [12] is entered the first time. This value Xst is the separation factor when the number of stages is optimal, and is used in the explanation process.

EXSEP evaluates the mole fraction in the output product and the number of stages. For example, when the separation factor equals 1, the evaluation of the number of stages could fail because of a division by 0 (ln(1)=0). As we describe in section 5.3.2.1 below, EXSEP considers all the possibilities of infinite limits, or division by zero.

5.3.2.1 *calc_number_stage* Clause

Figure 5.4 shows how EXSEP calculates the number of stages. The main restriction is when the separation factor A (absorption, here) equals 1 [2]. When the factor is different from 1 [1], EXSEP needs to check also if the ratio shown in [1.2] is strictly positive. If it is not, the clause fails, the backtracking makes the *Kremser* clause fail, and a new separation factor (increment) is studied.

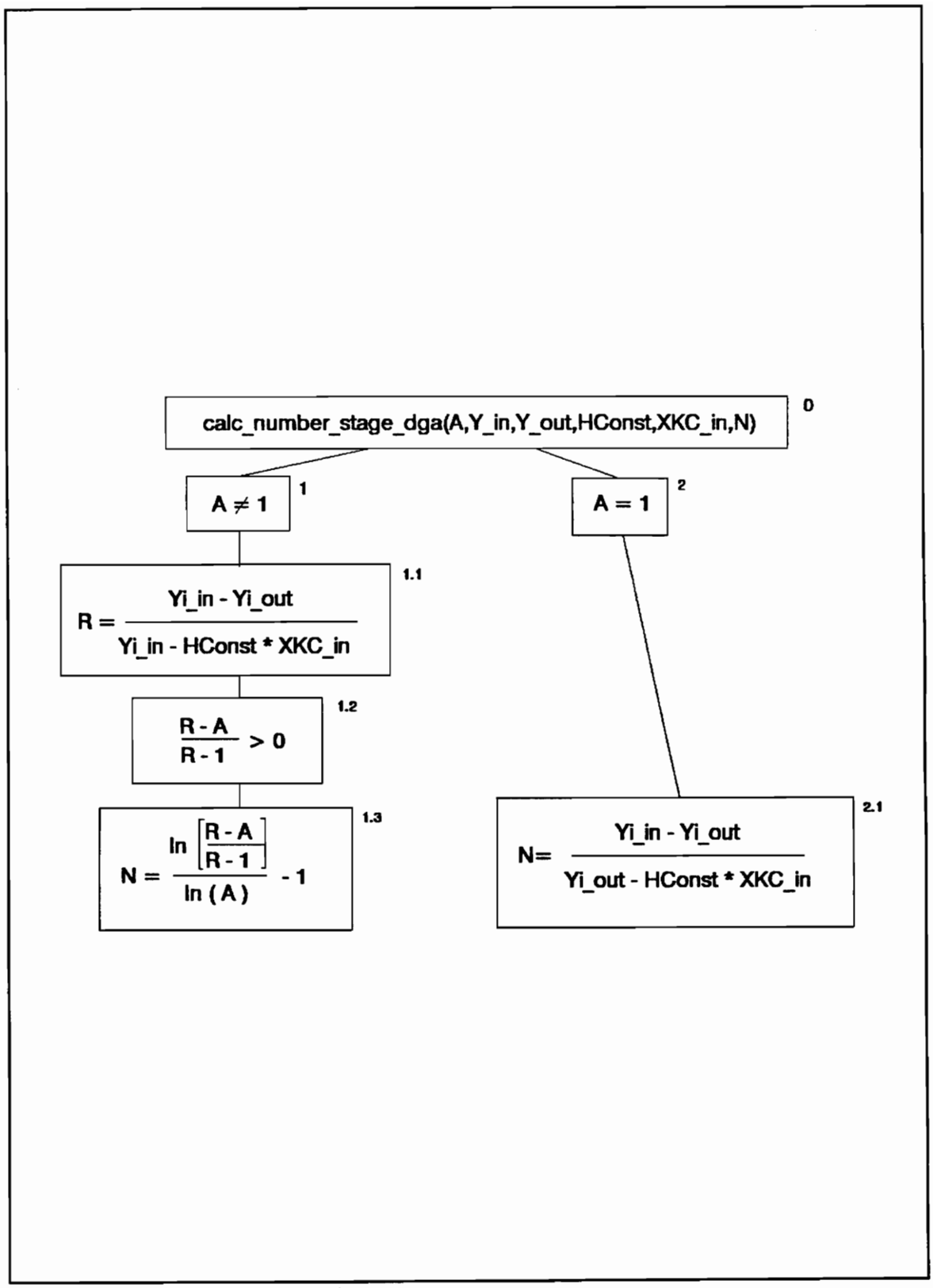


Figure 5.4 : calc_number_stage_dga

5.3.2.2 *calc_Yi_out_actual* Clause

This clause is the evaluation of the molar fraction of the non-key components in the outlet vapor stream: yi_{out} (xi_{out} for stripping and extraction). In Figure 5.5, we show the possible equations used. They depend on the magnitude of the separation factor, [1], [2], and [5], and its (N+1)th power, [3] and [4].

5.3.3 *recovery* Clause

When a flowsheet is generated for the chosen separation, EXSEP simulates the solvent "recovery column". We can choose between two alternatives for the *recovery* clause. We can choose to start another simulation with another separation module. Or we can assume that the split is sharp and feasible and EXSEP displays the split **without** heuristic analysis (see Fig. 5.6).

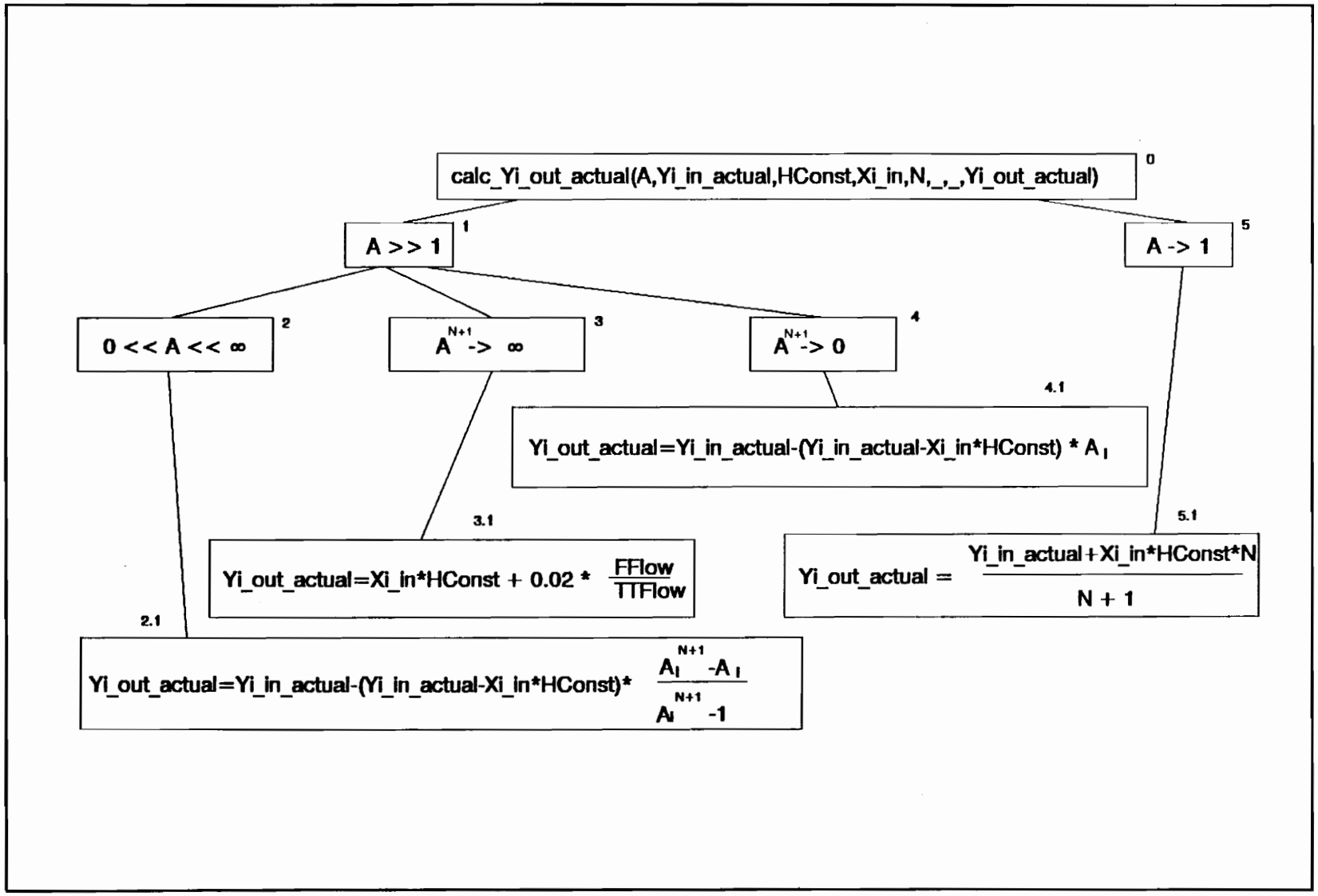


Figure 5.5 : calc_Yi_out_actual

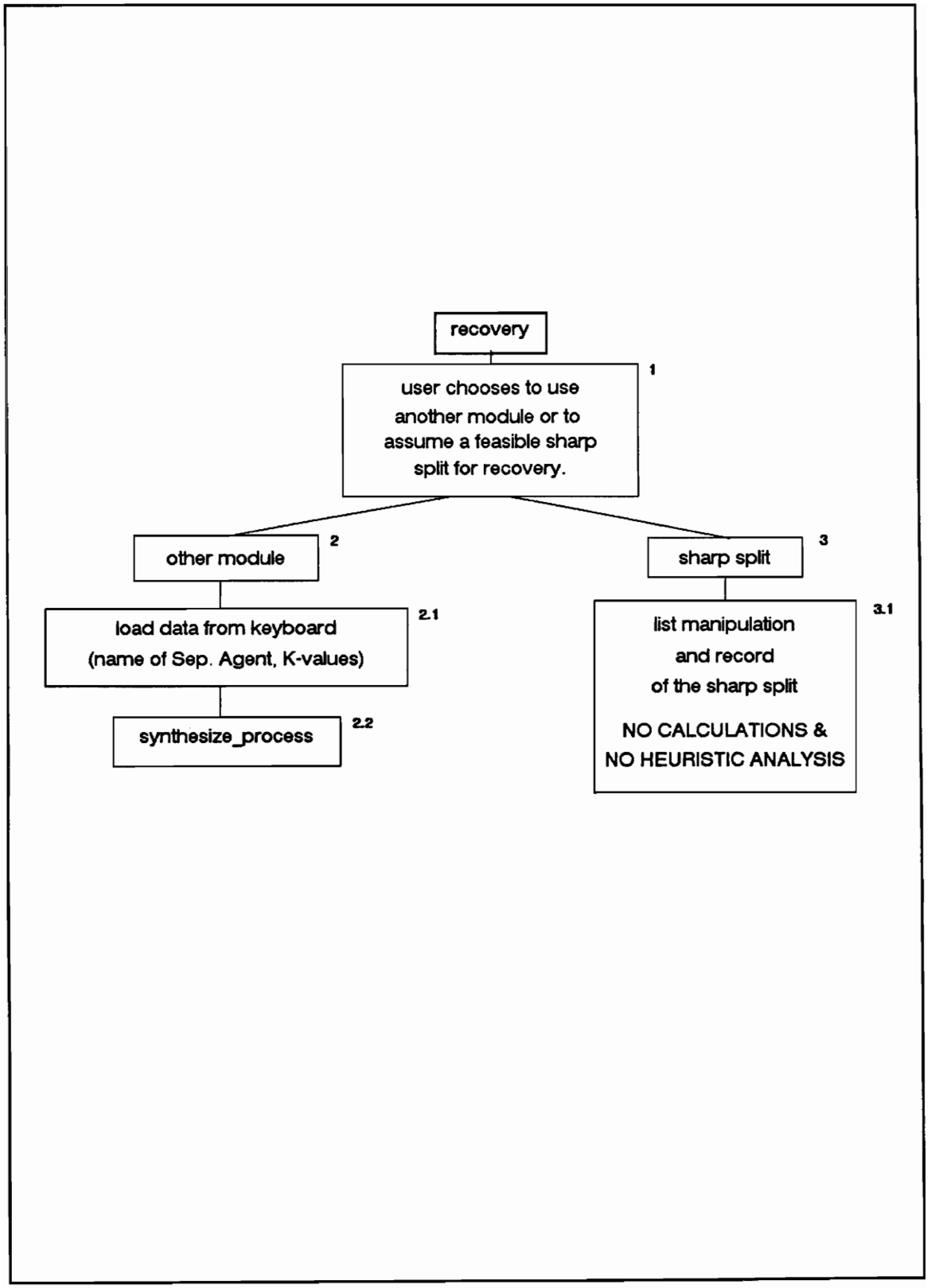


Figure 5.6 : *Recovery* Clause.

5.4 Test Stage

5.4.1 Explanation Facility

5.4.1.1 Solution Position

As discussed in section 3.3.2, EXSEP explains the "quality" of a solution according to the deviation between the actual number of stages and separation factor, and their respective optimal values. Figure 5.7 shows the four possible configurations.

Case 1 corresponds to branch [5] of the Kremser clause (Fig.5.4). In this case, EXSEP explains that the solvent flow rate is lower than the optimum and N higher than the optimum.

Case 2 corresponds to branch [6] in the Kremser clause (Fig. 5.4). It occurs when we impose the separation factor X to be its optimal value Xopt, which is equivalent to increasing the solvent flow rate. This is not an optimal solution because N is higher than the number of stages corresponding to Xopt ('o' mark on the graph in Fig. 5.7).

Case 3 is considered to be a good configuration. Indeed, X and N are close to the optima.

Case 4 is an unpractical configuration because N is too low and the solvent flow rate too high.

This explanation capability is based on the assumption that the four separation factors are in the following order :

$$Xmin < Xopt < Xst < Xsl$$

This is the best configuration for an efficient search. However, when they are not in this

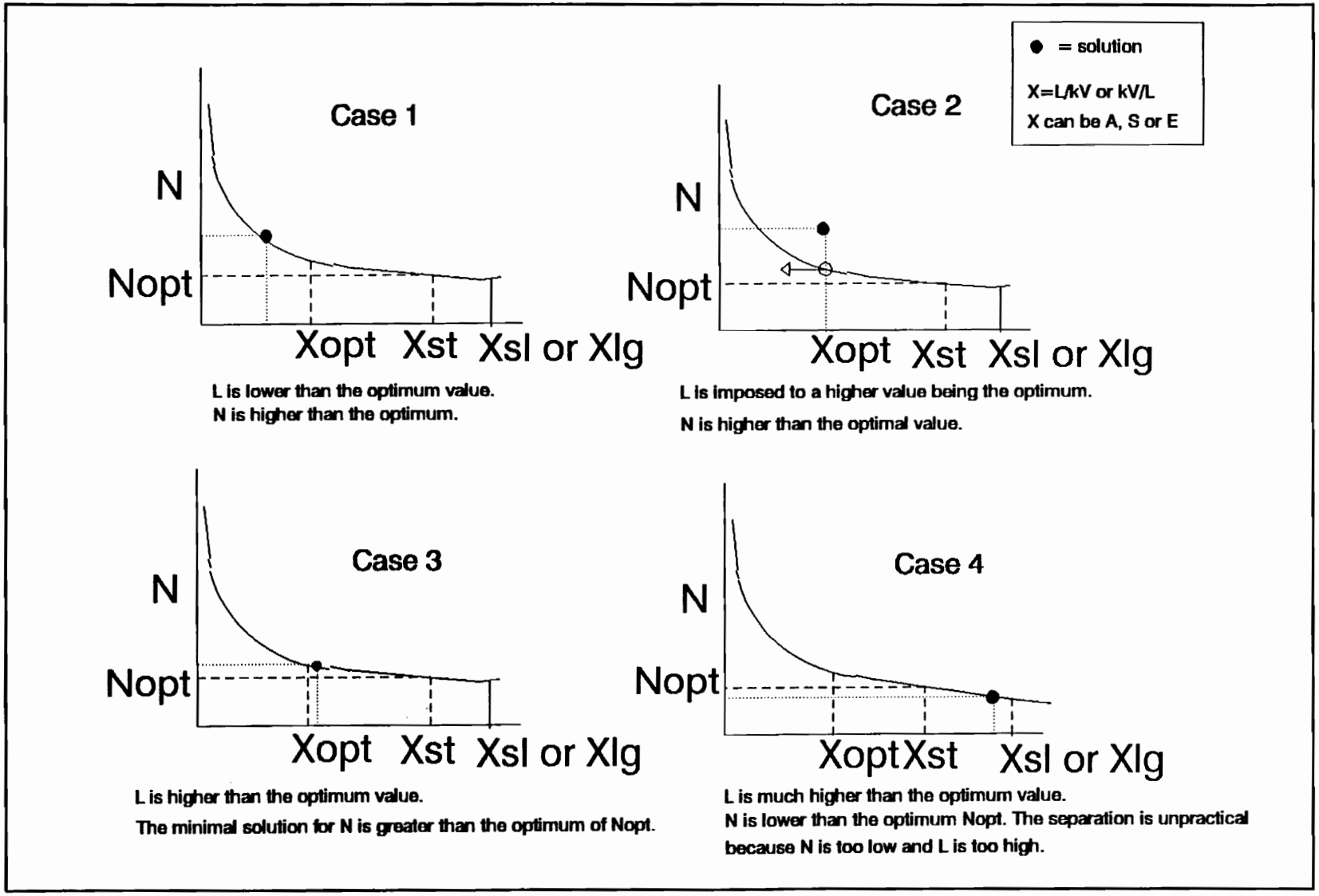


Figure 5.7 : Quality of the Solution Diagnosis.

order, EXSEP analyzes the actual order, gives explanations, and advices on the search-range size.

5.4.1.2 Search-Range Size

There are 24 possible combinations of the 4 factors: Xmin, Xopt, Xst, Xsl. In Figure 5.8 and 5.9, we show the structure of the *diagnosis* clause, and display the 8 most important cases. For example, if Xmin (the lower limit) is higher than Xsl (the upper limit), no calculation occurs (case 2). As the user has the possibility to change the values of the 4 separation factors, errors like this one could occur. EXSEP takes care of them.

One of the most important among the 8 cases is when Xst is higher than Xsl (case 4). This means that the upper limit is smaller than the value of the separation factor when the number of stages becomes optimal. In such a case, feasible solutions could exist with a separation factor higher than the upper limit, and a number of stages still higher than the optimum (see example in section 4.1.7).

5.4.2 Selection of the Best Solution

When several solutions are generated by EXSEP, it has to choose the best one. A solution is satisfactory when the number of stages is close to the optimum value (Nopt given by the heuristic) and when the separation factor is also close to the optimum Xopt.

5.4.2.1 Coefficient of Separation

In section 3.4.1, we presented the coefficient of separation CS. The predicate, *calc_CS(Dev,N,X,CS)*, is thus a function of the average deviation, Dev, between the actual and specified component-recovery ratios. Dev is calculated by the predicate *calc_dev*. CS is also

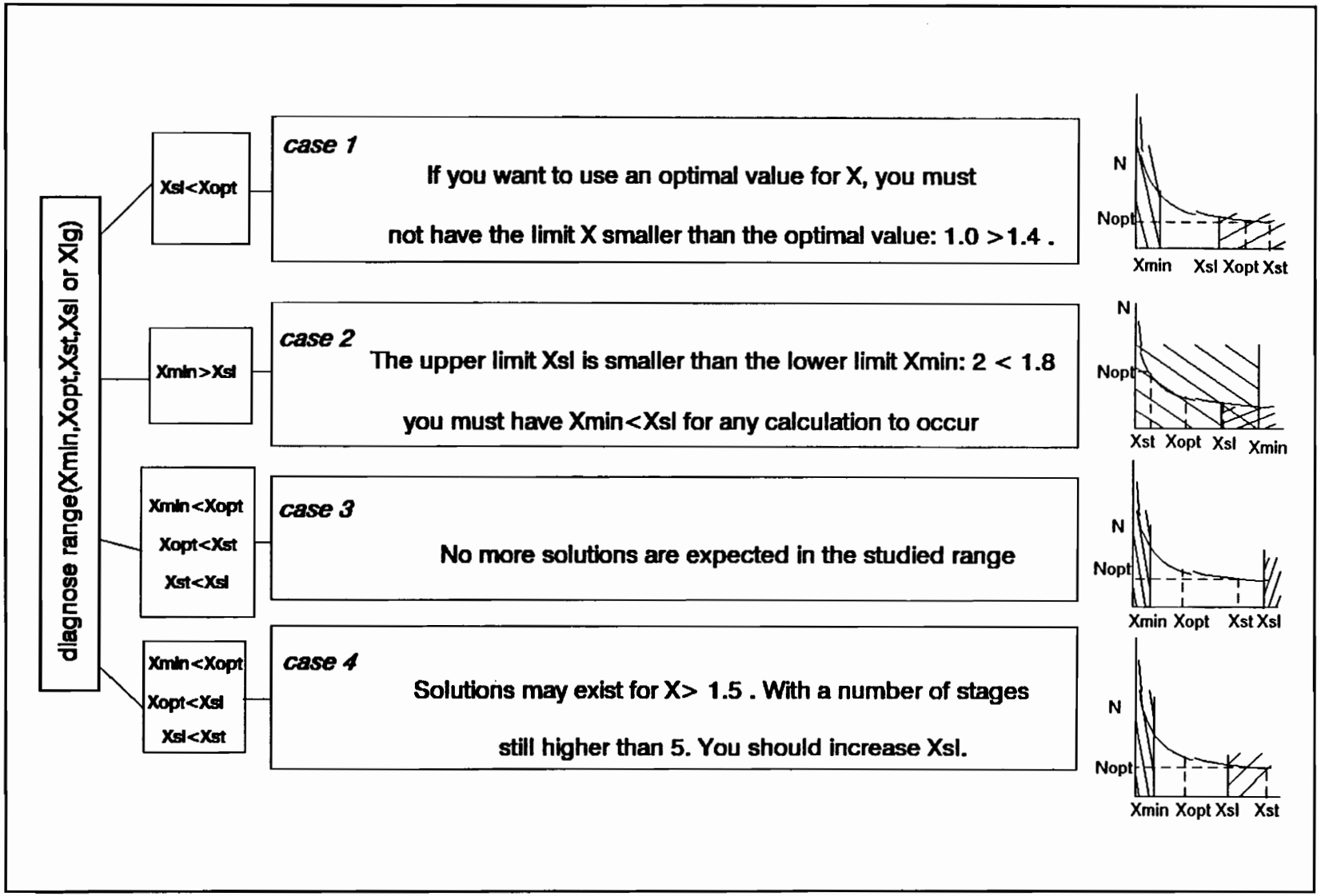


Figure 5.8 : Search-Space Diagnosis, Cases 1 to 4.

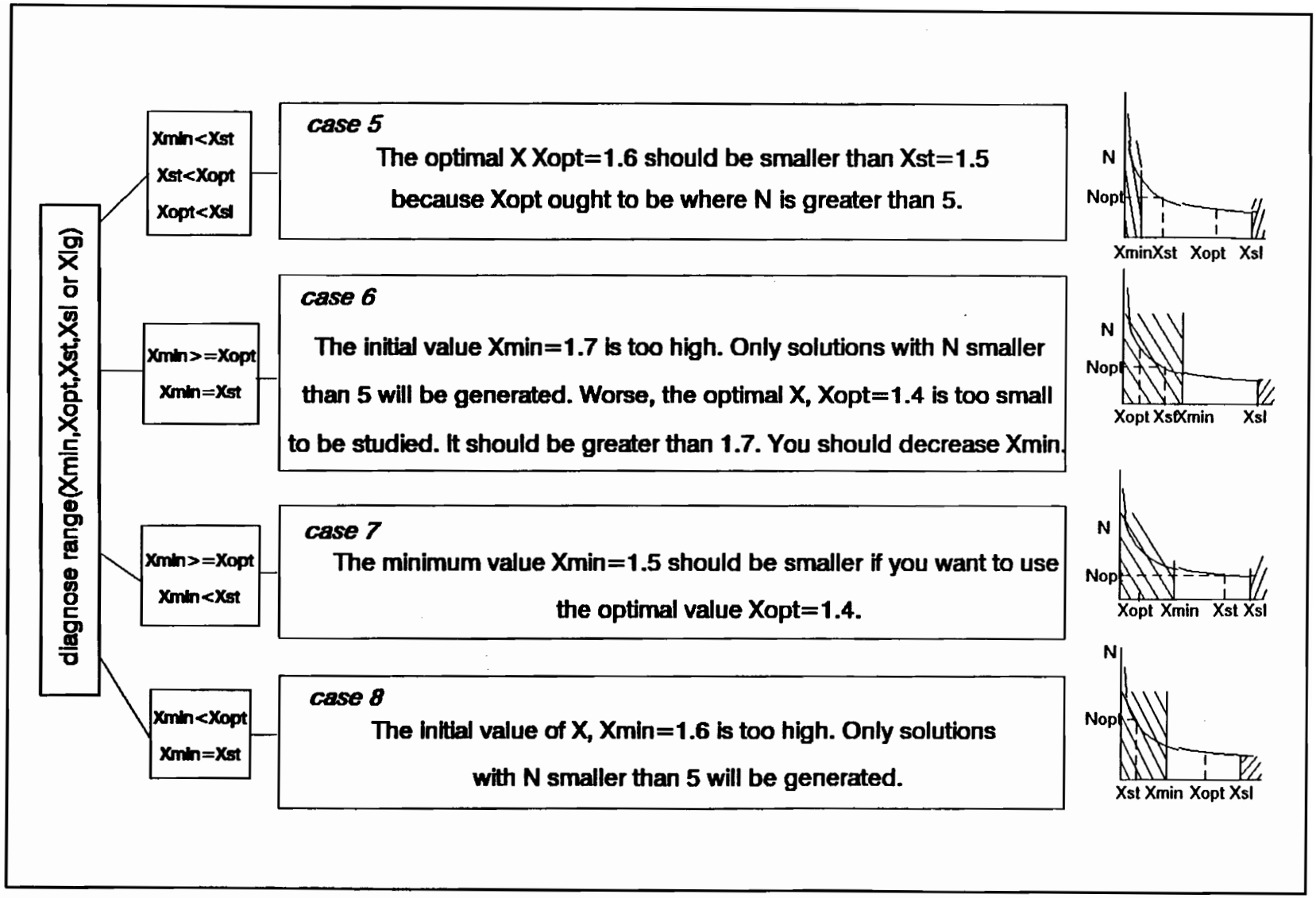


Figure 5.9 : Search-Space Diagnosis, Cases 5 to 8.

function of N the number of stages, and X the separation factor. The CS factors for all the feasible separation are stored in a list, and ranked.

5.4.2.2 Database Management and Final Selection

Each feasible solution is stored in an independent database. This database is called "absorb" for absorption, "strip" for stripping and "lle" for extraction. Inside the database, the separations are indexed with their respective CS factors (see section 5.5.1.2). To obtain the best separation, EXSEP displays the solution that has the highest CS factor (best solution), and asks the user to grant or deny this selection. If the user denies the solution EXSEP suggests, EXSEP displays the solution with the second highest CS factor, and so on (see Fig. 5.10).

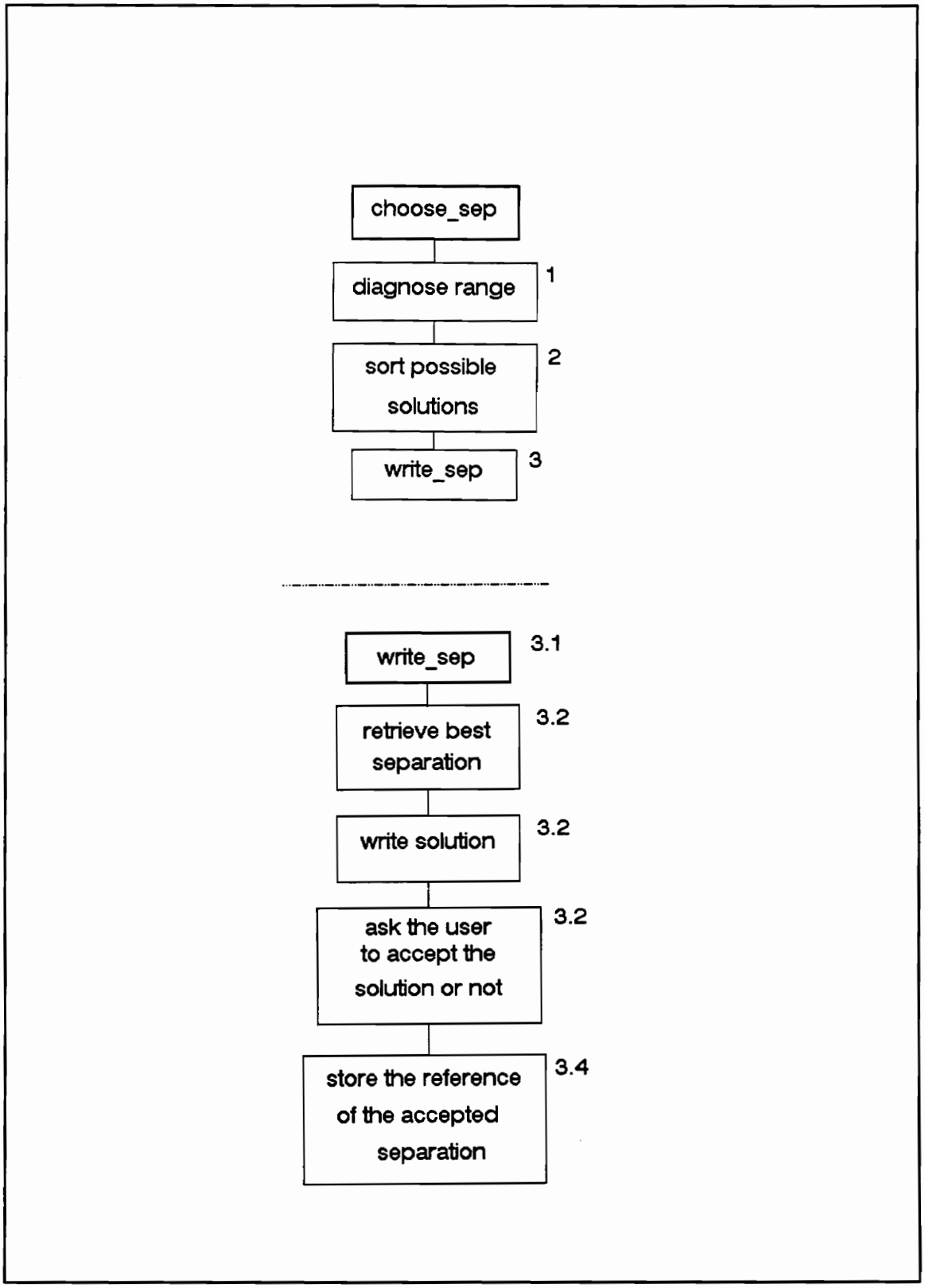


Figure 5.10 : Final Selection Chart.

5.5 Use of Arity/Prolog Built-In Tools

Arity/Prolog is a computer language distributed by Arity Corporation. When compared with Turbo-Prolog (Borland), its main advantage is the "interpreter" feature.

BASIC, for example, is an interpreted language. Each line is processed one after the other. When the program fails, it is easy to find where the error occurs. In Turbo-Prolog, one needs to compile the source file, which takes time and usually is not easy to debug.

Arity/Prolog is an interpreter in a sense that a program does not require the compilation to work. When a rule is written, it is added to the database (memory), and can be called at any time. It is not a real interpreter like BASIC, because, for example, the source file must be free of any syntax error to be incorporated into the database. Arity/Prolog is better than Turbo-Prolog because of the time savings with the interpreter, and also of the availability of the debug facilities (each rule can be traced independently of the others).

5.5.1 Database Management

5.5.1.1 *Recorded*

The *assert* predicate is used only for Prolog clauses in Arity/Prolog. To manipulate data, we must use the *record* and *recorded* predicates. The *assert* predicate stacks the clauses on the main database. The *record* predicate stacks the data in the named database, and all data are referenced by addresses. The *recorded* predicate increases the database access time, and thus the overall speed of the program. This contributes to the better efficiency of Arity/Prolog.

5.5.1.2 *B-trees*

Another built-in tool of the Arity/Prolog that Turbo-Prolog does not have, is the *B-trees*,

which is used also for database management. A B-tree is a part of the database, having a given name, and where facts or results are stored and **indexed**.

If we use the fact:

recordb(absorb,CS,dseparator_dga(N,A)).

The result *dseparator_dga(N,A)* is stored in the database *absorb* at an address corresponding to *CS*. In the main database, the data are stacked. With *recordb*, data are indexed and stored in specific addresses corresponding to the reference, which is *CS* here.

For example, EXSEP sorts the coefficients of separation (CS) of the feasible separations. Thus it knows the highest one, called H_CS. To recall from the database, the separation Y that has the highest CS, called H_CS (best separation), EXSEP simply uses the clause:

retrieveb(absorb,H_CS,Y).

Turbo-Prolog stacks the results without indexing them, if the expected data is in the middle of the database, Prolog has to browse each datum until it finds the expected one. This process is sometimes less convenient than indexed data and slows down the access time.

5.5.2 "Snips"

Figure 5.11a-b, illustrate the use of the cut, which prevents backtracking (Quantrille and Liu, 1991, pp. 90-93). In Fig. 5.11a, when *sub_goal_4* fails, Prolog backtracks to *sub_goal_3*; if *sub_goal_3* fails Prolog backtracks to *sub_goal_2*, then to *sub_goal_1*. If *sub_goal_1* fails, the all clause *intermediate_goal* fails and Prolog backtracks to *parent_goal*.

With the cut (Fig. 5.11b), when *sub_goal_4* fails, Prolog backtracks, encounters the cut predicate (*!*), the all predicate, *intermediate_goal*, fails, and Prolog backtracks to *parent_goal*.

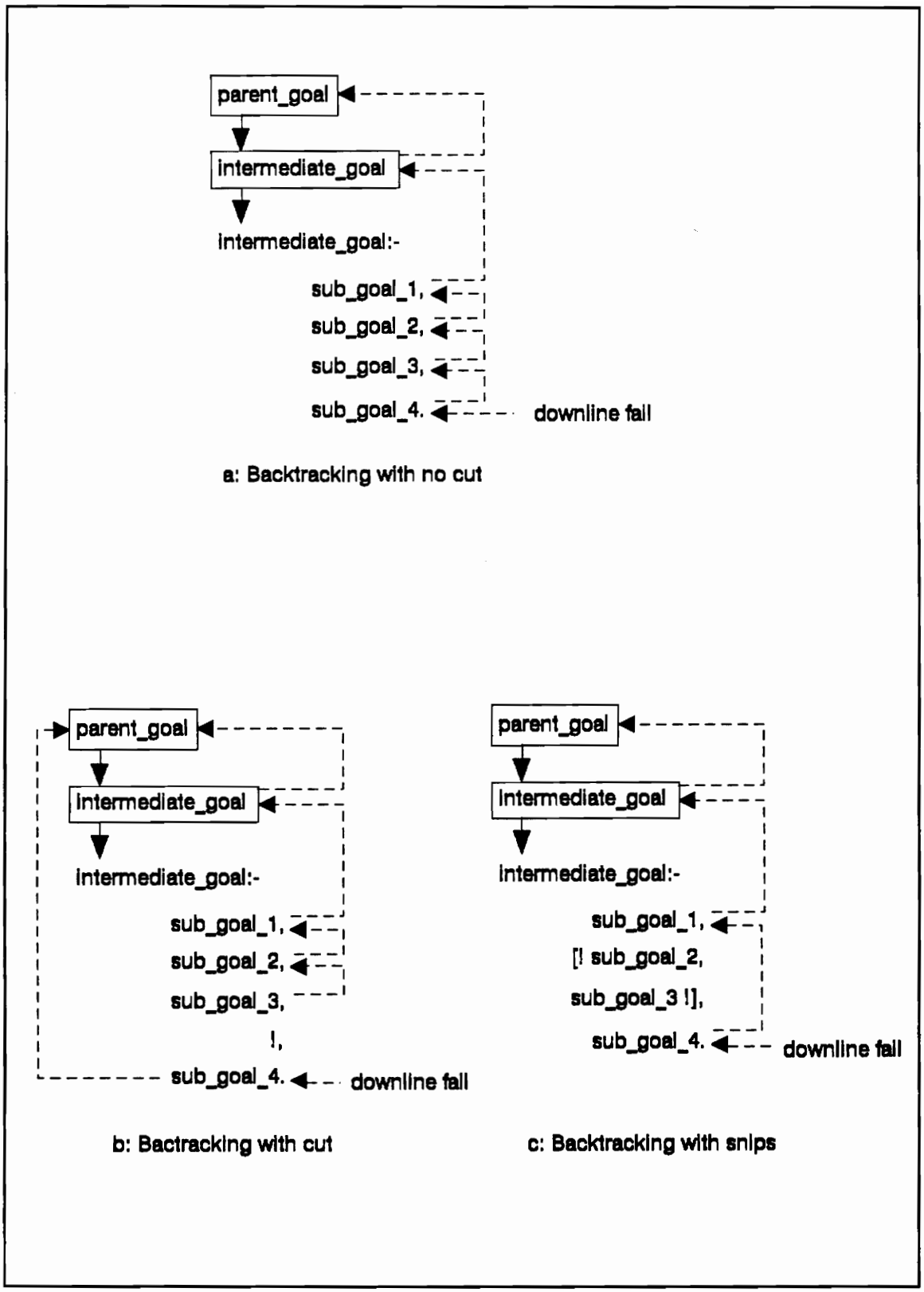


Figure 5.11 : Comparison Between Cut and Snips

The cut makes the entire clause fail without going through the backtracking steps.

If we want to backtrack from *sub_goal_4* to *sub_goal_1*, but skipping *sub_goal_2* and *sub_goal_3*, we need to use the snips (*[!..!]*). With the snips, if *sub_goal_4* fails, backtracking skips over *sub_goals_2 and 3* and proceeds to *sub_goal_1*. We use snips in EXSEP in the incremental procedure: *for*. In this case, the Prolog predicates are:

```
stripping:-      %intermediate_goal
  for(X,Xmax),   %sub_goal_1
[!sub_goals!],   %sub_goal_2 and 3
      fail.      %sub_goal_4
```

In the example, *sub_goal_4* (*fail*) always fails. Prolog backtracks to *for(X,Xmax)*, which succeeds until X reaches the upper limit Xmax (incremental procedure). Without the snips, Prolog backtracks to the *sub_goals*, which leads to an **infinite loop** and blocks the program.

The snips are also useful when retrieving data from the database with the predicate *recorded*. For example, we use in the MSA-modules:

```
check_A_opt(L,V,KConst,NewL):-                    %[1]
        A is L/(V*KConst),                        %[2]
        [!recorded(data,optimal_A(Aopt),_)!],     %[3]
        A<Aopt,                                   %[4]
        NewL is Aopt*KConst*V.                    %[5]
```

Prolog calculates the value of the absorption factor A [2], then Prolog extracts the data, Aopt, from the database [3]. Prolog compares A and Aopt [4], and finally evaluates NewL [5]. Often, the comparison, A < Aopt [4], fails. Without the snips, Prolog would backtrack to [3]. The built-in instruction *recorded* never fails. Thus, [3] is valid, and Prolog goes to [4], which fails again.

Prolog backtracks again to [3], and follows an infinite loop. With the snips, Prolog backtracks from [4] to [2], skipping [3]. The instruction [2] fails and Prolog backtracks to [1], then the whole function *check_Aopt* fails. Thus, we can avoid infinite loops by using the snips.

5.5.3 Menus

The pull-down menus (menus where we can move the cursor to select an item, see Fig. 4.1b) of EXSEP use the Arity/Prolog menu facility. The definition is contained in a separate file called MENU.ARI (see program listing in section A.5). To call a menu in the program, the instruction is simple:

send_menu_msg(activate(yes_no,(10,60)),Decision)

yes_no is the name of this menu, *(10,60)* is its position in the screen and *Decision* is the answer chosen by the user.

Arity\Prolog offers different types of menus. The elements of a menu can have different properties. For example, in the the main menu of EXSEP, some elements appear (e.g.: "Azeotropic Distillation") but cannot be selected ("Azeotropic Distillation" is not implemented yet) (see section 4.1.2). The Arity/Prolog menus can be customized with any colors, shapes and positions.

5.6 Compilation and Linkage Techniques

5.6.1 Main and Restart Predicates.

The main program must start with the predicate *main*. To be able to abort a program with Ctrl-C or Ctrl-Break, we must use the *restart* predicate.

5.6.2 Predicates Declaration

There are several types of declarations. The main ones are the *public* and *extrn* declarations. A predicate is declared "public" in module A when it is defined in module A and used in module B. A predicate is declared "extrn" in module A when it is defined in module B, and used in module A.

The declaration *visible* is used when a predicate is manipulated by the *call or findall* built-in tools. As an example, *input_henrys_law* is defined in the Absorb module, but used also by the Stripping and Extraction modules. It is declared "public" in the absorb module. The predicate *delete_element* is defined in the Utility module, but used in the Absorb module. It is declared "extrn" in the Absorb module.

5.6.3 Far Prolog

The size of a program to be compiled and linked on IBM-PC must be smaller than 64 Kbytes. EXSEP is bigger than this. We use "Far Prolog" (Arity Corporation, 1988, p.149). This "far" declaration enables the compiler to allocate another 64 Kb segment to write the executable code. Each new "far" segment must be declared at the beginning of the declaration part of the module. For example:

:-segment(absseg).

is the declaration for the "far" segment of the absorb module. Each "far" segment must be

processed with the file CLONE.EXE. The instruction is:*c:\> clone absseg [enter]* .

5.6.4 Compiling and Linkage Instruction

To compile the entire program, we use a batch file:

COMP.BAT

```
apc main,,main/n
apc utility,,main
apc bypass,,main
apc sst,,main
apc split,,main
apc absorb,,main
apc strippin,,main
apc lle,,main
```

"apc" is the name of the compiler. We specify the name of the module and the name of the database. The specification, "/n", refers to the case where only one database is used for several modules.

To link, the instruction is: *link @m.lnk* where m.lnk is the following file:

M.LNK

```
code+uiseg+utiseg+sstseg+splitseg+absseg+strseg+
lleseg+main+utility+bypass+sst+
split+absorb+strippin+lle [CR]
exsep [CR]
exsep/map [CR]
arity [CR]
```

code is required for any Arity file, *uiseg* is for the menu facility, and the next 6 names are the names of the "far" segments, followed by the names of the modules. *exsep* is the name of the executable file. *exsep/map* is the .MAP file, and *arity* is the library used by EXSEP. [CR] represents each new line.

5.7 Summary

In this chapter, we described the plan-generate-test search strategy used by EXSEP. We presented the logic structure of the main Prolog clauses which compose the MSA-modules. We briefly compared Arity/Prolog and Turbo Prolog, and summarized the methods used to make compiled versions of Arity/Prolog programs.

CHAPTER 6

Conclusions and Recommendations

- EXSEP is now operating for four separation methods: distillation, absorption, stripping and liquid-liquid extraction. This is *the first expert system* for multicomponent separations using both ESAs (energy-separating agents) and MSAs (mass-separating agents).

- Verifications of EXSEP results are satisfactory, and confirm that EXSEP can be used by the design engineer to generate preliminary separation process flowsheets.

- The development of EXSEP on an IBM-PC makes it "user friendly" and convenient software on the hard drive of every design engineer's PC.

- The use of EXSEP is so easy that it can be used by a non-specialized user, such as chemical engineering undergraduate students.

- The great advantage of EXSEP is to generate several feasible flowsheets in a matter of seconds. The solutions are reliable and can be used as preliminary flowsheets before conducting rigorous simulations with computer-aided design (CAD) software that requires these preliminary flowsheets and much longer processing time.

- The program structure we developed - search strategy and knowledge representation - could be used for other kinds of separations. We could use the *Kremser clause*, explained in section 5.3.2, for any separation for which the Kremser equation is applicable, such as leaching or adsorption (see Table 2.1). Adsorption is used in a large number of new processes. It would be interesting to have a preliminary-design tool such as EXSEP to predict the feasibility of flowsheets including adsorption.

- Another future development of EXSEP would be to write modules for extractive and azeotropic distillations, which combine ESAs and MSAs. These modules could use procedures already written for distillation and for the MSA-based separations modules. The main problem would be to avoid too much algorithmic computation by finding an appropriate shortcut method.

If too much algorithmic computation is necessary, part of the computer code could be written in C language. It appears that Arity/Prolog can easily call C subroutines.

- EXSEP should also contain a solvent-selection module. This domain is very demanding in knowledge. A large number of heuristics are available for solvent selection. The use of Arity\Expert software from Arity Corporation should enable the implementation of a large rule-based expert system for solvent selection. However, solvent selection requires substantial data concerning the mixture. These thermodynamics data are often difficult to find. It makes this kind of expert system difficult to implement on PCs because it should be linked with a large database.

- The development of a separation-selection capability is one of the most difficult tasks of flowsheet synthesis. Such capability would be the "umbrella" of EXSEP, as it would decide which separation module to enter. As in solvent selection, selecting separation methods also

requires substantial data concerning the mixture. It may be difficult to implement this type of expert system on PCs without a large database.

- The ultimate goal of this project would be to distribute EXSEP. Different commercial networks exist to distribute software developed by universities.

- EXSEP could be used as an educational tool to learn
    - Preliminary flowsheet synthesis;
    - Shortcut process design; and
    - Expert-system development.

REFERENCES

API, *API Manual on Disposal of Wastes, Volume of Liquid Wastes*, American Petroleum Institute, Washington D.C., 1967.

API, American Petroleum Institute, *Technical Data Book - Petroleum Refining*, Port City Press, Washington D.C., 1978.

Aspen Technology Inc., *Aspen Plus$_{TM}$ Example Problems*, Aspen Technology Inc., Cambridge, MA, 1986.

Bailes, P. J., C. Hanson and M. A. Hughes, "Liquid-Liquid Extraction: Nonmetallic materials" in *Separation technique 1:L/L Systems*, Chemical Engineering Magazine, McGraw Hill, New York, 1980.

Baird, M. H. I., "Solvent Extraction - The Challenges of a "Mature" Technology", *Can. J. Chem. Eng.*, (69), Dec. 1991.

Blumberg, R., *Liquid-Liquid Extraction*, Academic Press, San Diego, CA., 1988.

Chemshare Corporation, *Design II user's Guide*, Chemshare Corp, Houston Texas, 1985.

Chemshare Corporation, *Design II user's Guide, Supplement: Example of LLE*, Chemshare Corp, Houston Texas, 1986.

Cusack, R. W., P. Fremeaux and D. Glatz, "A fresh look at liquid-liquid extraction, part I", *Chemical Engineering Progress*, February 1991 (a).

Cusack, R. W., P. Fremeaux and D. Glatz, "A fresh look at liquid-liquid extraction, part II", *Chemical Engineering Progress*, March 1991 (b).

Douglas, J. M., *Conceptual Design of Chemical Processes*, McGraw- Hill, New York, 1988.

Edminster, W. C., "Absorption and Stripping Factor Functions for Distillation Calculation by Manual- and Digital-Computer Methods", *AIChe J.*, 3, 165-171 (1957).

Gas Processors Supplier's Association, *Engineering Data Book*, 9th Ed., Gas Processors Supplier's Association, Tulsa, OK, 1972.

Genesereth, M. R. and M. L. Ginsberg, "Logic Programming", *Communications of the ACM*, 28 (9),1985.

Hanson, C., *Recent Advances in Liquid-Liquid Extraction*, Pergamon Press, Oxford England, 1971.

Henley, E.J. and J.D. Seader, *Equilibrium-Stage Separation Operations in Chemical Engineering*, Wiley, New York, 1981.

Holland, C. D., *Fundamentals and Modeling of Separation Processes, Absorption, Distillation, Evaporation, Extraction*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

Jenson, V. G. and G.V. Jeffreys, "Analysis of Continuous Fat-Hydrolysing Column", *Br. Chem. Engng.*, 6, 676, 1961.

Keller, G. E., *Adsorption, Gas Absorption, and Liquid-Liquid Extraction*, M.I.T. press, Cambridge, MA., 1982.

King, C. J., *Separation Processes*, Mc Graw-Hill,New York, 1980.

Kohl, A. L., "Absorption and Stripping", pp.340-404, in *Handbook of Separation Process Technology*, R. W. Rousseau, Ed., Wiley, New York, 1987.

Kremser, A., "Theoretical Analysis of Absorption Process", *Nat. Petroleum News*, 22(21),43-49 (May 21,1930).

Nelson, W. L., *Petroleum Refinery Engineering*, 4th Ed., McGraw-Hill, 1967.

Othmer, D. F., "Random Reminiscences of Problems in Solvent Extraction", pp.xi-xvii, in *Handbook of Solvent Extraction*, Lo t. C., Baird M. H. I., Hanson C., Eds., Wiley, New York, 1983.

Perry, R. H. and D.W. Green, *Perry's Chemical Engineers' Handbook*, 6th ed., McGraw-Hill, New York, 1984.

Pratt, H. R. C., "Computation of Stagewise and Differential Contactors: Plug Flow", pp.151-198, in *Handbook of Solvent Extraction*, Lo t. C., Baird M. H. I., Hanson C., Eds., Wiley, New York, 1983.

Prausnitz, J. M., R. N. Lichtenthaler and E. G. de Azevedo, *Molecular Thermodynamics of Fluid-Phase Equilibria*, Second Edition, Prentice-Hall, Englewood Cliffs, N.J., 1985.

Quantrille, T. E. and Y.A. Liu, *Artificial Intelligence in Chemical Engineering*, Academic Press Inc., San Diego, CA., 1991.

Reissinger, K. H., and J. S. Schroter, "Selection Criteria for Liquid-Liquid Extractors", in *Separation technique 1:L/L Systems*, Chemical Engineering Magazine, McGraw Hill, New York 1980.

Robbins, L. A., "Liquid-Liquid Extraction", pp.1.256-1.281, in *Handbook of Separation*

Techniques for Chemical Engineers, P.A. Schweitzer, Ed., McGraw-Hill, New York, 1979.

Simulation Sciences, Inc, *ProII Example Manual*, Simulation Sciences, Inc., Fulleton, CA, 1987.

Smith, B. D. and W.K. Brinkley, "General Short-Cut Equation for Equilibrium Stage Processes", *AIChE J.*, 6, 446-450 (1960).

Smith, B. D., *Design of Equilibrium-Stage Processes*, McGraw-Hill, New York, 1963.

Treybal, R. E., *Mass-Transfer Operations*, Third edition,McGraw-Hill, New York, 1980.

Van Winkle, M., *Distillation*, McGraw-Hill, New York, 1967.

Wankat, P. C., *Separations in Chemical Engineering : Equilibrium-Staged Separations*, Elsevier Science Publishing, New York, 1988.

Wankat, P. C., *Rate-Controlled Separations*, Elsevier Science Publishing, New York, 1990.

Zenz, F. A., "Design of Gas Absorption Towers", pp.3.50-3.105, in *Handbook of Separation Techniques for Chemical Engineers*, P.A. Schweitzer, Ed., McGraw-Hill, New York, 1979.

APPENDICES

Further Examples and Supplementary Information.

A.1 Further Examples

A.1.1 Absorption

A.1.1.1 Eleven-Component Absorption Problem

a) Problem Statement

A 19-plate refinery absorber operates on cracking oil gas (Jackson and Sherwood, 1941). A lean oil (molecular weight: 285) enters the top of the column at 31.3°C (94.5°F) and 4.96×10^5 Pa (72 psia). A mixture (1841 mol/hr) of H_2S (2.4%), H_2(6.5%), CH_4(35.0%), C_2H_4(3.7%), C_2H_6(20.5%), C_3H_6(7.2%), C_3H_8(13.1%), C_4H_8(3.5%), C_4H_{10}(4.7%), C_5H_{12}(2.6%) and C_6H_{14}(0.8%) enters the bottom of the column at 24°C (80°F) and 4.96×10^5 Pa (72 psia). The column operates at 4.96×10^5 Pa (72 psia).

The plate efficiency is approximately 30 %. Thus, the number of theoretical stages is about 6 stages. The lean oil molar flow rate is: 728.34 mol/hr. The lean oil contains traces (molar fraction) of H_2S (0.06%), C_3H_8 (0.01 %), C_4H_8 (0.1%), C_4H_{10} (0.1%), C_5H_{12} (1%). Figure A.1 shows the data file used for EXSEP.

ABSJACK.DAT

```
henry('H2S',2.8).
henry('H2',18).
henry('CH4',23.5).
henry('C2H4',6.1).
henry('C2H6',4.32).
henry('C3H6',1.33).
henry('C3H8',1.17).
henry('C4H8',0.44).
henry('C4H10',0.33).
henry('C5H12',0.08).
henry('C6H14',0.01).

flow(p1,'H2S',37.79).
flow(p1,'H2',116.62).
flow(p1,'CH4',632.36).
flow(p1,'C2H4',64.82).
flow(p1,'C2H6',344.62).
flow(p1,'C3H6',94.09).
flow(p1,'C3H8',160.85).
flow(p1,'C4H8',12.27).
flow(p1,'C4H10',13.95).
flow(p1,'C5H12',2.34).
flow(p1,'C6H14',0).

flow(p2,'H2S',5.69).
flow(p2,'H2',2.38).
flow(p2,'CH4',12.91).
flow(p2,'C2H4',4.05).
flow(p2,'C2H6',34.36).
flow(p2,'C3H6',36.2).
flow(p2,'C3H8',80.34).
flow(p2,'C4H8',52.19).
flow(p2,'C4H10',72.21).
flow(p2,'C5H12',52.99).
flow(p2,'C6H14',8.34).

mole_fraction(solvent,'H2S',6e-4).
mole_fraction(solvent,'H2',0).
mole_fraction(solvent,'CH4',0).
mole_fraction(solvent,'C2H4',0).
mole_fraction(solvent,'C2H6',0).
mole_fraction(solvent,'C3H6',0).
mole_fraction(solvent,'C3H8',1.4e-4).
mole_fraction(solvent,'C4H8',10.4e-4).
mole_fraction(solvent,'C4H10',10.2e-4).
mole_fraction(solvent,'C5H12',0.01).
mole_fraction(solvent,'C6H14',0).

initial_set([p1,p2]).
initial_components(['H2S','H2','CH4','C2H4','C2H6','
C3H6',
        'C3H8','C4H8','C4H10','C5H12','C6H14']).
solvent(oil).

key_component(dga,'C4H8'). % dga = dilute gas
absorption
split_product(p1).

optimal_A(1.4).
minimum_A(0.85).
limit_A(0.87).
step(0.005).
optimal_N(5).
error(0.4).
```

Figure A.1: Data File for Absorption Problem (Jackson and Sherwood, 1941).

b) Results

In Table A.1, we show the design variables found by EXSEP.

Table A.1: Comparison of Design Variables

Design Variables	EXSEP	Jackson & Sherwood
N	5.02	6.33
A	0.875	1.88
L (mol/hr)	709	738

The number of stages predicted by EXSEP (5.02) is close and consistent with the value Jackson and Sherwood give (6.33). The lean-oil flow rate provided by EXSEP (709 mol/hr) is also satisfactory compared to the amount fed to the industrial column (728 mol/hr). The only concern with the results from EXSEP is the low value of the absorption factor A. This value of 0.85 is far from the optimum of 1.4.

For higher values of the absorption factor, EXSEP finds solutions with a very low number of stages ($\approx$ 1 stage). Jackson and Sherwood find that an absorption factor (1.88) is close to the optimum because they calculate the molal overflow (L/V) at several points of the column. The assumption of a constant molal overflow, used in EXSEP, is not valid. Rigorous simulation shows that this overflow varies from 0.52 to 0.62.

Figure A.2 compares the component-recovery ratios, sorted in decreasing K-value order, for the results from EXSEP (first series), the results from Design II (second series) and those of the experiments ran on an actual absorber by Jackson and Sherwood (third series). A cut appears between C_3H_8 and C_4H_8. C_4H_8 is the key component.

EXSEP overestimates the absorption of the components lighter than the key component.

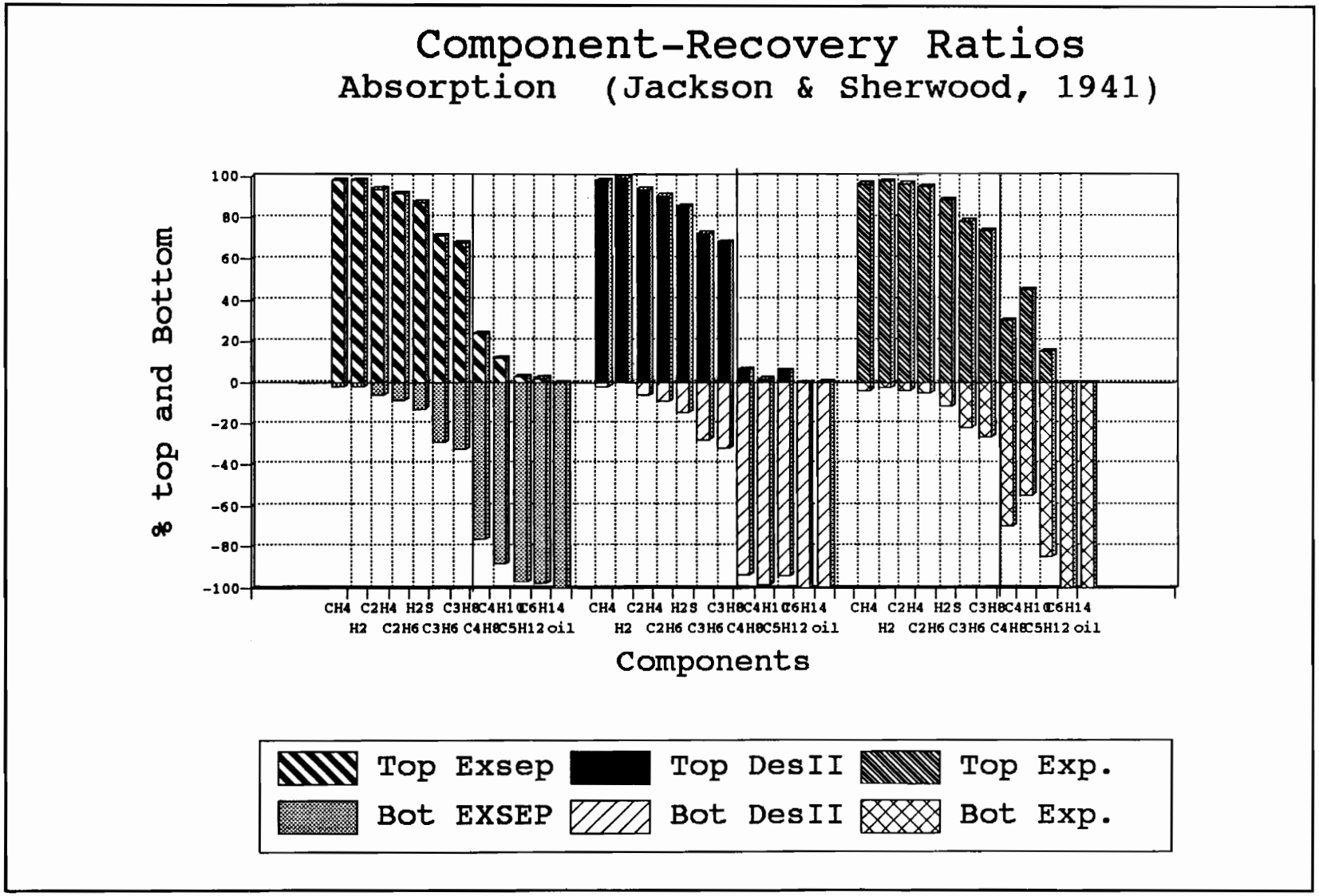


Figure A.2: Component-Recovery Ratios. Example of Absorption (Jackson and Sherwood, 1941).

If we compare the results with rigorous simulations (second series), EXSEP underestimates the absorption of the components lighter than the key. However, rigorous simulations by Design II overestimate the absorption of light components. This comes from the ideal thermodynamic model chosen to evaluate the K values in Design II.

To summarize, we can say that:

- EXSEP gives good preliminary design variables:
    - Number of stages
    - Solvent flow rate
- EXSEP gives satisfactory component-recovery ratios for the key component, but it:
    - underestimates the absorption of heavy components;
    - overestimates the absorption of light components.
- EXSEP does not provide a "good" absorption factor such as A=1.4 (heuristic optimum):
    - if we search only for a "good" separation factor, we would not get any solution for this example. However, we find one because in searching for a good flowsheet solution, we pay attention to both the number of stages and the absorption factor even if they are not independent of each other (see section 3.2.2).

A.1.2 Stripping

A.1.2.1 Sour-Water Stripper

a) Problem Statement

We illustrate how EXSEP handles the separations when the solvent is also a major component of the feed. The case of sour-water stripper (Chemshare, 1985, p.10) is very common. Usually, the problem is to strip pollutants (H_2S and NH_3, here) contained in water (sour or waste water), with steam.

Solving this example with EXSEP leads to several difficulties:

- In EXSEP, the K value for the water is infinite and the liquid feed is free of the lean-gas components. For example, if we use steam as the stripping gas, we should not have water in the liquid feed.
- The shortcut method assumes that L and V (the liquid and vapor flow rates) are constant throughout the column. This is not a valid assumption here.

We decided to use the Stripping module as it is, without any modification for this kind of problem.

We specify the expected flow rates in the data file. Table A.2 shows the specifications for a sharp split. These specifications are the results from rigorous simulation. But, with these specifications, EXSEP does not give any answer.

Table A.2: Specifications, Example of Stripping

Components	Top Feed (kg/hr)	Specified Top Product (kg/hr)	Specified Bot. Product (kg/hr)
H2S	89.6	87.8	1.8 (2%)
NH3	149.6	146.6	3 (2%)
Water	74500	1490	73010 (98%)

EXSEP is not able to evaluate the feasibility of the sharp split. It cannot predict that 98% of the water goes to the bottom of the column (Table A.2). Instead, a limit in the specifications appears where EXSEP starts giving solutions (Table A.3). Figure A.3 shows the datafile used.

Table A.3: Limit in The Specifications.

Components	Top Feed (kg/hr)	Specified Top Product (kg/hr)	Specified Bot. Product (kg/hr)
H2S	89.6	87.8	1.8 (2%)
NH3	149.6	146.6	3 (2%)
Water	74500	8616.7	658083 (88%)

b) Results

For this database, the best solution (among a dozen) that EXSEP gives, is:

Table A.4: Best Solution

Comp.	Spec. Top Product kg/hr	Spec. Bot. Product kg/hr	Solution Top Product kg/hr	Solution Bot. Product kg/hr
H2S	87.8	1.8	87.8	1.8
NH3	146.6	3	146.6	3
Water	8616.7	658083 (88%)	10057	64443 (86%)

SOURCHEM.DAT

```
henry('H2S',11.75).
henry('NH3',9.68).
henry('H2O',0.87).

flow(p1,'H2S',1.79).
flow(p1,'NH3',2.99).
flow(p1,'H2O',65883.33).

flow(p2,'H2S',87.79).
flow(p2,'NH3',146.59).
flow(p2,'H2O',8616.67).

mole_fraction(lean_gas,'H2S',0).
mole_fraction(lean_gas,'NH3',0).
mole_fraction(lean_gas,'H2O',0).

initial_set([p1,p2]).
initial_components(['H2S','NH3','H2O']).

lean_gas(steam).
key_component(s,'NH3'). % s = stripping
split_product(p2).


optimal_S(1.5).
minimum_S(0.5).
limit_S(2.2).
step(0.05).
optimal_N(5).
error(0.2).
```

Figure A.3 : Database for the Sour-Water Stripping Example (Chemshare, 1985).

With the following design variables, we compare with EXSEP those used in the Design II rigorous simulation by Chemshare:

Table A.5: Design Variable Comparison

	EXSEP		Chemshare (1985)
	Best	Other	
N	7.2	8.3	8
V	10809	10423	11800
S	1.4	1.35	1.53

The results are satisfactory in terms of number of stages, and steam flow rate.

- **Real Design Procedure**

The real design procedure would be:

- Find the K values for the components.

- Specify the flow rates of Table A.2, and run a shortcut simulation with EXSEP, which would not give any results.

- Keep running shortcut simulations, but decrease the specified proportion of water going to the bottom of the column each time no solution is found. When this proportion reaches 88% (in this example) the simulation gives the results of Table A.4 and A.5.

- Run a rigorous simulation with the parameters of Table A.5, but expect the flow rates to be very different from the product flow rates of the shortcut method. For example, in Design II, the product constraint would be PRO=370,0 and not PRO=8851,0.

- The design variables used for rigorous simulation are:

Table A.6: Design Variables From EXSEP

	Design Variables from EXSEP for the rigorous simulation with DesignII.
N	7.2
V	10809 (kg/hr)

In Table A.7, we compare the flow rates and percentages of stripped components for three methods. The first two columns are the results from EXSEP, which correspond to the design variables presented in Table A.6. With these design variables, we ran two kinds of rigorous simulation on Design II. The first stripper is with a partial condenser, the other is a total stripper (no reflux).

Table A.7: Product Flow Rate Comparison for Three Methods.

Comp.	**EXSEP** Top Product kg/hr	**EXSEP** Bot. Product kg/hr	**DES.II** Partial St. Top Product kg/hr	**DES.II** Partial St. Top Product kg/hr	**DES.II** Total St. Top Product kg/hr	**DES.II** Total St. Bottom Product kg/hr
H2S %	87.8 *98%*	1.8	83 *92.6%*	6.6	67.98 *75.8%*	21.6
NH3 %	146.6 *98%*	3	120.3 *80.4%*	29.3	58.52 *39.1%*	91.08
Water + Steam Total %	10057 + 11196 21253	64443 + 0 64443 *75 %*	0 + 165.8 165.8	74500 + 10643.2 85143.2 *99.8%*	0 + 239.05 239.05	74500 + 10570 85070 *99.7%*

Figure A.4 shows the corresponding component-recovery ratios. We observe that the results for the two components, H_2S and NH_3, predicted by EXSEP are closer to the results for the partial stripper than those for the total stripper. However, the prediction for the separation of the water is less accurate with EXSEP. Indeed, EXSEP predicts a top product more dilute than

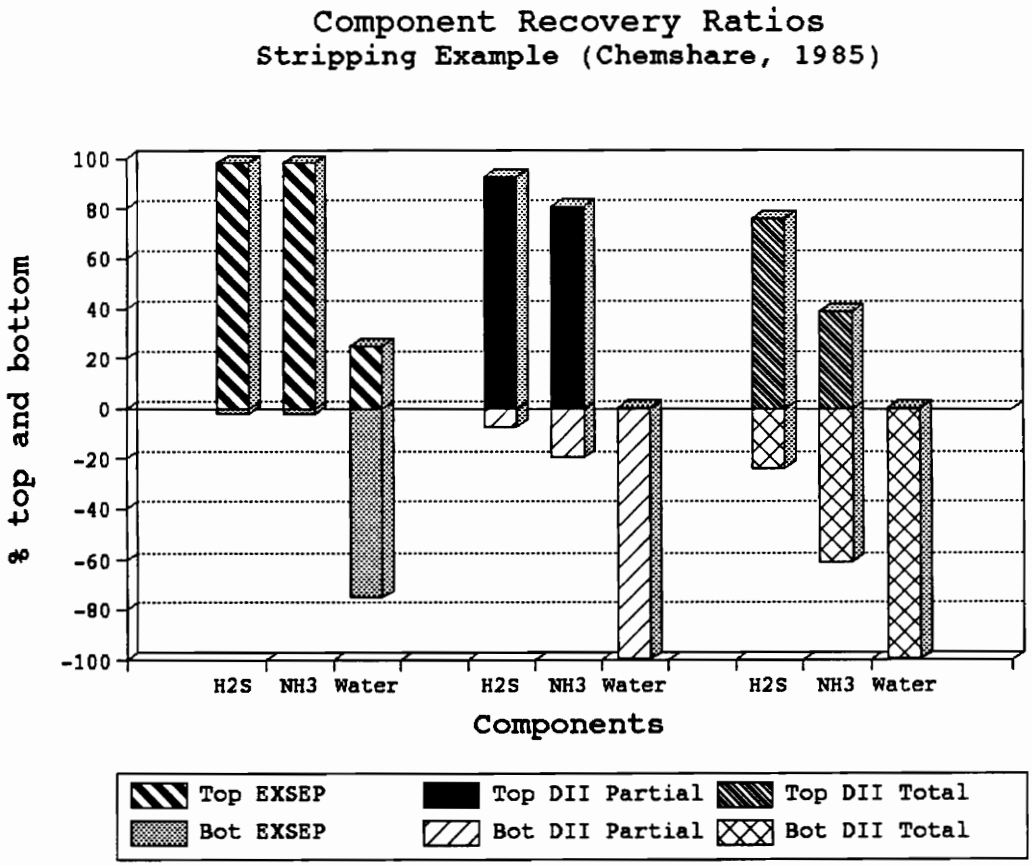


Figure A.4 : Sour-Water Stripping Example. (Chemshare, 1985).

the feed (0.4 % H2S), whereas it should be much purer, like the rigorous solution (22 % of H2S) !

- **To summarize**:

  - EXSEP gives a very good evaluation of the optimal number of stages and a fairly good evaluation of the lean-gas flow rate. They are reliable preliminary design variables.

  - The amounts of the two stripped products is overestimated by EXSEP. But the approximate values are acceptable.

  - The input data representing the separation of the water (also lean gas) must be underestimated for EXSEP to give solutions (Table A.3).

In conclusion, EXSEP is not designed to handle problems where the solvent is a major component. It can be used and may give useful results, but these results may be inaccurate.

A.1.3 Extraction

A.1.3.1 Example 1: DMA and DMF removal

a) Problem Statement

This example (Henley and Seader, 1981, p.479) consists of the extraction of dimethylamine (DMA) and dimethylformamide (DMF) present in an aqueous solution (0.5% and 10%, respectively) containing also 0.5% of formic acid (FA). The solvent is 99.73% methylene chloride (MC), plus **traces of DMF and water** (0.02% and 0.25%, respectively). We did not have to modify the problem, **as EXSEP can handle components in the solvent**.

For the K-value calculation, Henley and Seader give the distribution coefficients for MC,FA,DMA and W (water). For DMF, they consider that the distribution coefficient of DMF depends on the concentration in the water-rich phase, whereas this dependence is negligible for the other components (constant distribution coefficients). EXSEP can accept **in its database only constant K-values**. For this reason, we use an average value for the K-value of DMF. Figure A.5 shows the database used with EXSEP for this problem.

b)Results

Henley and Seader use a shortcut method (Edmister equation) similar to the Kremser equation used by EXSEP.

Table A.8 compares the results obtained by the two methods. The results from EXSEP are different but not false. Indeed, the heuristic approach of EXSEP may give a better solution than the algorithmic approach used by Henley and Seader. Henley and Seader start with the assumption of 10 stages, which comes from their knowledge, or from actual industrial design. They end up with a smaller solvent flow rate. EXSEP 'knows' from heuristics on the number

LLESEAD.DAT

```
henry('FA',0.005).
henry('DMA',2.2).
henry('DMF',0.8).  %approx avg
henry('H2O',0.003).

flow(ra,'FA',20).
flow(ra,'DMA',0).
flow(ra,'DMF',2).
flow(ra,'H2O',3560).

flow(ex,'FA',0).
flow(ex,'DMA',20).
flow(ex,'DMF',400).
flow(ex,'H2O',25).

mole_fraction(solvent,'FA',0).
mole_fraction(solvent,'DMA',0).
mole_fraction(solvent,'DMF',0.0002).
mole_fraction(solvent,'H2O',0.0025).

initial_set([ra,ex]).
initial_components(['FA','DMA','DMF','H2O']).
solvent(mc).
key_component(lle,'DMF'). % lle = liquid liquid extraction
split_product(ex).



optimal_E(0.8).
minimum_E(0.5).
limit_E(4).
step(0.2).
optimal_N(5).
error(0.1).
```

Figure A.5 : Datafile for the Extraction Problem 1 (Henley and Seader, 1981).

of stages that the optimum N is 5 so its knowledge base leads to a solution with 5 stages which increases the solvent flow rate. Both solutions are valid, and a precise economic calculation would decide the best one.

Table A.8: Design Variables

	Group Method (Henley and Seader, 1981)	**EXSEP**
N	10	5
V (kg/hr)	9973	14094.5
E	2.06	2.8

Figure A.6 and Table A.9 compares the results in terms of flow rates and component-recovery ratios.

Table A.9: Flow Rate Comparison

Comp.	**EXSEP** *Extract* Top Product kg/hr	**EXSEP** *Raffinate* Bot. Product kg/hr	**Group Met.** *Extract* Top Product kg/hr	**Group Met.** *Raffinate* Bot. Product kg/hr
MC (solvent)	14094.5	0	9882.2	90.8
DMA	19.55	0.45	20	0
DMF	393.96	8.04	400.7	1.3
FA	0.4	19.6	0.3	19.7
W	71.7	3513.3	37.8	3557.2

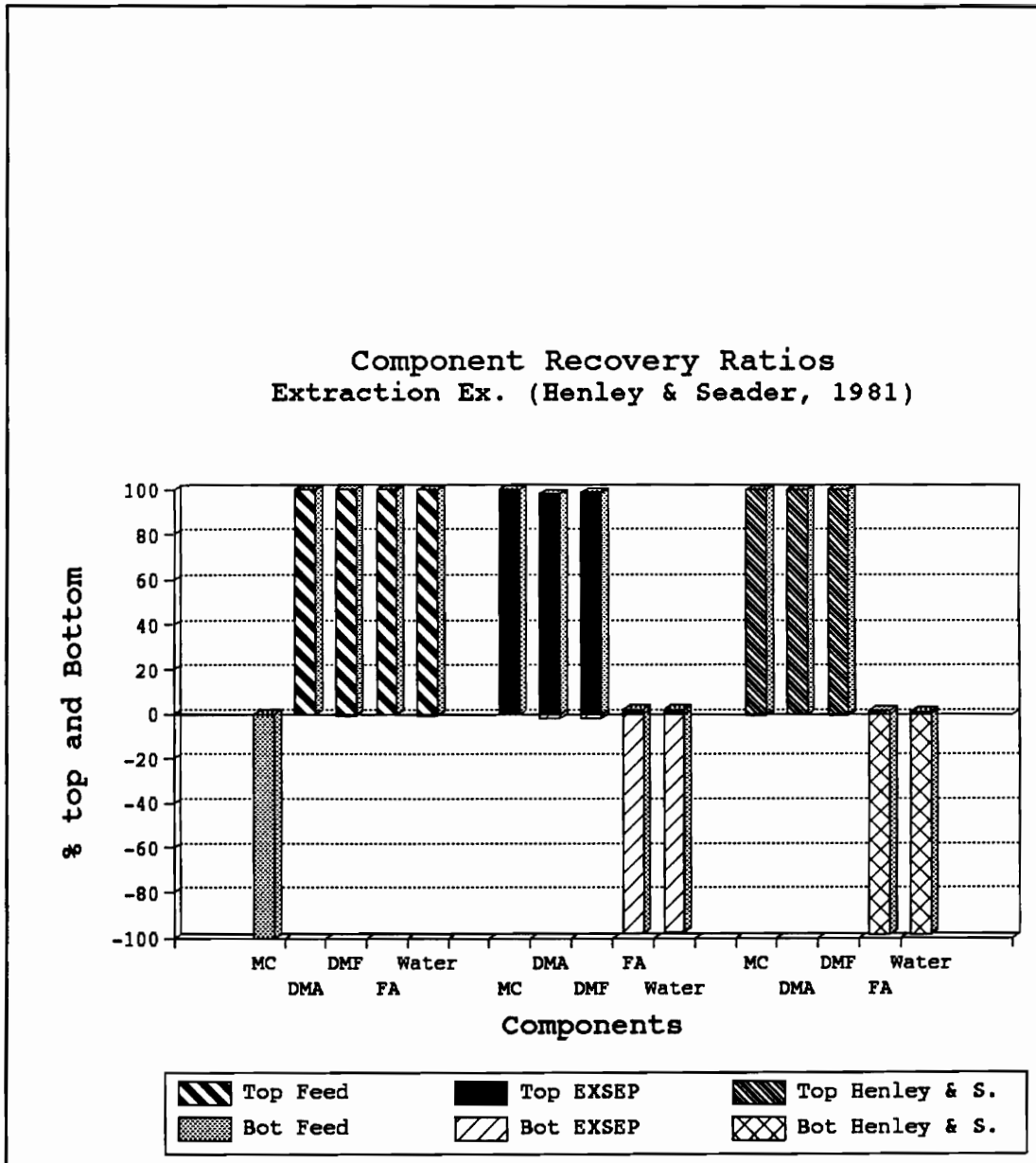


Figure A.6 : Comparison of Component-Recovery Ratios for the Extraction Example (Henley and Seader, 1981).

A.1.3.2 Example 2: Extraction of Acetone

a) Problem Statement

This example (Chemshare, 1986) consists of the extraction of acetone (23.6%) from an aqueous solution (76.4 % water). The solvent is 100 % trichloroethane. We used the database shown in Fig. A.7.

b) Results

It turned out that we could not find any solution.

After some reflection, we remembered Table 2.1 (*"The Table Which Should Never Be Forgotten"*) which says explicitly that the Kremser equation IS applicable ONLY when the solution is very dilute. This is obviously not the case with 23 % of acetone in water.

We have just shown that EXSEP is limited by the applicability of the Kremser equation.

LLECHEM.DAT

```
henry('Ace',4.38).
henry('H2O',0.14).

flow(p1,'Ace',0.72e-1).
flow(p1,'H2O',2.483).

flow(p2,'Ace',0.789).
flow(p2,'H2O',0.2923).

mole_fraction(solvent,'Ace',0).
mole_fraction(solvent,'H2O',0).

initial_set([p1,p2]).
initial_components(['Ace','H2O']).

solvent(tc).
key_component(lle,'Ace'). % lle = liquid liquid extraction

split_product(p2).
optimal_E(0.8).
minimum_E(0.7).
limit_E(3).
step(0.025).
optimal_N(4).
error(0.1).
```

Figure A.7 : Datafile for Extraction Problem 2 (Chemshare, 1986).

A.2 List of K-Value Sources

Engineering Data Book, (Gas Processors Supplier's Association, 1972, pp.18- 1-112).

Petroleum Refinery Engineering (Nelson, 1967, p.466).

Perry's Handbook (Perry and Green, 1986, p.13-19).

Technical Data Book - Petroleum Refining, (API, 1978, Chapter 8).

A.3 Derivation of the "Coefficient of Separation" (CS).

We want to derive a factor CS(Dev,N,S), which has the following characteristics:

$$\left(\frac{\partial CS(Dev,N,S)}{\partial N}\right)_{N=Nopt}=0 \quad and \quad CS(N+1)>CS(N-1)$$
$$\left(\frac{\partial CS(Dev,N,S)}{\partial S}\right)_{S=Sopt}=0 \quad and \quad CS(S+1)<CS(S-1) \qquad \text{(A.1)}$$

$$if\ Dev\neq 0$$
$$\left(\frac{\partial CS(Dev,N,S)}{\partial Dev}\right)<0$$

where: **Dev** is the average deviation between the actual and expected component-recovery ratios; **N** is the number of stages; $\mathbf{N_{opt}}$ is the optimal number of stages; **S** is the stripping factor (KV/L); $\mathbf{S_{opt}}$ is the optimal stripping factor.

We develop 3 functions. Each of them contributes to the expected characteristics such that:

$$CS(Dev,N,S)=f_1(Dev)+f_2(N)+f_3(S) \qquad \text{(A.2)}$$

$$\frac{\partial CS(Dev,N,S)}{\partial Dev} = \frac{df_1(Dev)}{dDev}$$
$$\frac{\partial CS(Dev,N,S)}{\partial N} = \frac{df_2(N)}{dN} \qquad \text{(A.3)}$$
$$\frac{\partial CS(Dev,N,S)}{\partial S} = \frac{df_3(S)}{dS}$$

Let us find equation f_1(Dev). We want CS to increase when Dev decreases. We choose f_1 such that:

$$f_1(Dev)=\frac{1}{Dev}$$

The expected properties for N and S lead to the similar functions f_2 and f_3. Let us consider the function f_2 which must have a maximum in Nopt. A parabola is the result of the integration of this property.

$$\left(\frac{df_2(N)}{dN}\right)_{N=Nopt} = 0 \rightarrow f_2(N) = \alpha N^2 + \beta N + \gamma \qquad \text{(A.5)}$$

$$2\,\alpha\, Nopt + \beta = 0\,; \quad \textit{let us choose } \alpha = \frac{1}{2} \textit{ then } \beta = -N_{opt} \qquad \text{(A.6)}$$

$$f_2(N) = \frac{N^2}{2} - N_{opt}\, N + \gamma \qquad \text{(A.7)}$$

This function is symmetric, however we want CS(N+1)>CS(N-1), with still a maximum in N_{opt}. We want the slope of f_2 to be higher in N-1 than in N+1. We choose to incorporate into f_2 the function arc-tangent which has a sharp slope around 0 and then tends to an asymptote. To meet the specifications of the derivative, we take the logarithm of the parabola. Thus function f_2 (curve no. 3 on the graph fig. A.11) is cut in two functions f_{21} and f_{22} defined by the following equations in their general form:

$$\begin{aligned} f_{21}(N) &= -\ln\{\, a N^2 + b N + c \,\} \\ f_{22}(N) &= d \arctan\left\{ \frac{N+e}{f} \right\} \end{aligned} \qquad \text{(A.8)}$$

and:

$$f_2(N) = f_{21}(N) + f_{22}(N) \qquad \text{(A.9)}$$

we need to find the parameters a to f. If we take the derivative of equation A.13:

no unique solution exists. We rearrange the equation and use the condition which states that the

$$\frac{df_2}{dN} = -\frac{2\,a\,N + b}{a\,N^2 + b\,N + c} + \frac{d}{f}\,\frac{1}{\left(\frac{N+e}{f}\right)^2 + 1} \qquad \text{(A.10)}$$

derivative is zero in $N = N_{opt}$.

$$-\frac{2\,N_{opt} + \frac{b}{a}}{N_{opt}^2 + \frac{b}{a}\,N_{opt} + \frac{c}{a}} + \frac{d\,f}{N_{opt}^2 + 2\,N_{opt}\,e + e^2 + f^2} = 0 \qquad \text{(A.11)}$$

There is no unique solution to this equation. We try to equalize the numerator and denominators. Which leads to the following system.

$$\begin{aligned} 2\,N_{opt} + \frac{b}{a} &= d\,f \\ \frac{b}{a} &= 2\,e \\ \frac{c}{a} &= e^2 + f^2 \end{aligned} \qquad \text{(A.12)}$$

One solution to this system is.

$$\begin{aligned} a &= \frac{1}{2} \\ b &= -(N_{opt} - 2) \\ c &= \frac{(N_{opt}-2)^2}{2} + 1 \\ d &= 2\sqrt{2} \\ e &= b \\ f &= \sqrt{2} \end{aligned} \qquad \text{(A.13)}$$

which finally leads to the equation for f_2:

$$f_2(N) = -\ln\left\{\frac{N^2}{2} - (N_{opt}-2)N + \frac{(N_{opt}-2)^2}{2} + 1\right\} + 2\sqrt{2}\arctan\left\{\frac{N-(N_{opt}-2)}{\sqrt{2}}\right\} \qquad \text{(A.14)}$$

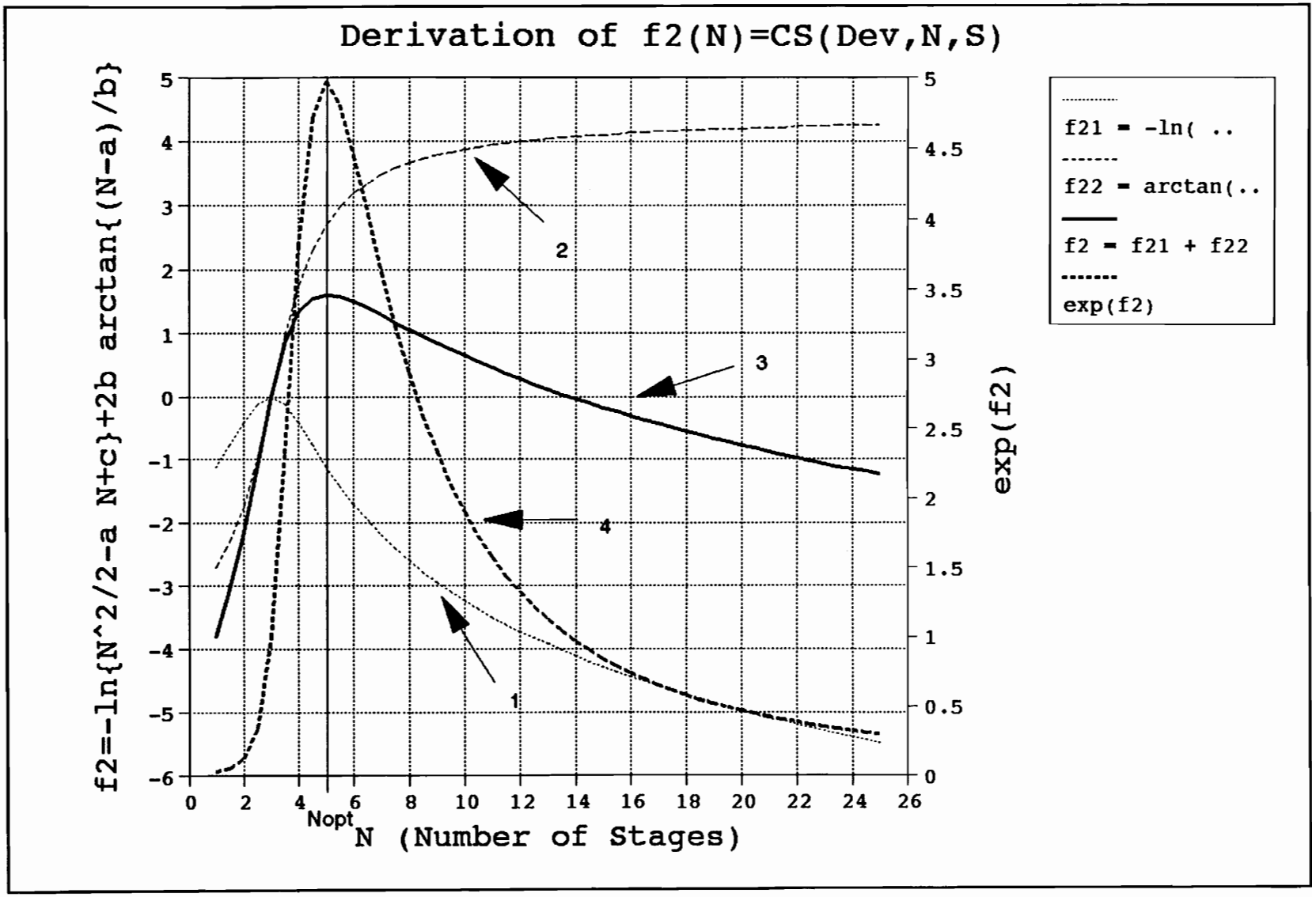


Figure A.8 : Contribution of N to the CS Factor.

where we use the exponential, to rescale the equation. One Figure A.11, curve #3 is the CS factor without the exponential, and curve #4 is the CS factor with the exponential. We observe the maximum in $N_{opt}=5$ and the fact that CS(4) is smaller than CS(5).

For the parameter S the equation is almost the same:

$$f_3(S)=-\ln\{\frac{S^2}{2}-(S_{opt}+2)S+\frac{(S_{opt}+2)^2}{2}+1\}-2\sqrt{2}\arctan\{\frac{S-(S_{opt}+2)}{\sqrt{2}}\} \quad \textbf{(A.15)}$$

Thus the final CS equation is:

$$\begin{aligned} CS(Dev,N,S)=&\exp[-\ln(Dev) \\ &-\ln\{\frac{N^2}{2}-(N_{opt}-2)N+\frac{(N_{opt}-2)^2}{2}+1\}+2\sqrt{2}\arctan\{\frac{N-(N_{opt}-2)}{\sqrt{2}}\} \\ &-\ln\{\frac{S^2}{2}-(S_{opt}+2)S+\frac{(S_{opt}+2)^2}{2}+1\}-2\sqrt{2}\arctan\{\frac{S-(S_{opt}+2)}{\sqrt{2}}\}\ ] \end{aligned} \quad \textbf{(A.16)}$$

A.4 Program Flowchart References

Table A.10 : Prolog Functors Corresponding
to the indices of Fig. 5.3 (see next page)

1	calc_number_stage
2	calc_number_stage
3	check_every
4	check_X_opt
4.1	check_X_opt
4.2	components_within_spec
5	components_within_spec in check_in_spec
5.1	diag
5.2	store_sep
5.3	fail in kremser
6	components_within_spec
6.1	check_X_opt
6.2	components_within_spec
7	components_within_spec in check_in_spec
7.1	diag
7.2	store_sep
7.3	fail in kremser
8	components_within_spec
9	check_X_opt
9.1	components_within_spec
10	components_within_spec
11	components_within_spec in check_in_spec
11.1	diag
11.2	store_sep
11.3	fail in kremser
12	check_every
12.1	store_Xst
12.2	components_within_spec
13	components_within_spec
14	components_within_spec in check_in_spec
14.1	diag
14.2	store_sep
14.3	fail in kremser

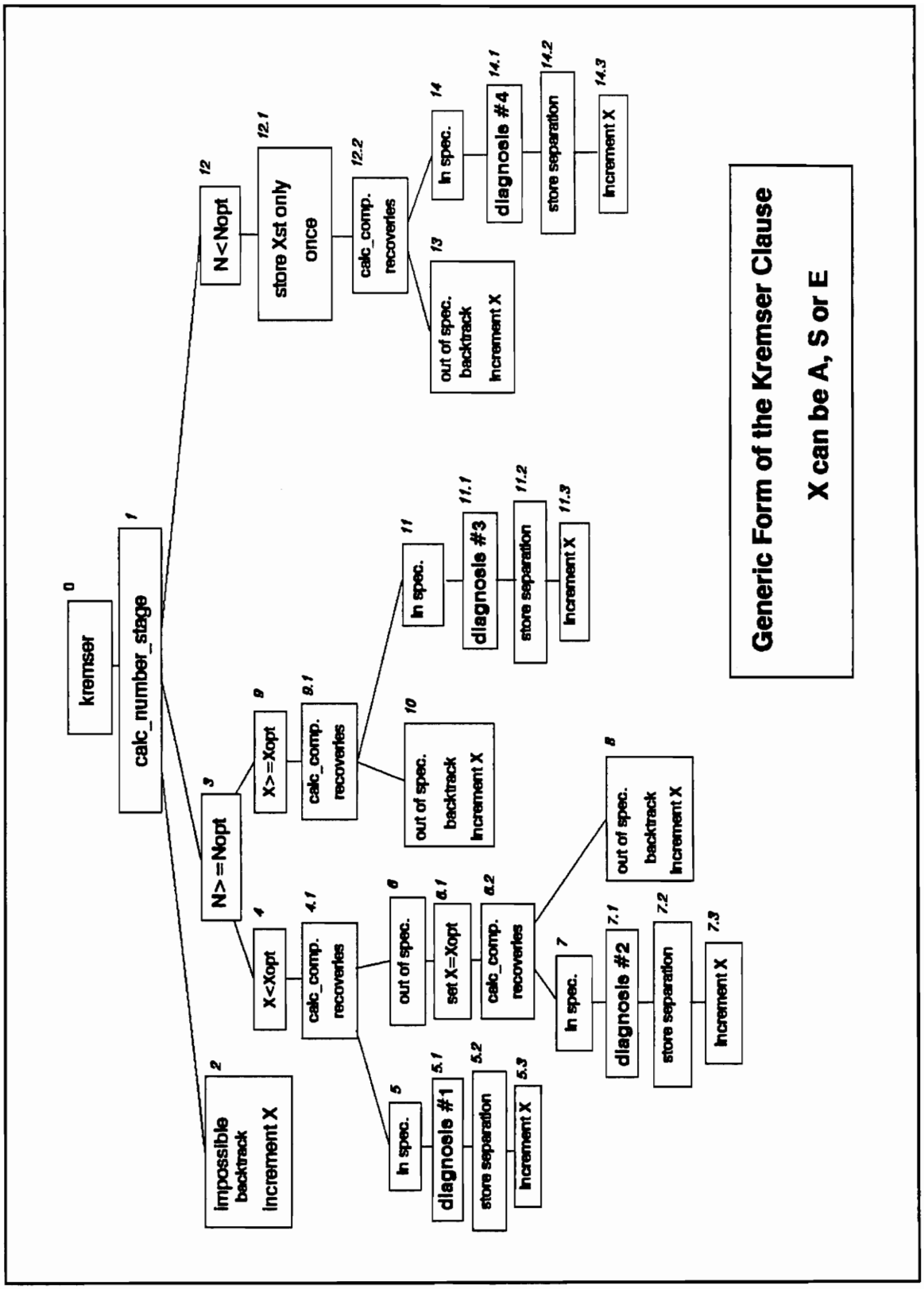


Figure A.9 : Reproduction of Figure 5.3.

Table 8.10: Typical Applications of Absorption for Gas Purification (Rousseau, 1984, p.341).

Impurity	Process	Absorbent
Ammonia	Indirect process (for coke oven gas)	Water
Carbon dioxide and hydrogen sulfide	Ethanolamine	Mono- or diethanolamine in water
Carbon dioxide and hydrogen sulfide	Water wash	Water (at elevated pressure)
Carbon dioxide and hydrogen sulfide	Estasolvan	Tributylphosphate
Carbon dioxide and hydrogen sulfide	Benfield	Potassium carbonate and activator in water
Carbon dioxide and hydrogen sulfide	Fluor solvent	Propylene carbonate
Carbon dioxide and hydrogen sulfide	Giammarco-vetrocoke	Potassium arsenite in water
Carbon dioxide and hydrogen sulfide	Purisol	N-Methyl-2 pyrrolidone
Carbon dioxide and hydrogen sulfide	Rectisol	Methanol
Carbon dioxide and hydrogen sulfide	Selexol	Dimethyl ether of polyethylene glycol
Carbon dioxide and hydrogen sulfide	Sulfinol	Diisopropanolamine, sulfolane, and water
Carbon monoxide	Copper ammonium salt	Cuprous ammonium carbonate and formate in water
Hydrogen chloride	Water wash	Water
Hydrogen fluoride	Water wash	Water
Hydrogen sulfide	Ferrox	Ferric hydroxide, sodium carbonate, and water
Hydrogen sulfide	Stretford	Sodium carbonate, sodium vanadate, and anthraquinone disulfonic acid
Hydrogen sulfide	Thylox	Sodium thioarsenate in water
Naphthalene	Oil wash	Oil
Sulfur dioxide	Aqueous carbonate process (ACP)	Sodium carbonate solution (in spray dryer)
Sulfur dioxide	ASARCO	Dimethylaniline
Sulfur dioxide	Double alkali process	Sodium sulfite
Sulfur dioxide	Wellman-Lord	Sodium sulfite
Sulfur dioxide	Lime/limestone scrubbing	Slurry of lime or limestone in water
Sulfur dioxide	Dry lime	Lime slurry (in spray dryer)
Water	Glycol dehydration	Di- or triethylene glycol
Water	Kathabar	Lithium chloride in water

Table 8.11: Typical Applications of Absorption for Product Recovery (Rousseau, 1984, p.342).

Product	Process	Absorbent
Acetylene	High Temperature Cracking of Hydrocarbons	Dimethylformamide
Acrylonitrile	Sohio process	Water
Ammonia	Phosam process	Ammonium phosphate in water
Butadiene	Houdry process	Oil
Carbon dioxide	Recovery from flue gas	Monoethanolamine in water
Ethanol	Hydration of ethylene	Water
Formaldehyde	Partial oxidation	Water
Hydrochloric acid	Synthesis from hydrogen and chlorine	Water
Light hydrocarbons	Oil absorption	Oil
Sodium carbonate	Solvay process (CO_2 absorption)	Ammoniated brine
Sulfur dioxide	Recovery from smelter gas	Dimethylaniline
Sulfuric acid	Contact process (SO_3 absorption)	Sulfuric acid
Urea	Synthesis (CO_2 and NH_3 absorption)	Ammonium carbamate solution

Table 8.12: Selection Guide for Absorbers and Strippers. (Rousseau, 1984, p.343).

	Tray Columns		Packed Columns (Random)	Spray Contactors
	Perforated	Bubble Cap		
Low liquid rate	D	A	C	D
Medium liquid rate	A	C	B	C
High liquid rate	B	C	A	A
Difficult separation (many stages)	A	B	A	D
Easy separation (one stage)	C	C	B	A
Foaming system	B	C	A	C
Corrosive fluids	B	C	A	A
Solids present	B	D	C	A
Low ΔP	C	D	B	A
High turndown ratio	C	A	B	D
Versatility (can be changed)	C	C	A	D
Multiple feed & drawoff points	B	A	C	C

Key: A, best selection; B, usually suitable; C, evaluate before specifying; D, generally not applicable.

8.6 Extraction References

8.6.1 Classification of equipment for LLE

Table 8.13: Classification of Equipment for Liquid-Liquid Extraction (Reiddinger and Schroter, 1980).

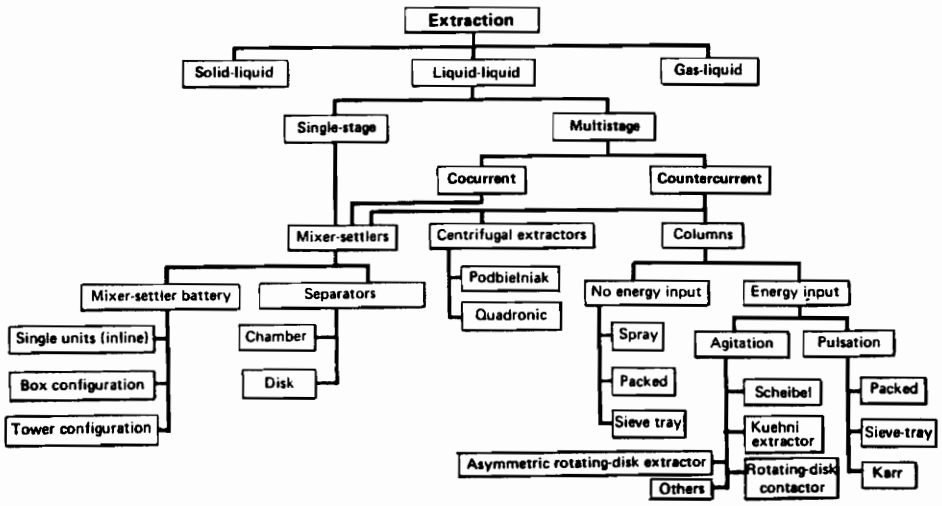
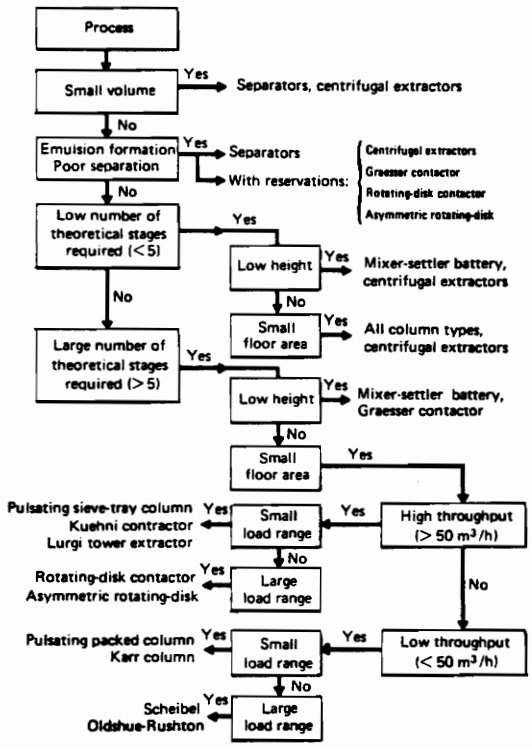


Table 8.14: Criteria for Preliminary Selection of Liquid-Liquid Extractors (Reiddinger and Schroter, 1980).



8.6.2 Solvent Selection

Table 8.15: Group Interactions Can Narrow the Search for a Solvent (Cusack, 1991b).

Group	Solute	Solvent 1	2	3	4	5	6	7	8	9
1	Acid, aromatic OH (phenol)	0	-	-	-	-	0	+	+	+
2	Paraffinic OH (alcohol), water, imide or amide with active H	-	0	+	+	+	+	+	+	+
3	Ketone, aromatic nitrate, tertiary amine, pyridine, sulfone, trialkyl phosphate, or phosphine oxide	-	+	0	+	+	-	0	+	+
4	Ester, aldehyde, carbonate, phosphate, nitrite, or nitrate, amide without active H; intramolecular bonding, e.g, *o*-nitro-phenol	-	+	+	0	+	-	+	+	+
5	Ether, oxide, sulfide, sulfoxide, primary and secondary amine or imine	-	+	+	+	0	-	0	+	+
6	Multihalo-paraffin with active H	0	+	-	-	-	0	0	+	0
7	Aromatic, halogenated aromatic, olefin	+	+	0	+	0	0	0	0	0
8	Paraffin	+	+	+	+	+	+	0	0	0
9	Monohalo-paraffin or olefin	+	+	+	+	+	0	0	+	0

Note: A plus-sign means that compounds in the group represented by the column tend to raise activity with respect to the compounds in the group represented by the row. A minus-sign means a lowering of activity; a zero means no effect. Solvents that lower activity should be chosen for extraction.

A.7 Program Listing of EXSEP

MAIN MODULE (MAIN.ARI)

```
/************************************************/
:-public main/0,restart/0,
        synthesize_process/3,report/0,
        readdata_default/1,read_key/1,
        read_solvent/2.
:-extrn
        absorb/6:far,
        append/3:far,
        append_v/3:far,
        bypass/3,
        component_flow/3:far,
        component_flow_list/4:far,
        component_flow_set/3:far,
        decide_difficult_or_easy/2:far,
        delete_element/3:far,
        delete_elements/3:far,
        equal/2:far,
        erase_term/2:far,
        ffactor/3:far,
        first_prod/2:far,
        flow_set/3:far,
        flow_set_c/3:far,
        get_split_flow/3:far,
        input_henrys_law/1:far,
        largest_flow/3:far,
        last_prod/2:far,
        length/2:far,
        list_member/2:far,
        lle/6:far,
        max/3:far,
        max_flist/2:far,
        max_flow/3:far,
        member/2:far,
        pause/0:far,
        pause_escape/0:far,
        positive_component_flow_list/2:far,
        print_cam_driver/3:far,
        product_flow/3:far,
        product_flow_match/4:far,
        product_flow_set/3:far,
        reverse/2:far,
        selected_positive_component_flow_list/3:far,
        separation_specification_table/2:far,
        set_above/3:far,
        set_below/3:far,
        split/6:far,
        split_above/4:far,
        split_below/4:far,
        stripping/6:far,
        sub_list/2:far,
        sum_component_flow/3:far,
        sum_flist/2:far,
        two_highest_flows/3:far.
```

```
main:-run.

restart:-
        uncheck_processes([od,dga,s,lle]),
        write('Program Interrupted').


/************************************************/
run:-
        consult('menu.ari'),
        repeat,
        initialize(List,CList,Methods,Graphics),
        synthesize_process(List,CList,Methods),
        report,
        terminate(Graphics,Methods).

initialize(List,CList,Methods,Graphics):-
        define_window(one,label('Knowledge-Based Process Synthesis',center,30,0),(0,0),(24,79),(30,-30)),
        define_window(two,label('Component Assignment Matrix',center,113,0),(1,0),(24,79),(113,-113)),
        define_window(three,label('Bypass Analysis',center,49,0),(2,0),(24,79),(49,-49)),
        define_window(four,label('Process Feasibility Analysis',center,79,0),(2,0),(24,79),(79,-79)),
        define_window(five,label('Process Heuristic Analysis',center,15,0),(2,0),(24,79),(116,-15)),
        define_window(six,label('Separation Specification Table',center,10,0),(2,0),(24,79),(29,-10)),
        define_window(seven,label('Diagnosis',center,31,0),(3,54),(15,79),(31,-255)),
        define_window(eight,label('Recovery Column Calculation',center,113,0),(1,0),(24,79),(127,-31)),
        current_window(_,one),
        cls,nl,
        write('       EEEEEEEEE XXX     XXX  SSSSSSSS  EEEEEEEEE  PPPPPPPPP'),nl,
        write('       EEEEEEEEE XXX    XXX  SSSSSSSSS  EEEEEEEEE  PPPPPPPPPPP'),nl,
        write('       EEE       XXX  XXX   SSS       EEE       PPP     PPP'),nl,
        write('       EEE        XXX XXX    SSS      EEE       PPP     PPP'),nl,
        write('       EEE         XXXXX     SSS      EEE       PPP    PPP'),nl,
        write('       EEEEEE       XXX      SSSSSSSSS  EEEEEE     PPPPPPPPP'),nl,
        write('       EEE         XXXXX         SSS  EEE       PPP'),nl,
        write('       EEE        XXX XXX        SSS  EEE       PPP'),nl,
        write('       EEE       XXX  XXX       SSS  EEE       PPP    '),nl,
        write('       EEEEEEEEE XXX    XXX  SSSSSSSSS  EEEEEEEEE  PPP    '),nl,
        write('       EEEEEEEEE XXX     XXX SSSSSSSS   EEEEEEEEE  PPP    '),nl,nl,nl,nl,
        write('    Please check the'),nl,
        write('    separation method'),nl,
        write('    that you wish'),nl,
        write('    to consider.'),
        region_ca((0,0),(12,70),Graphics),
        repeat,
        send_menu_msg(activate(separations,(13,32)),Choice),
        check_choice(Choice,separations),
        more_wanted,
        send_menu_msg(checked(separations,od),OD),   % od = ordinary distillation
        send_menu_msg(checked(separations,dga),DGA), % dga = dilute gas absorption
        send_menu_msg(checked(separations,bga),BGA), % bga = bulk gas absorption
        send_menu_msg(checked(separations,s),S),     % s = stripping
        send_menu_msg(checked(separations,lle),LLE), % lle = liquid-liquid extraction
        send_menu_msg(checked(separations,ed),ED),   % ed = extractive distillation
        send_menu_msg(checked(separations,ad),AD),   % ad = heterogeneous azeotropic distillation
        condense_methods([od,dga,bga,s,lle,ed,ad],[OD,DGA,BGA,S,LLE,ED,AD],Methods),
        eraseall(data),
        eraseall(sep),
        load_data(List,CList,Methods),!.

check_choice(unchecked(Item),Menu):-
```

```
        send_menu_msg(check(Menu,Item),_).
check_choice(checked(_),_).

more_wanted:-
        region_ca((0,0),(12,70),X),
        cls,
        region_ca((0,0),(12,70),X),
        tmove(14,2),
        write('Do you wish to consider other separations?'),nl,
        send_menu_msg(activate(no_yes,(17,32)),Decision),
        Decision = no.
more_wanted:-
        region_ca((0,0),(12,70),X),
        cls,
        region_ca((0,0),(12,70),X),
        fail.

condense_methods([],[],[]).
condense_methods([H1|T1],[checked|T2],[H1|T3]):-
        condense_methods(T1,T2,T3).
condense_methods([_|T1],[_|T2],T3):-
        condense_methods(T1,T2,T3).

load_data(_,_,[]):-
        erase_term(sep,dseparator(_,_,_,_,_)),
        recorda(sep,dseparator(0,[],[],[],[]),_),!.
load_data(List,CList,[od|T]):-
        cls,nl,
        write(' Ordinary distillation has been chosen as a separation method.'),nl,
        write(' Please enter desired choice for data loading.'),nl,
        send_menu_msg(activate(data_loading,(14,30)),Choice),
        resolve_choice(Choice,od),
        recorded(data,initial_set(List),_),
        recorded(data,initial_components(CList),_),
        load_data(List,CList,T).
load_data(List,CList,[dga|T]):-
        cls,nl,
        write(' Dilute gas absorption has been chosen as a separation method.'),nl,
        write(' Please enter desired choice for data loading.'),nl,
        send_menu_msg(activate(data_loading,(14,30)),Choice),
        resolve_choice(Choice,dga),
        ((var(List),recorded(data,initial_set(List),_));true),
        ((var(CList),recorded(data,initial_components(CList),_));true),
        load_data(List,Clist,T).
load_data(List,CList,[s|T]):-
        cls,nl,
        write(' Stripping has been chosen as a separation method.'),nl,
        write(' Please enter desired choice for data loading.'),nl,
        send_menu_msg(activate(data_loading,(14,30)),Choice),
        resolve_choice(Choice,s),
        ((var(List),recorded(data,initial_set(List),_));true),
        ((var(CList),recorded(data,initial_components(CList),_));true),
        load_data(List,Clist,T).

load_data(List,CList,[lle|T]):-
        cls,nl,
        write(' Liquid Liquid Extraction has been chosen as a separation method.'),nl,
        write(' Please enter desired choice for data loading.'),nl,
        send_menu_msg(activate(data_loading,(14,30)),Choice),
        resolve_choice(Choice,lle),
        ((var(List),recorded(data,initial_set(List),_));true),
```

```
        ((var(CList),recorded(data,initial_components(CList),_));true),
        load_data(List,Clist,T).


synthesize_process([_],_,_):- cls, current_window(_,one),cls,!.
synthesize_process(_,[_],_):- cls, current_window(_,one),cls,!.
synthesize_process(List,CList,Methods):-
        cls,current_window(_,two),                      % planner
        print_cam_driver(List,CList,'fresh'),                   % planner
        pause,cls,current_window(_,three),                      % planner
        bypass(List,CList,Executed_bypass),                     % planner
        decide_to_reprint_cam(List,CList,Executed_bypass),  % planner
        cls,current_window(_,four),                             % planner
        generate_and_test(List,CList,Methods,TopList,TopCList,BotList,BotCList),
        synthesize_process(TopList,TopCList,Methods),
        synthesize_process(BotList,BotCList,Methods),!.

generate_and_test(List,CList,[od],TopList,TopCList,BotList,BotCList):-
        separation_specification_table(List,CList),
        split(List,CList,TopList,TopCList,BotList,BotCList),!.

generate_and_test(List,CList,[dga],TopList,TopCList,BotList,BotCList):-
        absorb(List,CList,TopList,TopCList,BotList,BotCList),!.

generate_and_test(List,CList,[s],TopList,TopCList,BotList,BotCList):-
        stripping(List,CList,TopList,TopCList,BotList,BotCList),!.

generate_and_test(List,CList,[lle],TopList,TopCList,BotList,BotCList):-
        lle(List,CList,TopList,TopCList,BotList,BotCList),!.

% need to integrate distillation alone, absorption alone, or combination

report:-
        write('Separator  Bypass Around  Stream Flow In     Overhead      Bottoms'),nl,
        write('———————  ——————————  ——————————  ——————————  ——————————'),
        nl,
        recorded(sep,dseparator(0,_,_,_,_),Ref),
        erase(Ref),
        reverse_dseparator_database(1,LastSepNum),
        sequence_print(LastSepNum),
        erase_term(sep,dseparator),
        erase_term(sep,dstreamin),
        erase_term(bypass,dstreambypass),
        eraseall(data),
        pause,!.
        /*          Initializing Utilities                  */

resolve_choice(1,_):-
        nl,
        write(' Please enter file name to consult'),nl,
        read(Newfilename),nl,
        open(H,Newfilename,r),
        read_data(H),
        close(H).

read_data(H):-
        repeat,
        read(H,T),
        recorda(data,T,_),
        T=end_of_file.
```

```
resolve_choice(2,od):-
        distillation_input.

resolve_choice(2,dga):-
        absorb_input.

resolve_choice(2,s):-
        stripping_input.

resolve_choice(2,s):-
        lle_input.

distillation_input:-
        cls,write('                    DATA LOADING'),nl,
        write(' How many products are in the system?'),nl,
        read(NumProd),nl,
        write(' How many components are in the system?'),nl,
        read(NumComp),
        readdata(NumComp,NumProd),
        write(' Do you wish to save this information to a file?'),nl,
        send_menu_msg(activate(yes_no,(10,60)),Choicesave),!,
        save_to_file(Choicesave).

absorb_input:-
        cls,write('                    DATA LOADING'),nl,
        readdata_MSA(dga),
        readdata_default(dga),
        write(' Do you wish to save this information to a file?'),nl,
        send_menu_msg(activate(yes_no,(10,60)),Choicesave),!,
        save_to_file(Choicesave).

stripping_input:-
        cls,write('                    DATA LOADING'),nl,
        readdata_MSA(s),
        readdata_default(s),
        write(' Do you wish to save this information to a file?'),nl,
        send_menu_msg(activate(yes_no,(10,60)),Choicesave),!,
        save_to_file(Choicesave).

lle_input:-
        cls,write('                    DATA LOADING'),nl,
        readdata_MSA(lle),
        readdata_default(lle),
        write(' Do you wish to save this information to a file?'),nl,
        send_menu_msg(activate(yes_no,(10,60)),Choicesave),!,
        save_to_file(Choicesave).

readdata(NumComp,NumProd):-
        readcomponents(NumComp,CList,[],1),
        readproducts(NumProd,List,[],1),
        readflows(List,CList),
        read_k_values_and_boiling_points(CList),
        nl,nl,nl,nl,nl,write(' Are any components corrosive or hazardous?'),nl,
        write(' Enter 1 for yes and 2 for no'),nl,
        readcorrosive(Choicecor,CList),nl,nl,
        write(' Are there any pseudoproducts?'),nl,
        write(' Enter 1 for Yes and 2 for no'),nl,
        readpseudo(Choicepseudo,List).

readdata_MSA(Sep):-
        write(' How many products are in the system ? '),
```

```
        read(NumProd),nl,
        write(' How many components are in the system ? '),
        read(NumComp),nl,
        readproducts(NumProd,List,[],1),nl,
        readcomponents(NumComp,CList,[],1),nl,
        input_henrys_law(CList),nl,
        readflows(List,CList),
        readsolvent(Sep,Name),nl,
        readmolefrac(Sep,CList,Name),nl,
        readkey(Sep),nl,
        readsplit,nl.

readcomponents(NumComp,CList,RefList,Num):-
        Num =< NumComp,
        write(' What is the name of component '),write(Num),write(' ? '),
        read(Comp),
        append(RefList,[Comp],NewRefList),
        NNum is Num + 1,
        readcomponents(NumComp,CList,NewRefList,NNum).
readcomponents(NumComp,CList,CList,Num):-
        Num > NumComp,
        recorda(data,initial_components(CList),_).
readproducts(NumProd,List,RefList,Num):-
        Num =< NumProd,
        write(' What is the name of product '),write(Num),write(' ? '),
        read(Prod),
        append(RefList,[Prod],NewRefList),
        NNum is Num + 1,
        readproducts(NumProd,List,NewRefList,NNum).
readproducts(NumProd,List,List,Num):-
        Num > NumProd,
        recorda(data,initial_set(List),_).

readflows([],_).
readflows([Head|Tail],CList):-
        read_flows_for_product(Head,CList),nl,
        readflows(Tail,CList).

read_flows_for_product(_,[]).
read_flows_for_product(Prod,[Head|Tail]):-
        write(' What is the flow of '),write(Head),write(' in '),write(Prod),write(' ? '),
        read(X),
        recorda(data,flow(Prod,Head,X),_),
        read_flows_for_product(Prod,Tail).

read_k_values_and_boiling_points([]).
read_k_values_and_boiling_points([Head|Tail]):-
        write(' What is the normal boiling point (C) of '),write(Head),write(' ?'),nl,    read(X),
        recorda(data,boiling_temp(Head,X),_),
        write(' What is the k-value of '),write(Head),write(' ?'),nl,
        read(Y),
        recorda(data,k_value(Head,Y),_),
        read_k_values_and_boiling_points(Tail).

readsolvent(Sep,Name):-
        Sep = s,
        write(' What is the name of the lean gas ? '),
        read(Name),
        recorda(data,lean_gas(Name),_).
readsolvent(Sep,Name):-
        write(' What is the name of the solvent ? '),
```

```
        read(Name),
        recorda(data,solvent(Name),_).

readmolefrac(_,[],_).
readmolefrac(Sep,[H|T],Name):-
        Sep = dga,
        write(' What is the mole fraction of '),write(H),write(' in '),
        write(Name),write(' ? '),
        read(X),
        recorda(data,mole_fraction(solvent,H,X),_),
        readmolefrac(Sep,T,Name).
readmolefrac(Sep,[H|T]):-
        Sep = s,
        write(' What is the mole fraction of '),write(H),write(' in '),
        write(Name),write(' ? '),
        read(X),
        recorda(data,mole_fraction(lean_gas,H,X),_),
        readmolefrac(Sep,T,Name).
readmolefrac(Sep,[H|T]):-
        Sep = lle,
        write(' What is the mole fraction of '),write(H),write(' in '),
        write(Name),write(' ? '),
        read(X),
        recorda(data,mole_fraction(solvent,H,X),_),
        readmolefrac(Sep,T,Name).

readkey(Sep):-
        repeat,
        write(' What is the key component ? '),
        read(Key),
        member(Key,CList),
        recorda(data,key_component(Sep,Key),_).

readsplit:-
        write(' What is the split product ? '),
        read(Split_Pro),
        recorda(data,split_product(Split_Pro),_).

readcorrosive(2,_).
readcorrosive(_,[]).
readcorrosive(1,[Head|Tail]):-
        write(' Is '),write(Head),write(' considered corrosive or hazardous?'),nl,
        write(' Enter 1 for yes and 2 for no'),nl,
        read(Choicecor),
        resolve_corrosive(Head,Choicecor),
        readcorrosive(2,Tail).

resolve_corrosive(_,2).
resolve_corrosive(Comp,1):-
        recordz(data,corrosive(Comp),_).

readpseudo(2,_).
readpseudo(1,List):-
        write(' How many pseudoproducts are there?'),
        read(NumPseudo),
        readpseudo_products(List,NumPseudo,1).

readpseudo_products(_,NumPseudo,N):-
        N > NumPseudo.
readpseudo_products(List,NumPseudo,N):-
        nl,write(' What is the name of the actual product?'),nl,
```

```
            read(Prod),
            build_pseudo(Prod,List,[]),
            NNumPseudo is N + 1,
            readpseudo_products(List,NumPseudo,NNumPseudo).

build_pseudo(Prod,[],PseudoList):-
            recordz(data,pseudoproduct(PseudoList,Prod),_).
build_pseudo(Prod,[Head|Tail],PseudoList):-
        write(' Is '),write(Head),write(' a pseudoproduct of '),write(Prod),write(' ?'),nl,
        write(' Enter 1 for yes and 2 for No'),nl,
        read(Choicepseudo),
        add_pseudo(Choicepseudo,Head,PseudoList,NPseudoList),
        build_pseudo(Prod,Tail,NPseudoList).

add_pseudo(2,_,N,N).
add_pseudo(1,Head,PseudoList,NPseudoList):-
        append(PseudoList,[Head],NPseudoList).

readdata_default(dga):-
            open(G,'default.dga',r),
            read_data(G),
            close(G).
readdata_default(s):-
            open(G,'default.s',r),
            read_data(G),
            close(G).
readdata_default(lle):-
            open(G,'default.lle',r),
            read_data(G),
            close(G).
readdata_default(rec):-
            open(G,'default.rec',r),
            read_data(G),
            close(G).

save_to_file(no).
save_to_file(yes):-
        nl,write(' Please enter file name'),nl,
        read(FileName),
            create(H,FileName),
            findall(X,recorded(data,X,_),ListX),
            reverse(ListX,XList),
            write_data_file(H,XList),
            close(H).

write_data_file(H,[]).
write_data_file(H,[HX|TX]):-
            writeq(H,HX),write(H,'.'),
            nl(H),
            write_data_file(H,TX).


/*          End of Initializing Utilities                */

decide_to_reprint_cam(_,_,no).
decide_to_reprint_cam(List,CList,yes):-
            cls,current_window(_,two),
        print_cam_driver(List,CList,'bypass'),
            pause.

reverse_dseparator_database(Num,LastSepNum):-
```

```
        recorded(sep,dseparator(Num,TopCList,TopFList,BotCList,BotFList),Ref),
        erase(Ref),
        recordz(sep,dseparator(Num,TopCList,TopFList,BotCList,BotFList),_),
        NNum is Num + 1,
        reverse_dseparator_database(NNum,LastSepNum),!.
reverse_dseparator_database(Num,LastSepNum):-
        LastSepNum is Num - 1,!.

sequence_print(LastSepNum):-
        recorded(sep,dseparator(Num,TopCList,TopFList,BotCList,BotFList),_),nl,
        tget(Row,_),
        tmove(Row,4),
        write('S'),write(Num),
        tmove(Row,12),
        retrieve_and_print_bypass(Num,BypassRow),
        tmove(Row,27),
        retrieve_and_print_streamflow(Num,StreamRow),
        tmove(Row,43),
        print_overhead(TopCList,TopFList,OvhdRow),
        tmove(Row,59),
        print_bottoms(BotCList,BotFList,BottomsRow),
        recorded(sep,dseparator(Num,TopCList,TopFList,BotCList,BotFList),Ref),
        erase(Ref),
            max(BypassRow,StreamRow,X),
        max(OvhdRow,BottomsRow,Y),
        max(X,Y,Z),
        decide_cursor(Z,Num,LastSepNum),
        fail.
sequence_print(_).

retrieve_and_print_bypass(Num,BypassRow):-
        recorded(bypass,dstreambypass(Num,Prod,CList,FList),_),
        tget(_,Column),
        Prod \= 'none',
        write(Prod),write(': '),
        print_component_and_bypassflow(CList,FList,LastRow),
        recorded(bypass,dstreambypass(Num,Prod,CList,FList),Ref),
            erase(Ref),
        NRow is LastRow + 1,
        tmove(NRow,Column),
        retrieve_and_print_bypass(Num,BypassRow),!.
retrieve_and_print_bypass(Num,BypassRow):-
        recorded(bypass,dstreambypass(Num,'none',_,_),_),
        write(' none'),
        tget(Row,_),
        BypassRow is Row + 1,!.
retrieve_and_print_bypass(_,BypassRow):-
        tget(Row,_),
        BypassRow is Row,!.

retrieve_and_print_streamflow(Num,StreamRow):-
        recorded(sep,dstreamin(Num,CList,FList),_),
        print_component_and_streamflow(CList,FList),
        tget(Row,_),
        StreamRow is Row + 1,!.

print_component_and_bypassflow([],[],LastRow):-
        tget(LastRow,_).
print_component_and_bypassflow([HeadC|TailC],[HeadF|TailF],LastRow):-
        tget(R,C),
            WHeadF is round(HeadF,2),
```

```
        write(WHeadF),tab(1),write(HeadC),
        NR is R + 1,
        tmove(NR,C),
        print_component_and_bypassflow(TailC,TailF,LastRow),!.

print_component_and_streamflow([],[]):- !.
print_component_and_streamflow([HeadC|TailC],[HeadF|TailF]):-
        tget(R,C),
            WHeadF is round(HeadF,2),
            write(WHeadF),tab(1),write(HeadC),
        NR is R + 1,
        tmove(NR,C),
        print_component_and_streamflow(TailC,TailF),!.

print_overhead([],[],OvhdRow):-
        tget(Row,_),
        OvhdRow is Row,!.
print_overhead([TopHeadC|TopTailC],[TopHeadF|TopTailF],OvhdRow):-
        tget(R,C),
            WTopHeadF is round(TopHeadF,2),
        write(WTopHeadF),tab(1),write(TopHeadC),
        NR is R + 1,
        tmove(NR,C),
        print_overhead(TopTailC,TopTailF,OvhdRow),!.

print_bottoms([],[],BottomsRow):-
        tget(Row,_),
        BottomsRow is Row,!.
print_bottoms([BotHeadC|BotTailC],[BotHeadF|BotTailF],BottomsRow):-
        tget(R,C),
            WBotHeadF is round(BotHeadF,2),
        write(WBotHeadF),tab(1),write(BotHeadC),
        NR is R + 1,
        tmove(NR,C),
        print_bottoms(BotTailC,BotTailF,BottomsRow),!.

decide_cursor(Z,_,_):-
        Z < 18,
        tmove(Z,1),!.
decide_cursor(Z,Num,LastSepNum):-
        Z >= 18,nl,nl,
        Num < LastSepNum,
        write('Separator  Bypass Around  Stream Flow In      Overhead      Bottoms'),nl,
        write('---------  -------------  ----------------  ------------  ----------'),nl,!.
decide_cursor(_,Num,Num).


terminate(Graphics,Methods):-
            cls,
            nl,nl,nl,
            write(' Do you wish to terminate the program or'),nl,
        write(' synthesize another process?'),nl,
            send_menu_msg(activate(terminate,(13,32)),Choice),
            Choice = synthesize_another,
            uncheck_processes(Methods),
            cls,
            region_ca((0,0),(12,70),Graphics),!,
            fail.

terminate(_,Methods):-
            cls,
```

```
        uncheck_processes(Methods),
        current_window(_,main),
        delete_window(one),
        delete_window(two),
        delete_window(three),
        delete_window(four),
        delete_window(five),
        delete_window(six),
        delete_window(seven),
        delete_window(eight).

uncheck_processes([]):- !.
uncheck_processes([H|T]):-
        send_menu_msg(uncheck(separations,H),_),
        uncheck_processes(T).
```

BYPASS MODULE (BYPASS.ARI)

```
:-public bypass/3.
:-visible member/2,equal/2.
:-extrn
        append/3:far,
        append_v/3:far,
        component_flow/3:far,
        component_flow_list/4:far,
        component_flow_set/3:far,
        decide_difficult_or_easy/2:far,
        delete_element/3:far,
        delete_elements/3:far,
        equal/2:far,
        erase_term/2:far,
        ffactor/3:far,
        first_prod/2:far,
        flow_set/3:far,
        flow_set_c/3:far,
        largest_flow/3:far,
        last_prod/2:far,
        list_member/2:far,
        max/3:far,
        max_flist/2:far,
        max_flow/3:far,
        member/2:far,
        pause/0:far,
        pause_escape/0:far,
        positive_component_flow_list/2:far,
        product_flow/3:far,
        product_flow_match/4:far,
        product_flow_set/3:far,
        reverse/2:far,
        selected_positive_component_flow_list/3:far,
        set_above/3:far,
        set_below/3:far,
        split_above/4:far,
        split_below/4:far,
        sub_list/2:far,
        sum_component_flow/3:far,
        sum_flist/2:far,
        two_highest_flows/3:far.

bypass(List,CList,Executed_bypass):-
    eraseall(bypass),
    cls,
    check_bypassing(List,CList),
    solve_bypass_flow(List,CList),
    nl,nl,nl,write(' The bypass analysis is complete for this CAM.'),nl,
    recorded(bypass,status_executed(Executed_bypass),_),
    erase_term(bypass,dbypass_result(_,_)),
    erase_term(bypass,dbypass_amount(_,_,_)),!.

check_bypassing(_,[]).
check_bypassing(List,CList):-
    recorda(bypass,dbypass_status(nobypasspossible),_),
    check_bypassing_equal(List,List,CList),
    bypassing_choice_list(List).
```

```
check_bypassing_equal(_,[],_).
check_bypassing_equal(List,[Head|Tail],CList):-
    selected_positive_component_flow_list([Head],CList,ComponentsInProductStream),
    equal(ComponentsInProductStream,CList),
    recorda(bypass,dbypass_result(List,Head),_),
    check_bypassing_equal(List,Tail,CList).
check_bypassing_equal(List,[Head|Tail],CList):-
    recorded(data,pseudoproduct([Head|X],ActualProd),_),
    list_member([Head|X],List),
    selected_positive_component_flow_list([Head|X],CList,ComponentsInProductStream),
    equal(ComponentsInProductStream,CList),
    recordz(bypass,dbypass_result(List,ActualProd),_),
    check_bypassing_equal(List,Tail,CList).
check_bypassing_equal(List,[_|Tail],CList):-
    check_bypassing_equal(List,Tail,CList).

bypassing_choice_list(List):-
    recorded(bypass,dbypass_result(List,Prod),_),
    recorda(bypass,dbypass_status(bypasspossible),_),
    write(' Product '),write(Prod),write(' is all-component-inclusive and subject to bypass.'),nl,
    write(' Do you wish to bypass this product?'),
    send_menu_msg(activate(yes_no,(17,32)),Decision),
    resolve_bypass_choice(Decision,List,Prod),
    recorded(bypass,dbypass_result(List,Prod),Ref),
    erase(Ref),
    fail.
bypassing_choice_list(_):-
    recorded(bypass,dbypass_status(bypasspossible),_),
    erase_term(bypass,dbypass_status(_)),
    recorda(bypass,dbypass_status(bypasspossible),_).
bypassing_choice_list(_):-
    recorded(bypass,dbypass_status(nobypasspossible),_),
    erase_term(bypass,dbypass_status(_)),
    recorda(bypass,dbypass_status(nobypasspossible),_),
    nl,write(' Bypass is not possible with this CAM.'),nl,
    recorded(sep,dseparator(Num,_,_,_,_),_),
    NNum is Num +1,
    recordz(bypass,dstreambypass(NNum,none,[],[]),_),
    recorda(bypass,status_executed('no'),_),
    pause.

resolve_bypass_choice(yes,List,Prod):-
    cls,
    write('Choose the percent bypass desired:'),nl,
    send_menu_msg(activate(bypass,(15,32)),ChoiceBypassPercent),
    resolve_choice_bypass_percent(ChoiceBypassPercent,Percent),
    cls,
    Fract is 0.01*Percent,
    recordz(bypass,dbypass_amount(List,Prod,Fract),_),
    recorda(bypass,status_executed('yes'),_),!.

resolve_bypass_choice(no,_,Prod):-
    nl,nl,write('Product '),write(Prod),write(' has been dropped from'),nl,
    write('bypass considerations for this CAM.'),nl,
    recorded(sep,dseparator(Num,_,_,_,_),_),
    NNum is Num +1,
    recordz(bypass,dstreambypass(NNum,'none',[],[]),_),pause,cls,
    recorda(bypass,status_executed('no'),_),!.

resolve_choice_bypass_percent('100',Percent):-
    Percent is 100,!.
```

```
resolve_choice_bypass_percent('90',Percent):-
    Percent is 90,!.
resolve_choice_bypass_percent('manual',Percent):-
    nl,write(' Enter the percent bypass you desire'),nl,
    read(Percent),!.
resolve_choice_bypass_percent('manual',Percent):-
    cls,nl,
    write(' Input error. Entry must be a real number.'),nl,
    resolve_choice_bypass_percent('manual',Percent),!.

solve_bypass_flow(List,CList):-
    component_flow_set(List,CList,FList),
    flow_of_bypass_products(List,List,CList,FList,NFList),
    write(' Feed stream flow rate to separator is: '),nl,
    recorded(sep,dseparator(Num,_,_,_,_),_),
    NNum is Num +1,
    recordz(sep,dstreamin(NNum,CList,NFList),_),
    write('  Component   Flow'),nl,
    write('  ---------  ----'),nl,
    print_component(CList,NFList),
    pause.

flow_of_bypass_products(_,[],_,N,N).

flow_of_bypass_products(List,[Head|Tail],NewCList,FList,NFList):-
    recorded(bypass,dbypass_amount(List,Head,Fract),_),
    component_flow_set([Head],NewCList,ProductFList),
    limiting_component(NewCList,Component,FList,ProductFList),
    recorded(data,flow(Head,Component,LimitingComponentFlow),_),
    BypassFlowofLimComp is Fract*LimitingComponentFlow,
    bypass_flow_list(List,Component,BypassFlowofLimComp,NewCList,BypassFlowList),
    not_bypassed_flow_list(NewCList,FList,BypassFlowList,NewFList),
    update_flow_dbase_from_bypass(Head,NewCList,BypassFlowList,List),
    write('Bypass flow to product '),write(Head),write(' is:'),nl,
    write('  Component   Flow'),nl,
    print_component(NewCList,BypassFlowList),
    pause,
    recorded(sep,dseparator(Num,_,_,_,_),_),
    NNum is Num +1,
    recordz(bypass,dstreambypass(NNum,Head,NewCList,BypassFlowList),_),
    flow_of_bypass_products(List,Tail,NewCList,NewFList,NFList).

flow_of_bypass_products(List,[_|Tail],NewCList,FList,NFList):-
    recorded(bypass,dbypass_amount(List,ActualProd,Fract),_),
    not(member(ActualProd,List)),
    recorded(data,pseudoproduct(PList,ActualProd),_),
    component_flow_set(PList,NewCList,ProductFList),
    limiting_component(NewCList,Component,FList,ProductFList),
    component_flow(PList,Component,LimitingComponentFlow),
    BypassFlowofLimComp is Fract*LimitingComponentFlow,
    bypass_flow_list(List,Component,BypassFlowofLimComp,NewCList,BypassFlowList),
    not_bypassed_flow_list(NewCList,FList,BypassFlowList,NewFList),
    update_flow_dbase_from_bypass(ActualProd,NewCList,BypassFlowList,List),
    write('Bypass flow to product '),write(ActualProd),write(' is:'),nl,
    write('  Component   Flow'),nl,
    print_component(NewCList,BypassFlowList),
    pause,
    recorded(sep,dseparator(Num,_,_,_,_),_),
    NNum is Num + 1,
    recordz(bypass,dstreambypass(NNum,ActualProd,NewCList,BypassFlowList),_),
    recorded(bypass,dbypass_amount(List,ActualProd,Fract),Ref),
```

```
    erase(Ref),
    flow_of_bypass_products(List,Tail,NewCList,NewFList,NFList).


flow_of_bypass_products(List,[_|Tail],NewCList,NewFList,NFList):-
    flow_of_bypass_products(List,Tail,NewCList,NewFList,NFList).

limiting_component([Head1|Tail1],Component,[Head2|Tail2],[Head3|Tail3]):-
    Ratio1 is Head3/Head2,
    limiting_component_find(Ratio1,Head1,Component,Tail1,Tail2,Tail3).

limiting_component_find(_,Refcomponent,Component,[],[],[]):-
    Component = Refcomponent.
limiting_component_find(Refratio,Refcomponent,Component,[_|T1],[H2|T2],[H3|T3]):-
    Newratio is H3/H2,
    Newratio > Refratio,
    limiting_component_find(Refratio,Refcomponent,Component,T1,T2,T3).
limiting_component_find(Refratio,_,Component,[H1|T1],[H2|T2],[H3|T3]):-
    Newratio is H3/H2,
    Newratio =< Refratio,
    limiting_component_find(Newratio,H1,Component,T1,T2,T3).

bypass_flow_list(List,Component,BypassFlowofLimComp,NewCList,BypassFlowList):-
    component_flow(List,Component,CFlow),
    bypass_flow_of_components(NewCList,List,CFlow,BypassFlowofLimComp,BypassFlowList,[]).

bypass_flow_of_components([],_,_,_,BypassFlowList,RefList):-
    BypassFlowList = RefList.
bypass_flow_of_components([Head|Tail],List,RefFlow1,RefFlow2,BypassFlowList,RefList):-
    component_flow(List,Head,TotalCFlow),
    BypassFlow is RefFlow2*TotalCFlow/RefFlow1,
    append(RefList,[BypassFlow],NewRefList),
    bypass_flow_of_components(Tail,List,RefFlow1,RefFlow2,BypassFlowList,NewRefList).

not_bypassed_flow_list([],[],[],[]).

not_bypassed_flow_list([_|Tail1],[Head2|Tail2],[Head3|Tail3],[Head4|Tail4]):-
    Head4 is Head2 - Head3,
    not_bypassed_flow_list(Tail1,Tail2,Tail3,Tail4).

print_component([],[]).
print_component([Head1|Tail1],[Head2|Tail2]):-
    tab(5),write(Head1),tget(R,_),tmove(R,14),write(Head2),nl,
    print_component(Tail1,Tail2).

update_flow_dbase_from_bypass(_,[],[],_).
update_flow_dbase_from_bypass(Prod,[Head1|Tail1],[Head2|Tail2],List):-
    member(Prod,List),
    recorded(data,flow(Prod,Head1,OldFlow),_),
    NewFlow is OldFlow - Head2,
    NewFlow >= 0.0001,
    recorded(data,flow(Prod,Head1,OldFlow),Ref),
    erase(Ref),
    recorda(data,flow(Prod,Head1,NewFlow),_),
    update_flow_dbase_from_bypass(Prod,Tail1,Tail2,List).
update_flow_dbase_from_bypass(Prod,[Head1|Tail1],[Head2|Tail2],List):-
    member(Prod,List),
    recorded(data,flow(Prod,Head1,OldFlow),_),
    NewFlow is OldFlow - Head2,
    NewFlow < 0.0001,
    recorded(data,flow(Prod,Head1,OldFlow),Ref),
```

```
    erase(Ref),
    recorda(data,flow(Prod,Head1,0),_),
    update_flow_dbase_from_bypass(Prod,Tail1,Tail2,List).
update_flow_dbase_from_bypass(ActualProd,[Head1|Tail1],[Head2|Tail2],List):-
    not(member(ActualProd,List)),
    recorded(data,pseudoproduct(PList,ActualProd),_),
    update_pseudoproduct_flows(PList,Head1,Head2),
    update_flow_dbase_from_bypass(ActualProd,Tail1,Tail2,List).

update_pseudoproduct_flows([],_,_).
update_pseudoproduct_flows([Head|_],Head1,Head2):-
    recorded(data,flow(Head,Head1,OldFlow),_),
    OldFlow > 0,
    NewFlow is OldFlow - Head2,
    NewFlow >= 0.0001,
    recorded(data,flow(Head,Head1,OldFlow),Ref),
    erase(Ref),
    recorda(data,flow(Head,Head1,NewFlow),_).
update_pseudoproduct_flows([Head|_],Head1,Head2):-
    recorded(data,flow(Head,Head1,OldFlow),_),
    OldFlow > 0,
    NewFlow is OldFlow - Head2,
    NewFlow >= 0,
    NewFlow < 0.0001,
    recorded(data,flow(Head,Head1,OldFlow),Ref),
    erase(Ref),
    recorda(data,flow(Head,Head1,0),_).

update_pseudoproduct_flows([_|Tail],Head1,Head2):-
    update_pseudoproduct_flows(Tail,Head1,Head2).
```

SEPARATION SPECIFICATION TABLE MODULE (SST.ARI)

```
/*****************************************************************
*          SEPARATION SPECIFICATION TABLE MODULE                 *
*                                                                 *
* This module develops the SST. Its ultimate goal is to recordz*
* the following fact to the Prolog database:                      *
*                                                                 *
*  (dsst(List,Clist,Split,Top,Bot,LK,HK,Delta,Listofd,Listofb,*
*      Status,Ease,CES),table)                                    *
*                                                                 *
* where:                                                          *
*      List = list of feed products, e.g., [p1,p2,p3,p4].         *
*     CList = list of feed components, e.g., [a,b,c,d].           *
*     Split = 'sharp or 'sloppy', depending on the split.         *
*       Top = list of top products, e.g., [p1,p2] for a           *
*             sloppy split, or [a,b] for a sharp split.           *
*       Bot = list of bottoms products, e.g., [p3,p4] for a       *
*             sloppy split, or [c,d] for a sharp split.           *
*        LK = the light key component in the split                *
*        HK = the heavy key component in the split                *
*     Delta = normal boiling point temperature difference         *
*             between the light key and heavy key component,      *
*             in F or C.                                          *
*   Listofd = list of distillate recoveries for each component*
*             in CList, e.g., for [a,b,c,d] a Listofd could       *
*             be [0.98,0.98,0.60,0.02].                           *
*   Listofb = list of bottoms recoveries for each component in*
*             CList, e.g., [0.02,0.02,0.40,0.98].                 *
*    Status = 'feasible' or 'infeasible', depending on            *
*             component recovery specification test and           *
*             nonkey component distribution test.                 *
*      Ease = difficult or easy, depending on the temperature     *
*             difference between the LK and HK component. This*
*             paramter applies to heuristic S2. If a split is     *
*             labelled 'difficult', it is considered an           *
*             essential last split by S2.                         *
*       CES = the coefficient of ease of separation for the       *
*             split.                                              *
*****************************************************************/


/*****************************************************************
* The separation_specification_table clause calls two main       *
* drivers: sharp_split_sst_driver and sloppy_split_sst_driver. *
* These drivers assess split feasibility, calculate the CES for*
* feasible splits, and record the result into the Prolog         *
* database via the recordz(dsst(... ))statement. The SST clause*
* then reports the results to the user.                          *
*****************************************************************/

:-segment(sstseg).
:-public separation_specification_table/2:far,solve_ratio/3:far.
:-visible equal/2.
:-extrn
        append/3:far,
```

```
            append_v/3:far,
            component_flow/3:far,
            component_flow_list/4:far,
            component_flow_set/3:far,
            decide_difficult_or_easy/2:far,
            delete_element/3:far,
            delete_elements/3:far,
            equal/2:far,
            erase_term/2:far,
            ffactor/3:far,
            first_prod/2:far,
            flow_set/3:far,
            flow_set_c/3:far,
            get_split_flow/4:far,
            largest_flow/3:far,
            last_prod/2:far,
            list_member/2:far,
            max/3:far,
            max_flist/2:far,
            max_flow/3:far,
            member/2:far,
            pause/0:far,
            pause_escape/0:far,
            positive_component_flow_list/2:far,
            product_flow/3:far,
            product_flow_match/4:far,
            product_flow_set/3:far,
            reverse/2:far,
            selected_positive_component_flow_list/3:far,
            set_above/3:far,
            set_below/3:far,
            split_above/4:far,
            split_below/4:far,
            sub_list/2:far,
            sum_component_flow/3:far,
            sum_flist/2:far,
            two_highest_flows/3:far.


separation_specification_table(List,CList):-
            eraseall(table),
            erase_term(sst,sloppy_keys(_,_)),
            erase_term(sst,process_product(_)),
        current_window(_,six),
        cls,
        write('        Calculating Separation Specification Table... '),nl,
        initialize_keys(CList,CList),
        sharp_split_sst_driver(List,CList),
        initialize_products(List),
        sloppy_split_sst_driver(List,CList),
        write(' SST is complete. Enter choice for display '),
        send_menu_msg(activate(display,(14,30)),Choicesst),
        check_print_sst(Choicesst),
        sloppy_split_mk_sst_driver(List,CList),
        cls,
        current_window(_,one),!.

initialize_keys(L,[_|Tail]):-
        !,initialize_keys_help(L,Tail).

    initialize_keys_help(_,[]):-!.
```

```
    initialize_keys_help([H1|T1],[H2|T2]):-
        !,recordz(sst,process_keys(H1,H2),_),
        initialize_keys_help(T1,T2).

/*************************************************************
 * Heuristic M1: Favor normal distillation and avoid MSA's.  *
 * However, if the relative volatility between the LK and HK *
 * components is < 1.05, do not use distillation. This clause*
 * records 'extractive distillation' for status if the       *
 * relative volatility is < 1.05.                            *
 *************************************************************/

    sharp_split_sst_driver(List,CList):-
            recorded(sst,process_keys(LK,HK),Ref),
            sharp_split_sst(List,CList,LK,HK),
            erase(Ref),
            fail.
    sharp_split_sst_driver(_,_):- !.

    sharp_split_sst(List,CList,LK,HK):-
            recorded(data,k_value(LK,KLK),_),
            recorded(data,k_value(HK,KHK),_),
            Alpha is KLK/KHK,
            Alpha =< 1.10,
            split_above(CList,LK,TopComponents,BotComponents),
            sharp_split_dandb(CList,TopComponents,BotComponents,Listofd,Listofb),
            get_delta_temp(LK,HK,Delta),
            recordz(table,ddsst(List,CList,'sharp',TopComponents,BotComponents,LK,HK,Delta,Listofd,Listofb,'extractive
distillation','neither',0),_),
        !.

    sharp_split_sst(List,CList,LK,HK):-
        get_delta_temp(LK,HK,Delta),
        split_above(CList,LK,TopComponents,BotComponents),
        get_split_flow(List,TopCompFlow,TopComponents,0),
        get_split_flow(List,BotCompFlow,BotComponents,0),
        ffactor(TopCompFlow,BotCompFlow,FFactor),
        CES is FFactor*Delta/log(2401),
        sharp_split_dandb(CList,TopComponents,BotComponents,Listofd,Listofb),
        decide_difficult_or_easy(Delta,Ease),

recordz(table,ddsst(List,CList,'sharp',TopComponents,BotComponents,LK,HK,Delta,Listofd,Listofb,'feasible',Ease,CES),_),
            !.

    get_delta_temp(LK,HK,Delta):-
        recorded(data,boiling_temp(LK,LKT),_),
        recorded(data,boiling_temp(HK,HKT),_),
        Delta is abs(HKT-LKT),!.

    sharp_split_dandb([],_,_,[],[]):-!.
    sharp_split_dandb([Head1|Tail1],TopComponents,BotComponents,[HeadD|TailD],[HeadB|TailB]):-
        member(Head1,TopComponents),!,
        HeadD is 0.98,
        HeadB is 0.02,
            sharp_split_dandb(Tail1,TopComponents,BotComponents,TailD,TailB).

    sharp_split_dandb([Head1|Tail1],TopComponents,BotComponents,[HeadD|TailD],[HeadB|TailB]):-
        member(Head1,BotComponents),!,
        HeadD is 0.02,
        HeadB is 0.98,
            sharp_split_dandb(Tail1,TopComponents,BotComponents,TailD,TailB).
```

```
initialize_products([_]):-!.
initialize_products([H|T]):-
    recordz(sst,process_product(H),_),
    !,initialize_products(T).

sloppy_split_sst_driver(List,CList):-
    recorded(sst,process_product(Prod),Ref),
    sloppy_split_sst(List,Prod,CList),
        erase(Ref),
    fail.
sloppy_split_sst_driver(_,_):-!.

sloppy_split_mk_sst_driver(List,CList):-
    length(CList,X),
    X >= 3,
    repeat,
    finish_mk_analysis(List,CList).
sloppy_split_mk_sst_driver(_,_):-!.

sloppy_split_sst(List,Prod,CList):-
    split_above(List,Prod,Top,Bot),
        component_recovery_specification_test(Top,Bot,CList,Status),
      here,
    Status == 'infeasible',
    component_recovery_dandb(CList,Top,Bot,Listofd,Listofb),
      here,
        recordz(table,ddsst(List,CList,'sloppy',Top,Bot,'any','any',0,Listofd,Listofb,'infeasible','difficult',0),_),!.

sloppy_split_sst(List,Prod,CList):-
    split_above(List,Prod,Top,Bot), % Component recovery specification
                                    % test is feasible from above
    determine_potential_LKs(Top,Bot,CList,CList,ListofLKs),
    not(equal(ListofLKs,[])),
      here,
    initialize_keys_for_nonkey_test(ListofLKs,CList),
      here,
    nonkey_component_distribution_test_driver(List,CList,Top,Bot),!.

/*************************************************************
 * This third sloppy_split_sst clause is for when the list *
 * of light keys (ListofLKs) is nil. In this case, all the *
 * sloppy splits are in actuality sharp. Thus there's no   *
 * true sloppy split that has a LK/HK.                     *
 *************************************************************/

sloppy_split_sst(_,_,_):-!.

finish_mk_analysis(List,CList):-
    cls,
    write(' Do you wish to do a split key analysis? '),
        send_menu_msg(activate(no_yes,(15,32)),Decision),
        Decision==yes,
    !,
    initialize_products(List),
    input_split_keys(CList,CList,LK,HK),
    sloppy_split_mk_analysis(List,CList,LK,HK).
finish_mk_analysis(_,_).

input_split_keys([LK,_,HK],_,LK,HK):-!.
input_split_keys(KeyList,LK_List,LK,HK):-
    write(' Here is a list of potential LKs.'),nl,
```

```
        write(LK_List),nl,
        repeat,
        write('Enter choice for LK: '),
        read(LK),
        check_member(LK,LK_List),
        write('   Here is a list of potential HKs.'),nl,
        set_below(KeyList,LK,HK_List),
        write(HK_List),nl,
        repeat,
        write('Enter choice for HK: '),
        read(HK),
        check_member(HK,HK_List),!.

    check_member(X,XList):-
        member(X,XList).
    check_member(_,_):-
        write('Proposed component is not in stream. '),
        fail.

    sloppy_split_mk_analysis(List,CList,LK,HK):-
        recorded(sst,process_product(Prod),Ref),
        split_above(List,Prod,Top,Bot),
        nonkey_component_distribution_test_for_mks(List,CList,Top,Bot,LK,HK),
            erase(Ref),
        fail.
    sloppy_split_mk_analysis(_,_,_,_):-
        write(' SST is complete for split key analysis. Enter choice for display '),
        send_menu_msg(activate(display,(14,30)),Choicesst),
        check_print_sst(Choicesst),
        fail.

    initialize_keys_for_nonkey_test([],_):-!.
    initialize_keys_for_nonkey_test([H|T],CList):-
        shorten_list(H,CList,NCList),
        initialize_keys_for_nonkey_test_help([H|T],NCList).

    shorten_list(X,[X|T],T):-!.
    shorten_list(X,[_|T],List):-
        !,shorten_list(X,T,List).

    initialize_keys_for_nonkey_test_help([],_):-!.
    initialize_keys_for_nonkey_test_help([H1|T1],[H2|T2]):-
        !,recordz(sst,sloppy_keys(H1,H2),_),
        initialize_keys_for_nonkey_test_help(T1,T2).

component_recovery_specification_test(Top,Bot,[Head|Tail],Status):-
        component_flow(Bot,Head,BotFlow),
        component_flow(Top,Head,TopFlow),
        0 < (TopFlow + BotFlow),
        D is TopFlow/(TopFlow + BotFlow),
        B is BotFlow/(TopFlow + BotFlow),
        component_recovery_run(Top,Bot,Tail,D,B,Status),
        !.

    component_recovery_run(_,_,[],_,_,'feasible').
    component_recovery_run(Top,Bot,[Head|Tail],D,B,Status):-
        component_flow(Top,Head,TopFlow),
        component_flow(Bot,Head,BotFlow),
        Total is TopFlow + BotFlow,
        Total > 0,
        NewD is TopFlow/Total,
```

```
    D >= NewD,
    NewB is BotFlow/Total,
    B =< NewB,
    component_recovery_run(Top,Bot,Tail,NewD,NewB,Status).
component_recovery_run(Top,Bot,[Head|_],D,_,Status):-
    component_flow(Top,Head,TopFlow),
    component_flow(Bot,Head,BotFlow),
    Total is TopFlow + BotFlow,
    Total > 0,
    NewD is TopFlow/Total,
    D < NewD,
    Status == 'infeasible'.
component_recovery_run(Top,Bot,[Head|_],_,B,Status):-
    component_flow(Top,Head,TopFlow),
    component_flow(Bot,Head,BotFlow),
    Total is TopFlow + BotFlow,
    Total > 0,
    NewB is BotFlow/Total,
    B > NewB,
    Status == 'infeasible'.

determine_potential_LKs(_,_,_,[],[]).

determine_potential_LKs(Top,Bot,CList,[Head1|_],[Head2|Tail2]):-
    component_flow(Top,Head1,TopFlow),
    TopFlow > 0,
    component_flow(Bot,Head1,BotFlow),
    BotFlow > 0,
    Ratio is TopFlow/BotFlow,
    Ratio < 49.0,
    Ratio > 0.02040816327,   % Head1 is a distributed component
    split_below(CList,Head1,ComponentsAbove,ComponentsBelowIncludingHead1),
    last_prod(ComponentsAbove,PotentialLK),
    not(equal(PotentialLK,[])),  % a component exists above Head1 that is non-distributed
    append([],PotentialLK,[Head2]),    % This is the most volatile LK
        determine_remaining_LKs(Top,Bot,CList,ComponentsBelowIncludingHead1,Tail2).

determine_potential_LKs(Top,Bot,CList,[Head1|_],[Head2|Tail2]):-
    component_flow(Top,Head1,TopFlow),
    TopFlow > 0,
    component_flow(Bot,Head1,BotFlow),
    BotFlow > 0,
    Ratio is TopFlow/BotFlow,
    Ratio < 49.0,
    Ratio > 0.02040816327,   % Head1 is a distributed component
    split_below(CList,Head1,ComponentsAbove,ComponentsBelowIncludingHead1),
    last_prod(ComponentsAbove,PotentialLK),
    equal(PotentialLK,[]),  % no components exist above Head1; Head1 is the most volatile LK
    append([],[Head1],[Head2]),    % This is the most volatile LK
    delete_element(Head1,ComponentsBelowIncludingHead1,ComponentsBelow),
        determine_remaining_LKs(Top,Bot,CList,ComponentsBelow,Tail2).

determine_potential_LKs(Top,Bot,CList,[_|Tail1],TempListLKs):-
        determine_potential_LKs(Top,Bot,CList,Tail1,TempListLKs).

determine_remaining_LKs(_,_,_,[_],Tail2):-
    Tail2 = [].
determine_remaining_LKs(Top,Bot,CList,[H1|T1],[H2|Tail2]):-
    component_flow(Top,H1,TopFlow),
    TopFlow > 0,
    component_flow(Bot,H1,BotFlow),
```

```
    BotFlow > 0,
    Ratio is TopFlow/BotFlow,
    Ratio < 49.0,
    Ratio > 0.02040816327,  % H1 is a distributed component
    first_prod(T1,PotentialHK),
    not(equal(PotentialHK,[])),  % a HK does exist
        H2 = H1,      % This is a LK
    determine_remaining_LKs(Top,Bot,CList,T1,Tail2).
determine_remaining_LKs(Top,Bot,CList,[_|T1],Tail2):-
    determine_remaining_LKs(Top,Bot,CList,T1,Tail2).

nonkey_component_distribution_test_driver(List,CList,Top,Bot):-
    recorded(sst,sloppy_keys(LK,HK),Ref),
    nonkey_component_distribution_test(List,CList,Top,Bot,LK,HK),
    erase(Ref),
    fail.
nonkey_component_distribution_test_driver(_,_,_,_):-!.

nonkey_component_distribution_test(List,CList,Top,Bot,LK,HK):-
    calc_min_stages(Top,Bot,LK,HK,Nmin,LKdoverb,HKdoverb),
    solve_fenske_light_driver(CList,Top,Bot,LK,HK,Nmin,LLKdoverbList,LStatus),
    solve_fenske_heavy_driver(CList,Top,Bot,LK,HK,Nmin,HHKdoverbList,HStatus),
    append_v(LLKdoverbList,[LKdoverb,HKdoverb|HHKdoverbList],DoverBList),
    sloppy_split_dandb(DoverBList,Listofd,Listofb),
    assess_feasibility(LStatus,HStatus,Status),
        calculate_ces_and_split_ease(Top,Bot,LK,HK,CList,Status,Delta,CES,Ease),
      here,
        recordz(table,ddsst(List,CList,'sloppy',Top,Bot,LK,HK,Delta,Listofd,Listofb,Status,Ease,CES),_),!.

nonkey_component_distribution_test_for_mks(List,CList,Top,Bot,LK,HK):-
    calc_min_stages(Top,Bot,LK,HK,Nmin,LKdoverb,HKdoverb),
    solve_fenske_light_driver(CList,Top,Bot,LK,HK,Nmin,LLKdoverbList,LStatus),
    solve_fenske_middle_driver(CList,Top,Bot,LK,HK,Nmin,MKdoverbList,MStatus),
    solve_fenske_heavy_driver(CList,Top,Bot,LK,HK,Nmin,HHKdoverbList,HStatus),
    append_v(LLKdoverbList,[LKdoverb|MKdoverbList],IDoverBList),
    append_v(IDoverBList,[HKdoverb|HHKdoverbList],DoverBList),
    sloppy_split_dandb(DoverBList,Listofd,Listofb),
    assess_feasibility(LStatus,HStatus,IStatus),
    assess_feasibility(IStatus,MStatus,Status),
        calculate_ces_and_split_ease(Top,Bot,LK,HK,CList,Status,Delta,CES,Ease),
      here,
        recordz(table,ddsst(List,CList,'sloppy',Top,Bot,LK,HK,Delta,Listofd,Listofb,Status,Ease,CES),_),!.

calc_min_stages(Top,Bot,LK,HK,Nmin,Ratio1,Ratio3):-
    component_flow(Top,LK,TFlow1),
    TFlow1 > 0,
    component_flow(Bot,LK,BFlow1),
    component_flow(Top,HK,TFlow2),
    component_flow(Bot,HK,BFlow2),
    BFlow2 > 0,
    recorded(data,k_value(LK,KLK),_),
    recorded(data,k_value(HK,KHK),_),
    Alpha is KLK/KHK,
    solve_ratio(TFlow1,BFlow1,Ratio1),
    solve_ratio(BFlow2,TFlow2,Ratio2),
    Ratio3 is 1/Ratio2,
    Nmin is log(Ratio1*Ratio2)/log(Alpha),!.
calc_min_stages(Top,_,LK,_,Nmin,0,0):-
    component_flow(Top,LK,TFlow1),
    TFlow1 =< 0,
    Nmin is 0.
```

```
    calc_min_stages(_,Bot,_,HK,Nmin,0,0):-
        component_flow(Bot,HK,BFlow2),
        BFlow2 =< 0,
        Nmin is 0.

/*************************************************************
 * Fenske equation is feasible on the LLK side if no LLK *
 * components exist.                                     *
 *************************************************************/

    solve_fenske_light_driver(CList,_,_,LK,_,_,[],'feasible'):-
        set_above(CList,LK,LLKList),
        LLKList == [],!.

/*************************************************************
 * If LLK components do exist, a Fenske feasibility      *
 * analysis is required.                                 *
 *************************************************************/

    solve_fenske_light_driver(CList,Top,Bot,LK,HK,Nmin,LLKdoverbList,LStatus):-
        set_above(CList,LK,LLKList),
        component_flow(Top,HK,TFLow1),
        component_flow(Bot,HK,BFLow1),
        solve_ratio(TFLow1,BFLow1,Ratio1),  % Ratio1 is (d/b)HK
        solve_fenske_light(HK,LLKList,Ratio1,LLKdoverbList,Nmin),
        test_LLK_feasibility(Top,Bot,LLKList,LLKdoverbList,LStatus).

/*************************************************************
 * The solve_fenske_light clause determines (d/b) for all *
 * LLK components.                                       *
 *************************************************************/

    solve_fenske_light(_,[],_,[],_).
    solve_fenske_light(HK,[LLK|Tailc],Ratio1,[LLKdoverb|Tailr],Nmin):-
        recorded(data,k_value(HK,KHK),_),
        recorded(data,k_value(LLK,KLLK),_),
        Alpha is KLLK/KHK,
        E is exp(1),
        Constant is Nmin*log(Alpha)/log(E),
        LLKdoverb is Ratio1*exp(Constant),
        solve_fenske_light(HK,Tailc,Ratio1,Tailr,Nmin).

/*************************************************************
 * The test_LLK_feasibility clause has four cases:       *
 * First Clause: the base case; if it gets here, the split*
 *   is feasible.                                        *
 * Second Clause: sharp split desired on LLK; split is   *
 *   is feasible if (d/b)LLK >= 0.93/0.07                *
 * Third Clause: nonsharp split desired on LLK; split is *
 *   feasible if (d/b)LLK is within +/- 20% of desired   *
 * Fourth Clause: split is infeasible if it fails clauses *
 *   two and three above. No more recursion needed.      *
 *************************************************************/

    test_LLK_feasibility(_,_,[],[],'feasible'):-!.
    test_LLK_feasibility(Top,Bot,[HC|TC],[HR|TR],LStatus):-
        component_flow(Top,HC,TFLow),
        component_flow(Bot,HC,BFLow),
        solve_ratio(TFLow,BFLow,Desireddoverb),
        Desireddoverb >= 49.0,
        HR >= 13.28571429,
```

```
        test_LLK_feasibility(Top,Bot,TC,TR,LStatus).
    test_LLK_feasibility(Top,Bot,[HC|TC],[HR|TR],LStatus):-
        component_flow(Top,HC,TFLow),
        component_flow(Bot,HC,BFLow),
        solve_ratio(TFLow,BFLow,Desireddoverb),
        Lowerbound is 0.8*Desireddoverb,
        Lowerbound =< HR,
        Upperbound is 1.2*Desireddoverb,
        Upperbound >= HR,
        test_LLK_feasibility(Top,Bot,TC,TR,LStatus).
    test_LLK_feasibility(_,_,_,_,'infeasible'):-!.

/************************************************************
 * A Fenske feasibility analysis is required for            *
 * MK components.                                           *
 ************************************************************/

    solve_fenske_middle_driver(CList,Top,Bot,LK,HK,Nmin,MKdoverbList,MStatus):-
        set_above(CList,HK,LK_and_MKList),
        set_below(LK_and_MKList,LK,MKList),
        component_flow(Top,HK,TFLow1),
        component_flow(Bot,HK,BFLow1),
        solve_ratio(TFLow1,BFLow1,Ratio1),  % Ratio1 is (d/b)HK
        solve_fenske_middle(HK,MKList,Ratio1,MKdoverbList,Nmin),
        test_MK_feasibility(Top,Bot,MKList,MKdoverbList,MStatus).

/************************************************************
 * The solve_fenske_middle clause determines (d/b) for all* *
 * MK components.                                           *
 ************************************************************/

    solve_fenske_middle(_,[],_,[],_).
    solve_fenske_middle(HK,[MK|Tailc],Ratio1,[MKdoverb|Tailr],Nmin):-
        recorded(data,k_value(HK,KHK),_),
        recorded(data,k_value(MK,KMK),_),
        Alpha is KMK/KHK,
        E is exp(1),
        Constant is Nmin*log(Alpha)/log(E),
        MKdoverb is Ratio1*exp(Constant),
        solve_fenske_middle(HK,Tailc,Ratio1,Tailr,Nmin).

/************************************************************
 * The test_MK_feasibility clause has four cases:           *
 * First Clause: the base case; if it gets here, the split* *
 *   is feasible.                                           *
 * Second Clause: sharp split desired on MK; split          *
 *   is feasible if (d/b)MK >= 0.93/0.07                    *
 * Third Clause: nonsharp split desired on MK; split is     *
 *   feasible if (d/b)MK is within +/- 20% of desired       *
 * Fourth Clause: split is infeasible if it fails clauses * *
 *   two and three above. No more recursion needed.         *
 ************************************************************/

    test_MK_feasibility(_,_,[],[],'feasible'):-!.
    test_MK_feasibility(Top,Bot,[HC|TC],[HR|TR],MStatus):-
        component_flow(Top,HC,TFLow),
        component_flow(Bot,HC,BFLow),
        solve_ratio(TFLow,BFLow,Desireddoverb),
        Desireddoverb >= 49.0,
        HR >= 13.28571429,
        test_MK_feasibility(Top,Bot,TC,TR,MStatus).
```

```
test_MK_feasibility(Top,Bot,[HC|TC],[HR|TR],MStatus):-
    component_flow(Top,HC,TFLow),
    component_flow(Bot,HC,BFLow),
    solve_ratio(TFLow,BFLow,Desireddoverb),
    Lowerbound is 0.8*Desireddoverb,
    Lowerbound =< HR,
    Upperbound is 1.2*Desireddoverb,
    Upperbound >= HR,
    test_MK_feasibility(Top,Bot,TC,TR,MStatus).
test_MK_feasibility(_,_,_,_,'infeasible'):-!.

/*************************************************************
 * Fenske equation is feasible on the HHK side if no HHK *
 * components exist.                                     *
 *************************************************************/


solve_fenske_heavy_driver(CList,_,_,_,HK,_,[],'feasible'):-
    set_below(CList,HK,HHKList),
    HHKList == [],!.


/*************************************************************
 * If HHK components do exist, a Fenske feasibility      *
 * analysis is required.                                 *
 *************************************************************/



solve_fenske_heavy_driver(CList,Top,Bot,LK,HK,Nmin,HHKdoverbList,HStatus):-
    set_below(CList,HK,HHKList),
    component_flow(Top,LK,TFLow1),
    component_flow(Bot,LK,BFLow1),
    solve_ratio(BFLow1,TFLow1,Ratio1),  % Ratio1 is (b/d)LK
    solve_fenske_heavy(LK,HHKList,Ratio1,HHKdoverbList,Nmin),
    test_HHK_feasibility(Top,Bot,HHKList,HHKdoverbList,HStatus).


/*************************************************************
 * The solve_fenske_heavy clause determines (d/b) for all *
 * HHK components.                                       *
 *************************************************************/


solve_fenske_heavy(_,[],_,[],_).
solve_fenske_heavy(LK,[HHK|Tailc],Ratio1,[HHKdoverb|Tailr],Nmin):-
    recorded(data,k_value(LK,KLK),_),
        recorded(data,k_value(HHK,KHHK),_),
    Alpha is KLK/KHHK,
    E is exp(1),
    Constant is Nmin*log(Alpha)/log(E),
    HHKdoverb is 1/(Ratio1*exp(Constant)),
    solve_fenske_heavy(LK,Tailc,Ratio1,Tailr,Nmin).


/*************************************************************
 * The test_HHK_feasibility clause has four cases:       *
 * First Clause: the base case; if it gets here, the split*
 *   is feasible.                                        *
 * Second Clause: sharp split desired on HHK; split is   *
 *   is feasible if (b/d)HHK >= 0.93/0.07                *
```

```
* Third Clause: nonsharp split desired on HHK; split is  *
*   feasible if (b/d)HHK is within +/- 20% of desired    *
* Fourth Clause: split is infeasible if it fails clauses *
*   two and three above. No more recursion needed.       *
*********************************************************/


    test_HHK_feasibility(_,_,[],[],'feasible'):-!.
    test_HHK_feasibility(Top,Bot,[HC|TC],[HR|TR],HStatus):-
        component_flow(Top,HC,TFLow),
        component_flow(Bot,HC,BFLow),
        solve_ratio(BFLow,TFLow,Desiredboverd),
        Desiredboverd >= 49.0,
        Calcboverd is 1/HR,
        Calcboverd >= 13.28571429,
        test_HHK_feasibility(Top,Bot,TC,TR,HStatus).
    test_HHK_feasibility(Top,Bot,[HC|TC],[HR|TR],HStatus):-
        component_flow(Top,HC,TFLow),
        component_flow(Bot,HC,BFLow),
        solve_ratio(BFLow,TFLow,Desiredboverd),
        Calcboverd is 1/HR,
        Lowerbound is 0.8*Desiredboverd,
        Lowerbound =< Calcboverd,
        Upperbound is 1.2*Desiredboverd,
        Upperbound >= Calcboverd,
        test_HHK_feasibility(Top,Bot,TC,TR,HStatus).
    test_HHK_feasibility(_,_,_,_,'infeasible'):-!.

    component_recovery_dandb([],_,_,[],[]).
    component_recovery_dandb([HC|TC],Top,Bot,[HD|TD],[HB|TB]):-
        component_flow(Top,HC,TFLow),
        component_flow(Bot,HC,BFlow),
        solve_ratio(TFlow,BFlow,Ratio),
        HB is 1/(1+Ratio),
        HD is 1 - HB,
        component_recovery_dandb(TC,Top,Bot,TD,TB).

/*********************************************************
* The sloppy_split_dandb clause is a utility that       *
* separates the numeric value of d/b into the fractional *
* recoveries of d and b such that:                      *
*       0=< d =< 1  and 0 =< b =< 1.                    *
*********************************************************/


    sloppy_split_dandb([],[],[]):-!.
    sloppy_split_dandb([H|T],[HD|TD],[HB|TB]):-
        HD is H/(1+H),
        HB is 1 - HD,
        sloppy_split_dandb(T,TD,TB).

    assess_feasibility('feasible','feasible','feasible').
    assess_feasibility(_,_,'infeasible').


    calculate_ces_and_split_ease(_,_,LK,HK,_,'infeasible',Delta,0,'difficult'):-
        recorded(data,boiling_temp(LK,LKTemp),_),
        recorded(data,boiling_temp(HK,HKTemp),_),
        Delta is abs(HKTemp-LKTemp).

    calculate_ces_and_split_ease(Top,Bot,LK,HK,CList,'feasible',Delta,CES,Ease):-
```

```
        ces_log_term(Top,Bot,LK,HK,INVLTerm),
        calculate_ces(Top,Bot,LK,HK,INVLTerm,Delta,CES,CList),
        decide_difficult_or_easy(Delta,Ease).


    /*  CES Log terms  */

/* 0
        0 0
        0 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,LK,BFlow2),
        BFlow2 == 0,
        INVLTerm is 0.

/* 1
        a 0
        0 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 == 0,
        INVLTerm is 0,
        write('Must transpose CAM, EXIT'),
        pause,halt.

/* 2
        0 b
        0 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
            BFlow2 == 0,
        INVLTerm is 0.

/* 3
        0 0
        0 b
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
```

```
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        INVLTerm is 0,
        write('Must transpose CAM, EXIT'),
        pause,halt.

/* 4
        0 0
        a 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 == 0,
        INVLTerm is 0.

/* 5
        a b
        0 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 == 0,
        INVLTerm is 0.

/* 6
        a 0
        0 b
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        Ratio is 2401,
        INVLTerm is 1/log(Ratio).

/* 7
        a 0
        a 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
```

```
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 == 0,
        INVLTerm is 0.

/* 8
        0 b
        0 b
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        INVLTerm is 0.

/* 9
        0 b
        a 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 == 0,
        INVLTerm is 0,
        write('Must transpose CAM, EXIT'),
        pause,halt.

/* 10
        0 0
        a b
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        INVLTerm is 0.

/* 11
        a b
        0 b
*/
```

```
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 == 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        Ratio is 49*BFlow2/TFlow2,
        INVLTerm is 1/log(Ratio).

/* 12
        0 b
        a b
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 == 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        INVLTerm is 0,
        write('Must transpose CAM, EXIT'),
        pause,halt.

/* 13
        a b
        a 0
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 == 0,
        INVLTerm is 0,
        write('Must transpose CAM, EXIT'),
        pause,halt.

/* 14
        a 0
        a b
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 == 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        Ratio is 49*TFlow1/BFlow1,
        INVLTerm is 1/log(Ratio).
```

```
/* 15
        a b
        a b
*/
    ces_log_term(Top,Bot,LK,HK,INVLTerm):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,LK,BFlow1),
        BFlow1 > 0,
        component_flow(Top,HK,TFlow2),
        TFlow2 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        Ratio is TFlow1*BFlow2/BFlow1/TFlow2,
        INVLTerm is 1/log(Ratio).

    calculate_ces(Top,Bot,LK,HK,INVLTerm,Delta,CES,CList):-
        component_flow(Top,LK,TFlow1),
        TFlow1 > 0,
        component_flow(Bot,HK,BFlow2),
        BFlow2 > 0,
        get_split_flow(Top,TopFlow,CList,0),
        get_split_flow(Bot,BotFlow,CList,0),
        recorded(data,boiling_temp(LK,LKTemp),_),
        recorded(data,boiling_temp(HK,HKTemp),_),
        Delta is abs(HKTemp-LKTemp),
        ffactor(TopFlow,BotFlow,FFactor),
        CES is FFactor*Delta*INVLTerm.
    calculate_ces(Top,_,LK,_,_,Delta,CES,_):-
        component_flow(Top,LK,TFlow1),
        TFlow1 =< 0,
        Delta is 0,
        CES is 0.
    calculate_ces(_,Bot,_,HK,_,Delta,CES,_):-
        component_flow(Bot,HK,BFlow1),
        BFlow1 =< 0,
        Delta is 0,
        CES is 0.

    check_print_sst(donotdisplay):-
        transfer_database.
    check_print_sst(print):-
            tell(lpt1),
        print_sst('P'),
            told.
    check_print_sst(displaytoscreen):-
        cls,
        print_sst('S').

print_sst('P'):-
            recorded(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),Ref),
            write('Split: '),
            tab(3),write(Top),write('/'),write(Bot),
            write('  LK/HK: '),write(LK),write('/'),write(HK),
            write('   Status: '),write(Feasibility),
            nl,
       print_doverb(Listofd,Listofb,CList),
            [! recorded(data,k_value(LK,KLK),_),
       recorded(data,k_value(HK,KHK),_) !],
       Alpha is KLK/KHK,
       write('LK/HK Temperature Difference (C):'),
```

```
            WDelta is round(Delta,2),write(WDelta),
            write(' CES:'),WCES is round(CES,2),write(WCES),
            write(' Alpha(LK,HK):'),WAlpha is round(Alpha,2),write(WAlpha),
            nl,
        write('------------------------------------------------------------------------'),nl,
        erase(Ref),
            recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),_),
        fail.
print_sst('S'):-
        recorded(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),Ref),
            write('Split: '),
            tab(3),write(Top),write('/'),write(Bot),
            write('  LK/HK: '),write(LK),write('/'),write(HK),
            write('   Status: '),write(Feasibility),
            nl,
        print_doverb(Listofd,Listofb,CList),
            [! recorded(data,k_value(LK,KLK),_),
        recorded(data,k_value(HK,KHK),_) !],
        Alpha is KLK/KHK,
        write('LK/HK Temperature Difference (C):'),
            WDelta is round(Delta,2),write(WDelta),
            write(' CES:'),WCES is round(CES,2),write(WCES),
            write(' Alpha(LK,HK):'),WAlpha is round(Alpha,2),write(WAlpha),
            nl,
        write('------------------------------------------------------------'),nl,nl,
        erase(Ref),
            recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),_),
        current_window(_,six),
        pause,
        fail.
    print_sst(_).

print_doverb([],[],[]):- nl,!.
print_doverb([Headd|Taild],[Headb|Tailb],[HeadC|TailC]):-
        write('(d/b)'),
            write(HeadC),write(' '),
            Headdt is round(Headd,2),
            Headbt is round(Headb,2),
            write(Headdt),write('/'),write(Headbt),nl,
        print_doverb(Taild,Tailb,TailC).

    transfer_database:-
        recorded(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),Ref),
        erase(Ref),
        recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),_),
        fail.
    transfer_database.



solve_ratio(F1,F2,R):-
            F1>0,
            F2>0,
            R is F1/F2,!.
solve_ratio(F1,F2,R):-
            F2 == 0,
            F1>0,
            R is 49,!.
solve_ratio(F1,F2,R):-
            F1 == 0,
            F2>0,
```

R is 0.02040816327,!.

/* * * End of SST Calculations * * */

here.

SPLIT MODULE (SPLIT.ARI)

```
/*****************************************************************
 * SPLIT MODULE -- This module implements the rank-ordered      *
 * heuristics of Nadgir and Liu, and heuristically chooses a    *
 * split that has been previously generated and assessed as     *
 * being technically feasible.                                  *
 *****************************************************************/
:-segment(splitseg).
:-public split/6:far.
:-visible member/2,recorded/3,equal/2.
:-extrn
        append/3:far,
        append_v/3:far,
        component_flow/3:far,
        component_flow_list/4:far,
        component_flow_set/3:far,
        decide_difficult_or_easy/2:far,
        delete_element/3:far,
        delete_elements/3:far,
        equal/2:far,
        erase_term/2:far,
        ffactor/3:far,
        first_prod/2:far,
        flow_set/3:far,
        flow_set_c/3:far,
        get_split_flow/4:far,
        largest_flow/3:far,
        last_prod/2:far,
        list_member/2:far,
        max/3:far,
        max_flist/2:far,
        max_flow/3:far,
        member/2:far,
        pause/0:far,
        pause_escape/0:far,
        positive_component_flow_list/2:far,
        product_flow/3:far,
        product_flow_match/4:far,
        product_flow_set/3:far,
        reverse/2:far,
        selected_positive_component_flow_list/3:far,
        set_above/3:far,
        set_below/3:far,
        split_above/4:far,
        split_below/4:far,
        sub_list/2:far,
        sum_component_flow/3:far,
        sum_flist/2:far,
        two_highest_flows/3:far,
                solve_ratio/3:far.


/*****************************************************************
 * The split clause is the central control clause in the SPLIT  *
 * module. It implements heuristics S1, S2, C1, and C2, chooses*
 * the best split based on the heuristics, and determines the   *
 * overhead and bottoms streams for the splitter.               *
```

```
*****************************************************************/

/*****************************************************************
*        Base cases and trivial solutions                       *
*****************************************************************/

    split([],_,[],[],[],[]):-!.
    split([_],_,[],[],[],[]):-!.
    split(_,[],[],[],[],[]):-!.
    split(_,[_],[],[],[],[]):-!.

/*****************************************************************
*        Implementation of heuristics M1 and M2                 *
*****************************************************************/

    split(List,CList,_,_,_,_):-
            current_window(_,five),cls,
        write('*******************************************************************************'),nl,
        write('**            Heuristic M1: Favor ordinary distillation.            **'),nl,
        write('*******************************************************************************'),nl,
            recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,Listofd,Listofb,Delta,'extractive distillation',Ease,CES),Ref),
        recorded(data,k_value(LK,KLK),_),
        recorded(data,k_value(HK,KHK),_),
        Alpha = KLK/KHK,
        nl,write('Normal distillation for split '),
            write(Top),write('/'),write(Bot),nl,
        write('with '),write(LK),write('/'),write(HK),write(' as LK/HK'),nl,
        write('and alpha(LK,HK) = '),
        Alphat is round(Alpha,2),write(Alphat),write(' is not recommended.'),nl,
        write('Due to the low relative volatility, an alternate'),nl,
        write('separation process is recommended.'),nl,
            pause,
        erase(Ref),
        recordz(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Listofd,Listofb,Delta,'extractive distillation',Ease,CES),_),
        fail.

    split(_,_,_,_,_,_):-
            nl,write('Ordinary distillation is recommended for'),nl,
            write('remaining splits in the database'),
            pause,cls,
            fail.

    split(List,CList,_,_,_,_):-
        recorded(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Listofd,Listofb,Delta,'extractive distillation',Ease,CES),Ref),
        erase(Ref),
        recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Listofd,Listofb,Delta,'extractive distillation',Ease,CES),_),
        fail.

/*****************************************************************
*          Implementation of heuristic M2                       *
*****************************************************************/

/*  split(List,CLIst,_,_,_,_):-
        cls,
        bubble_point,
        dew_point,
        ... not yet implemented ...*/

/*****************************************************************
*        Implementation of heuristic S1                         *
*****************************************************************/
```

```
split(List,[H|T],TopList,TopCList,BotList,BotCList):-
    cls,
    write('**************************************************************************'),nl,
    write('** Heuristic S1: Remove corrosive and hazardous materials first.  **'),nl,
    write('**************************************************************************'),nl,nl,
    recorded(data,corrosive(Element),_),
    CList = [H|T],
    H == Element,
    recorded(table,dsst(List,CList,'sharp',Top,Bot,H,_,_,_,_,'feasible',_,_),Ref),
    write('Component '),write(Element),write(' is corrosive.'),nl,
    write('The recommended split is '),write(Top),write(' / '),write(Bot),
        write(' Do you wish to make this split? '),
    send_menu_msg(activate(split,(14,30)),Choicesplit),
    choose_split(Choicesplit),
    resulting_products(List,'sharp',Top,Bot,TopList,TopCList,BotList,BotCList),
    recorded(sep,dseparator(Num,_,_,_,_),_),
        NNum is Num +1,
    component_flow_set(TopList,TopCList,TopFList),
    component_flow_set(BotList,BotCList,BotFList),
    recorda(sep,dseparator(NNum,TopCList,TopFList,BotCList,BotFList),_),
    erase_several(dsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),
    erase_several(ddsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),!.

split(List,CList,TopList,TopCList,BotList,BotCList):-
    recorded(data,corrosive(Element),_),
    last_prod(CList,Bot),
    Bot = [H|_],
    H == Element,
    recorded(table,dsst(List,CList,'sharp',Top,Bot,_,H,_,_,_,'feasible',_,_),Ref),
    write('Component '),write(Element),write(' is corrosive.'),nl,
    write('The recommended split is '),write(Top),write(' / '),write(Bot),
        write(' Do you wish to make this split? '),
    send_menu_msg(activate(split,(14,30)),Choicesplit),
    choose_split(Choicesplit),
    resulting_products(List,'sharp',Top,Bot,TopList,TopCList,BotList,BotCList),
    recorded(sep,dseparator(Num,_,_,_,_),_),
    NNum is Num +1,
    component_flow_set(TopList,TopCList,TopFList),
    component_flow_set(BotList,BotCList,BotFList),
    recorda(sep,dseparator(NNum,TopCList,TopFList,BotCList,BotFList),_),
    erase_several(dsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),
    erase_several(ddsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),!.

split(_,[Head|Tail],_,_,_,_):-
    recorded(data,corrosive(Element),_),
    Element \= Head,
    member(Element,[Head|Tail]),
    last_prod([Head|Tail],Bot),
    Bot is [H|_],
    Element \= H,
    write('Component '),write(Element),write(' is corrosive.'),nl,
    write('However, it cannot be isolated in a sharp split.'),nl,
    write('It is recommended to proceed with other heuristics.'),
        cls,
        current_window(_,five),
        pause,
    fail.

split(_,CList,_,_,_,_):-
    recorded(data,corrosive(Element),_),
    not(member(Element,CList)),
```

```
        write('No corrosive elements exist.'),nl,
        write('It is recommended to apply other heuristics'),nl,
            pause,cls,
            current_window(_,five),
        fail.

/*******************************************************
* Heuristic S2: Perform difficult separations last.  *
* This has already been implented in the dsst state- *
* ment when constructing SST. It has the qualifier   *
* 'easy' or 'difficult' based on boiling point delta *
* less than 20 degrees C. These results are merely   *
* shown to the user below.                           *
*******************************************************/

    split(List,CList,_,_,_,_):-
            cls,
            current_window(_,five),
        write('*************************************************************************'),nl,
        write('**       Heuristic S2: Perform difficult separations last.       **'),nl,
        write('*************************************************************************'),nl,nl,
        recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','difficult',CES),Ref),
        erase(Ref),
        recordz(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','difficult',CES),_),
        write('Split '),write(Top),write('/'),write(Bot),write(' with '),
            write(LK),write('/'),write(HK),write(' as LK/HK'),nl,
        write('has a normal boiling point temperature difference of '),
            write(Delta),nl,
        write('and has been identified as an essential last split by heuristic S1.'),nl,nl,
        pause,
        fail.

    split(List,CList,_,_,_,_):-
        not(recorded(table,ddsst(List,CList,_,_,_,_,_,_,_,_,'feasible','difficult',_),Ref)),
        write('No splits are particularly difficult.'),nl,
        write('Heuristic S1 does not apply, and no splits are identified as essential'),nl,
            write(' last splits.'),nl,
            current_window(_,five),
            pause,
        fail.

    split(List,CList,_,_,_,_):-
        recorded(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','difficult',CES),Ref),
        erase(Ref),
        recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','difficult',CES),_),
        fail.

/*******************************************************
* Heuristic C1: Remove most plentiful product first. *
* This section retracts all splits from SST (with    *
* interactive user approval) if they do not remove   *
* the most plentiful product in full. After          *
* retraction, the clause fails to force splitting of *
* remaining splits via heuristic C2.                 *
*******************************************************/

    split(List,CList,_,_,_,_):-
        cls,
        write('*************************************************************************'),nl,
        write('**     Heuristic C1: Remove the most plentiful product first.     **'),nl,
        write('*************************************************************************'),nl,nl,
```

```
        product_flow_set(List,FList,CList),
    two_highest_flows(FList,MaxFlow,SecondFlow),
    MaxFlow/SecondFlow =< 1.2,
    write('No product is particularly dominant in '),nl,
    write('flow rate. Heuristic C1 does not apply.'),nl,
        pause,
        current_window(_,five),
    fail.

split(List,CList,_,_,_,_):-
    product_flow_set(List,FList,CList),
    two_highest_flows(FList,MaxFlow,SecondFlow),
        MaxFlow/SecondFlow > 1.2,
    sum_flist(FList,TFlow),
    MaxFlow/TFlow =< 0.285,
    write('No product is particularly dominant in '),nl,
    write('flow rate. Heuristic C1 does not apply.'),nl,
        pause,
        current_window(_,five),
    fail.

split(List,CList,_,_,_,_):-
    product_flow_set(List,FList,CList),
    two_highest_flows(FList,MaxFlow,SecondFlow),
        MaxFlow/SecondFlow > 1.2,
    sum_flist(FList,TFlow),
    MaxFlow/TFlow > 0.285,
    product_flow_match(List,MaxFlow,Prod,CList),
    check_sloppy_splits(List,List,CList,Prod),
    selected_positive_component_flow_list([Prod],CList,CompList),
    not(equal(CList,CompList)),
    check_sharp_splits(List,CList,CList,Prod),
    fail.

split(List,CList,_,_,_,_):-
    product_flow_set(List,FList,CList),
    two_highest_flows(FList,MaxFlow,SecondFlow),
        MaxFlow/SecondFlow > 1.2,
    sum_flist(FList,TFlow),
    MaxFlow/TFlow > 0.285,
        product_flow_match(List,MaxFlow,Prod,CList),
    selected_positive_component_flow_list([Prod],CList,CompList),
    equal(CList,CompList),
    write('Note: product '),write(Prod),write(' is the most plentiful.'),
        write('It is also all-component-inclusive.'),nl,
        write(' Therefore, heuristic C1 will not  be applied to sharp splits here.'),
        pause,
        current_window(_,five),
    fail.

split([H1|[A1]],[H2|[A2]],_,_,_,_):-
        diagonal_zeros(H1,A1,H2,A2),
        write('Only one sharp split remains.'),nl,
        write('Heuristic C1 does not apply.'),nl,
        pause,
        current_window(_,five),
        fail.

/*****************************************************************
 * Heuristic C2: Favor a 50/50 split. This split closest to   *
 * 50/50 will be chosen. If another split is within 30% of a  *
```

```
* 50/50 split, then the CES will also be used.                    *
*****************************************************************/

    split(List,CList,TopList,TopCList,BotList,BotCList):-
            cls,
        write('*************************************************************************'),nl,
        write('**                Heuristic C2: Favor a 50/50 Split              **'),nl,
        write('*************************************************************************'),nl,
        write('                          Alpha  Distillate  Bottoms  Molar Split       '),nl,
        write('   Split      LK    HK    LK,HK     Flow       Flow     Ratio    CES'),nl,
        write('-----------  ----  ----  ------  --------  --------  -----  -----'),nl,

determine_two_best_split_ratios(List,CList,Top1,Bot1,Type1,LK1,HK1,Ratio1,CES1,Top2,Bot2,Type2,LK2,HK2,Ratio2,CES2),
            Testratio is 1.3*Ratio1,
            determine_whether_to_use_CES(Testratio,Ratio2,Answer),

decide_split_choice(Top1,Bot1,Type1,LK1,HK1,Ratio1,CES1,Top2,Bot2,Type2,LK2,HK2,Ratio2,CES2,Top,Bot,Type,LK,HK,R
atio,CES,Answer),
        X is 100/(Ratio + 1),
        Y is 100 - X,
        max(X,Y,D),
        B is 100 - D,
        cls,
        write('  Split '),write(Top),write('/'),write(Bot),write(' with:'),nl,
        write('    -- components '),write(LK),write('/'),write(HK),write(' as LK/HK'),nl,
        write('    -- a CES = '),WCES is round(CES,2),write(WCES),nl,
        write('    -- a molar split ratio = '),WD is round(D,2),write(WD),
            write('/'),WB is round(B,2),write(WB),nl,
        write('is the recommended split. Do you wish to make this split? '),
        send_menu_msg(activate(split,(14,30)),Choicesplit),
        choose_split(Choicesplit),
        resulting_products(List,Type,Top,Bot,TopList,TopCList,BotList,BotCList),
        recorded(sep,dseparator(Num,_,_,_,_),_),
        NNum is Num +1,
        component_flow_set(TopList,TopCList,TopFList),
        component_flow_set(BotList,BotCList,BotFList),
        recorda(sep,dseparator(NNum,TopCList,TopFList,BotCList,BotFList),_),
        erase_several(dsst(List,CList,_,_,_,_,_,_,_,_,_,_,_,_,_)),
        erase_several(ddsst(List,CList,_,_,_,_,_,_,_,_,_,_,_,_,_)),!.

/*************************************************************
* Heuristic S2: Do difficult splits last. Difficult     *
* separations are in dsst statement. This section splits*
* difficult separations after other splits are complete.*
* If more than one difficult separation exist, then the *
* one with the maximum CES is chosen first.             *
*************************************************************/

    split(List,CList,TopList,TopCList,BotList,BotCList):-
        max_ces_sloppy(List,List,CList,0,CESsloppy,'difficult'),
        max_ces_sharp(List,CList,CList,0,CESsharp,'difficult'),
        max(CESsharp,CESsloppy,CES),
        CES > 0.0,
        recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,_,_,_,'feasible','difficult',CES),Ref),
        write('Difficult separation'),nl,
        write('Split '),write(Top),write('/'),write(Bot),nl,
        write('with '),write(LK),write('/'),write(HK),write(' as LK/HK'),nl,
        write('and CES = '),write(CES),nl,
        write('is the recommended split. Do you  wish to make this split? '),
        send_menu_msg(activate(split,(14,30)),Choicesplit),
        choose_split(Choicesplit),
```

```
        resulting_products(List,Type,Top,Bot,TopList,TopCList,BotList,BotCList),
        recorded(sep,dseparator(Num,_,_,_,_),_),
        NNum is Num +1,
        component_flow_set(TopList,TopCList,TopFList),
        component_flow_set(BotList,BotCList,BotFList),
        recorda(sep,dseparator(NNum,TopCList,TopFList,BotCList,BotFList),_),
        erase_several(dsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),
        erase_several(ddsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),!.

/*    This section is if no splits above were executed.  */

    split(List,CList,TopList,TopCList,BotList,BotCList):-
        cls,
        write('Heuristics complete. No splits have yet been chosen. Heuristic search can be reinitiated or a split can be chosen
manually. Enter your choice. '),
        send_menu_msg(activate(technique,(14,30)),Choicesplit),
        decide_split_path(Choicesplit),
        recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible',Ease,CES),Ref),
        erase(Ref),
        recordz(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible',Ease,CES),_),
        write(' If desired, '),write(Type),write(' split'),
            write(Top),write('/'),write(Bot),write(' with a CES = '),
        WCES is round(CES,2),write(WCES),
        write(' can be done. Please enter your choice. '),
        send_menu_msg(activate(split,(14,30)),SecondChoicesplit),
        choose_split(SecondChoicesplit),
        resulting_products(List,Type,Top,Bot,TopList,TopCList,BotList,BotCList),
        recorded(sep,dseparator(Num,_,_,_,_),_),
        NNum is Num +1,
        component_flow_set(TopList,TopCList,TopFList),
        component_flow_set(BotList,BotCList,BotFList),
        recorda(sep,dseparator(NNum,TopCList,TopFList,BotCList,BotFList),_),
        erase_several(dsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),
        erase_several(ddsst(List,CList,_,_,_,_,_,_,_,_,_,_,_)),!.

/*    This section is if no manual split is chosen. The *
 *    above rule would then fail                        */

    split(List,CList,_,_,_,_):-
        cls,
        max_flist([X],X).
            write('  No splits were chosen manually. The heuristic search will be reinitiated... '),
        recorded(table,ddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible',Ease,CES),Ref),
        erase(Ref),
        recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible',Ease,CES),_),
        fail.

    split(List,CList,TopList,TopCList,BotList,BotCList):-
        split(List,CList,TopList,TopCList,BotList,BotCList).

/*************************************
 *     Heuristic C1 Utilities        *
 *************************************/

    check_sloppy_splits(_,[],_,_):- !.
      check_sloppy_splits(List,[Head|Tail],CList,Prod):-
        Head == Prod,
        check_sloppy_splits(List,Tail,CList,Prod),!.

    check_sloppy_splits(List,[Prod1,Prod2|Tail],CList,Prod):-
        Prod1 \= Prod,
```

```
    Prod2 \= Prod,
    split_above(List,Prod1,Top,Bot),
    remove_split_from_choice(List,CList,CList,Top,Bot,Prod),
    check_sloppy_splits(List,[Prod2|Tail],CList,Prod),!.
check_sloppy_splits(List,[Prod1,Prod2|Tail],CList,Prod):-
    Prod1 \= Prod,
    Prod2 == Prod,
    check_sloppy_splits(List,[Prod2|Tail],CList,Prod),!.
check_sloppy_splits(List,[Prod1|Tail],CList,Prod):-
    Prod1 \= Prod,
    Tail == [],
    check_sloppy_splits(List,Tail,CList,Prod),!.

remove_split_from_choice(_,_,[],_,_,_):- !.
remove_split_from_choice(List,CList,[Head|Tail],Top,Bot,Prod):-
        recorded(table,dsst(List,CList,Type,Top,Bot,Head,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),Ref),
    write('Most plentiful product is '),write(Prod),write('. The '),
        write(Type),
    write(' split '),write(Top),write(' / '),write(Bot),write(' with '),
        write(Head),write('/'),write(HK),write(' as LK/HK'),
    write(' violates heuristic C1: remove the most plentiful'),
    write(' product first. Do you wish to remove this'),
    write(' split from consideration or maintain it? '),
    send_menu_msg(activate(remove,(14,30)),RChoicesplit),
    choose_split(RChoicesplit),
    erase(Ref),
    recordz(table,ddsst(List,Clist,Type,Top,Bot,Head,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),_),
    remove_split_from_choice(List,CList,Tail,Top,Bot,Prod),!.
remove_split_from_choice(List,CList,[_|Tail],Top,Bot,Prod):-
    remove_split_from_choice(List,CList,Tail,Top,Bot,Prod),!.

check_sharp_splits(_,[_],_,_):- !.
check_sharp_splits(List,[Comp1|Tail],CList,Prod):-
    recorded(data,flow(Prod,Comp1,CompFlow),_),
    CompFlow == 0,
    split_below(CList,Comp1,TopComp,_),
    sum_component_flow(TopComp,TopCompFlow,[Prod]),
    TopCompFlow == 0,
    check_sharp_splits(List,Tail,CList,Prod),!.
check_sharp_splits(List,[Comp1|Tail],CList,Prod):-
        recorded(data,flow(Prod,Comp1,CompFlow),_),
    CompFlow == 0,
    split_above(CList,Comp1,_,BotComp),
    sum_component_flow(BotComp,BotCompFlow,[Prod]),
    BotCompFlow == 0,
    check_sharp_splits(List,Tail,CList,Prod),!.

check_sharp_splits(List,[Comp1|Tail],CList,Prod):-
        recorded(data,flow(Prod,Comp1,CompFlow),_),
    CompFlow == 0,
    split_above(CList,Comp1,TempTopComp,BotComp),
    sum_component_flow(BotComp,BotCompFlow,[Prod]),
    BotCompFlow \=0,
    delete_element(Comp1,TempTopComp,TopComp),
    sum_component_flow(TopComp,TopCompFlow,[Prod]),
    TopCompFlow \= 0,
    recorded(table,dsst(List,CList,'sharp',Top,Bot,Comp1,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),Ref),
    write('Most plentiful product is '),write(Prod),write('. The sharp'),
    write(' split '),write(Top),write(' / '),write(Bot),write(' with '),
        write(Comp1),write('/'),write(HK),write(' as LK/HK'),
        write(' violates heuristic C1: remove the most plentiful'),
```

```
        write(' product first. Do you wish to remove this'),
        write(' split from consideration or maintain it? '),
        send_menu_msg(activate(remove,(14,30)),RChoicesplit),
        choose_split(RChoicesplit),
        erase(Ref),
        recordz(table,ddsst(List,CList,'sharp',Top,Bot,Comp1,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),_),
        check_sharp_splits(List,Tail,CList,Prod),!.

    check_sharp_splits(List,[Comp1|Tail],CList,Prod):-
        recorded(data,flow(Prod,Comp1,CompFlow),_),
        CompFlow \= 0,
        split_above(CList,Comp1,_,BotComp),
        sum_component_flow(BotComp,BotCompFlow,[Prod]),
        BotCompFlow == 0,
        check_sharp_splits(List,Tail,CList,Prod),!.

    check_sharp_splits(List,[Comp1|Tail],CList,Prod):-
        recorded(data,flow(Prod,Comp1,CompFlow),_),
        CompFlow \= 0,
        split_above(CList,Comp1,_,BotComp),
        sum_component_flow(BotComp,BotCompFlow,[Prod]),
            BotCompFlow \= 0,
        recorded(table,dsst(List,CList,'sharp',Top,Bot,Comp1,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),Ref),
        write('Most plentiful product is '),write(Prod),write('.'),nl,
            write('The sharp split '),write(Top),write(' / '),write(Bot),write(' with '),
        nl,write(Comp1),write('/'),write(HK),write(' as LK/HK'),
        write(' violates heuristic C1: '),nl,
            write('remove the most plentiful product first.'),nl,
            write('Do you wish to remove this split from '),nl,
            write('consideration or maintain it? '),
        send_menu_msg(activate(remove,(14,30)),RChoicesplit),
        choose_split(RChoicesplit),
        erase(Ref),
        recordz(table,ddsst(List,CList,'sharp',Top,Bot,Comp1,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),_),
        check_sharp_splits(List,Tail,CList,Prod),!.
    check_sharp_splits(List,[_|Tail],CList,Prod):-
        check_sharp_splits(List,Tail,CList,Prod),!.

    diagonal_zeros(H1,A1,H2,A2):-
        recorded(data,flow(H1,H2,0),_),
        recorded(data,flow(A1,A2,0),_),!.
    diagonal_zeros(H1,A1,H2,A2):-
            recorded(data,flow(H1,A2,0),_),
            recorded(data,flow(A1,H2,0),_),!.

/*  End of Heurstic C1 Utilitites  */

/*****************************************************************
 *  Heuristic C2 Utilities                                       *
 *****************************************************************/

    determine_two_best_split_ratios(List,CList,Top1,Bot1,Type1,LK1,HK1,Ratio1,CES1,Top2,Bot2,Type2,LK2,HK2,Ratio2,CES2):-
      determine_best_split_ratio(List,CList,[],[],'none','none','none',100,0,Top1,Bot1,Type1,LK1,HK1,Ratio1,CES1,'yes'),
          adjust_database(List,CList,Type1,Top1,Bot1,LK1,HK1),
          determine_best_split_ratio(List,CList,[],[],'none','none','none',100,0,Top2,Bot2,Type2,LK2,HK2,Ratio2,CES2,'no'),
          readjust_database(List,CList), !.

    adjust_database(List,CList,_,_,_,_,_):-
        recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),Ref),
        recordz(table,ddddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),_),
        recordz(table,dddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),_),
```

```
        erase(Ref),
        fail.

    adjust_database(List,CList,_,_,_,_,_):-
        recorded(table,dddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),Ref),
        recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),_),
        erase(Ref),
        fail.

    adjust_database(List,CList,Type,Top,Bot,LK,HK):-
        recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),Ref),
        erase(Ref),
            !.

    readjust_database(List,CList):-
        recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),Ref),
        erase(Ref),
        fail.

    readjust_database(List,CList):-
        recorded(table,ddddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),Ref),
        recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),_),
        erase(Ref),
        fail.

    readjust_database(_,_):- !.

    determine_whether_to_use_CES(TestRatio,Ratio2,'no'):-
        TestRatio < Ratio2, !.
    determine_whether_to_use_CES(TestRatio,Ratio2,'yes'):-
        TestRatio >= Ratio2, !.

    determine_best_split_ratio(List,CList,_,_,_,_,_,TempRatio,_,Top,Bot,Type,LK,HK,Ratio,CES,Writeoption):-
        get_split(List,CList,TestType,TestTop,TestBot,TestLK,TestHK,TestCES),
            [! resulting_products(List,TestType,TestTop,TestBot,TestTopList,TestTopCList,TestBotList,TestBotCList),
        get_split_flow(TestTopList,TestTFlow,TestTopCList,0),
        get_split_flow(TestBotList,TestBFlow,TestBotCList,0),
            solve_ratio(TestTFlow,TestBFlow,Ratio1),
            solve_ratio(TestBFlow,TestTFlow,Ratio2),
        max(Ratio1,Ratio2,TestRatio),
        line_write(TestTop,TestBot,TestLK,TestHK,TestTFlow,TestBFlow,Ratio1,TestCES,Writeoption)
            !],
        TestRatio < TempRatio,

determine_best_split_ratio(List,CList,TestTop,TestBot,TestType,TestLK,TestHK,TestRatio,TestCES,Top,Bot,Type,LK,HK,Ratio,
CES,Writeoption).


determine_best_split_ratio(List,CList,TestTop,TestBot,TestType,TestLK,TestHK,TestRatio,TestCES,Top,Bot,Type,LK,HK,Ratio,
CES,Writeoption):-
        recorded(table,dsst(List,CList,_,_,_,_,_,_,_,'feasible','easy',_),Ref),

determine_best_split_ratio(List,CList,TestTop,TestBot,TestType,TestLK,TestHK,TestRatio,TestCES,Top,Bot,Type,LK,HK,Ratio,
CES,Writeoption).

    determine_best_split_ratio(List,CList,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_):-
        recorded(table,ddddddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),Ref),
        erase(Ref),
        recordz(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,Feasibility,Ease,CES),_),
        fail.
```

```
determine_best_split_ratio(_,_,Top,Bot,Type,LK,HK,Ratio,CES,Top,Bot,Type,LK,HK,Ratio,CES,_):- !.

line_write(Top,Bot,LK,HK,TFlow,BFlow,Ratio,CES,'yes'):-
   X is 100/(1 + Ratio),
   Y is 100 - X,
   max(X,Y,Z1),
   minimum(X,Y,Z2),
   recorded(data,k_value(LK,KLK),_),
       recorded(data,k_value(HK,KHK),_),
   Alpha is KLK/KHK,
       Alphat is round(Alpha,2),
       TFlowt is round(TFlow,2),BFlowt is round(BFlow,2),
       Z1t is integer(round(Z1,0)),Z2t is integer(round(Z2,0)),
   CESt is round(CES,2),
   nl,
       tget(R,C),
       write(Top),write('/'),write(Bot),
       NR is R+1,CR is C+16,
       tmove(NR,CR),
   write(LK),C2 is CR+6,tmove(NR,C2),write(HK),
       C3 is C2+6,tmove(NR,C3),write(Alphat),
       C4 is C3+10,C5 is C4+11,
       tmove(NR,C4),write(TFlowt),
   tmove(NR,C5),write(BFlowt),
       C6 is C5+9,
   tmove(NR,C6),write(Z1t),write('/'),write(Z2t),
       C7 is C6+9,
   tmove(NR,C7),write(CESt),
   pause, !.

line_write(_,_,_,_,_,_,_,_,'no'):- !.

get_split(List,CList,Type,Top,Bot,LK,HK,CES):-
       recorded(table,dsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),Ref),
       erase(Ref),
       recordz(table,dddddsst(List,CList,Type,Top,Bot,LK,HK,Delta,Listofd,Listofb,'feasible','easy',CES),_),
       !.

decide_split_choice(Top,Bot,Type,LK,HK,Ratio,CES,_,_,_,_,_,_,_,Top,Bot,Type,LK,HK,Ratio,CES,'no'):- !.

decide_split_choice(Top,Bot,Type,LK,HK,Ratio,CES1,_,_,_,_,_,_,CES2,Top,Bot,Type,LK,HK,Ratio,CES1,'yes'):-
  CES1 >= CES2, !.
decide_split_choice(_,_,_,_,_,_,_,CES1,Top,Bot,Type,LK,HK,Ratio,CES2,Top,Bot,Type,LK,HK,Ratio,CES2,'yes'):-
  CES1 < CES2, !.

resulting_products(_,Type,Top,Bot,TopList,TopCList,BotList,BotCList):-
    Type == 'sloppy',
    TopList = Top,
    BotList = Bot,
    positive_component_flow_list(TopList,TopCList),
    positive_component_flow_list(BotList,BotCList).

resulting_products(List,Type,Top,Bot,TopList,TopCList,BotList,BotCList):-
    Type == 'sharp',
    TopCList = Top,
    BotCList = Bot,
    determine_sharp_products(List,TopCList,[],TopList),
    determine_sharp_products(List,BotCList,[],BotList).
    determine_sharp_products([],_,R,R).

determine_sharp_products([Head|Tail],CList,RefList,TorBList):-
```

```
        product_flow(Head,PFlow,CList),
        PFlow > 0,
        append(RefList,[Head],NewRefList),
        determine_sharp_products(Tail,CList,NewRefList,TorBList).

    determine_sharp_products([Head|Tail],CList,RefList,TorBList):-
        product_flow(Head,PFlow,CList),
        PFlow =< 0,
        determine_sharp_products(Tail,CList,RefList,TorBList).

/* End of Heuristic C2 Utilities */

    decide_split_path(heuristically):- fail.
    decide_split_path(manually):-nl,!.

    choose_split(notsplit):- fail.
    choose_split(split):-nl,!.
    choose_split(maintain):- fail.
    choose_split(remove):-nl,!.


    max_ces_sloppy(_,[_],_,TempCES,TempCES,_):-!.
    max_ces_sloppy(List,[Head|Tail],CList,TempCES,CESsloppy,Ease):-
      split_above(List,Head,Top,Bot),
      search_splits_for_max(List,CList,CList,Top,Bot,TempCES,TestCES,Ease),
      max_ces_sloppy(List,Tail,Clist,TestCES,CESsloppy,Ease),!.
    max_ces_sloppy(List,[_|Tail],CList,TempCES,CESsloppy,Ease):-
      max_ces_sloppy(List,Tail,Clist,TempCES,CESsloppy,Ease),!.

    search_splits_for_max(_,_,[],_,_,CES,CES,_):- !.
    search_splits_for_max(List,CList,[Head|Tail],Top,Bot,TempCES,CESsloppy,Ease):-
        recorded(table,dsst(List,CList,_,Top,Bot,Head,_,_,_,_,'feasible',Ease,TestCES),Ref),
        TestCES > TempCES,
        search_splits_for_max(List,CList,Tail,Top,Bot,TestCES,CESsloppy,Ease).
    search_splits_for_max(List,CList,[Head|Tail],Top,Bot,TempCES,CESsloppy,Ease):-
      recorded(table,dsst(List,CList,_,Top,Bot,Head,_,_,_,_,'feasible',Ease,TestCES),Ref),
      TestCES =< TempCES,
      search_splits_for_max(List,CList,Tail,Top,Bot,TempCES,CESsloppy,Ease).

    max_ces_sharp(_,_,[],CES,CES,_):- !.
    max_ces_sharp(List,CList,[Head|Tail],TempCES,CESsharp,Ease):-
        split_above(List,Head,TopComp,BotComp),
        recorded(table,dsst(List,CList,'sharp',TopComp,BotComp,_,_,_,_,_,'feasible',Ease,TestCES),Ref),
        TestCES > TempCES,
        max_ces_sharp(List,CList,Tail,TestCES,CESsharp,Ease).
    max_ces_sharp(List,CList,[Head|Tail],TempCES,CESsharp,Ease):-
          split_above(List,Head,TopComp,BotComp),
          recorded(table,dsst(List,CList,'sharp',TopComp,BotComp,_,_,_,_,_,'feasible',Ease,TestCES),Ref),
          TestCES =< TempCES,
          max_ces_sharp(List,CList,Tail,TempCES,CESsharp,Ease).
    max_ces_sharp(List,CList,[_|Tail],TempCES,CESsharp,Ease):-
      max_ces_sharp(List,CList,Tail,TempCES,CESsharp,Ease).


    minimum(X,Y,X):- X < Y, !.
    minimum(X,Y,Y):- X >= Y, !.

erase_several(X):-
            recorded_tro(table,X,Ref),
            hard_erase(Ref),
            fail.
```

```
erase_several(X).
```

ABSORPTION MODULE (ABSORB.ARI)

```
/*******************************************************
*ABSORPTION MODULE -- This module implements a short*
*cut method for the evaluation of A and N for an    *
* absorption column with recovery, using Douglas and*
*Keller heuristics,  and Kremser equation.          *
*The main goal is called 'ABSORB'  this version     *
*needs 2 database files: one for the flows and one  *
*for the Henry law constants     2/4                *
*******************************************************/
:-segment(absseg).
:-public absorb/6:far,
          d_over_b/3:far,
          identify_desired_separation/3:far,
          input_henrys_law/1:far,
          make_henlist_from_list/3:far,
          make_hlist_from_clist/2:far,
          print_specifications/3:far,
          sort_a_list_by_hhlist/3:far,
          actual_flows/4:far,
          write_round/2:far.
:-visible retrieveb/3,recorded/3.
:-extrn
          append/3:far,
          append_v/3:far,
          component_flow/3:far,
          component_flow_list/4:far,
          component_flow_set/3:far,
          decide_difficult_or_easy/2:far,
          delete_element/3:far,
          delete_elements/3:far,
          equal/2:far,
          erase_term/2:far,
          ffactor/3:far,
          first_prod/2:far,
          flow_set/3:far,
          flow_set_c/3:far,
          get_split_flow/4:far,
          largest_flow/3:far,
          last_prod/2:far,
          list_member/2:far,
          max/3:far,
          max_flist/2:far,
          max_flow/3:far,
          member/2:far,
          pause/0:far,
          pause_escape/0:far,
          positive_component_flow_list/2:far,
          product_flow/3:far,
          product_flow_match/4:far,
          product_flow_set/3:far,
          print_cam_driver/3:far,
          quicksort/2:far,
          reverse/2:far,
          selected_positive_component_flow_list/3:far,
          set_above/3:far,
          set_below/3:far,
```

```
        split_above/4:far,
        split_below/4:far,
        sub_list/2:far,
        sum_component_flow/3:far,
        sum_flist/2:far,
        two_highest_flows/3:far,
        readkey/1,
        readsolvent/2,
        readdata_default/1,
        synthesize_process/3,
        report/0.

absorb(List,CList,TopList,TopCList,BotList,BotCList):-

        /*******************************************************
        *Instantiate the knowledge of the database to the     *
        *variable used in the module. Then transform and      *
        *create lists for the CAM representation              *
        *******************************************************/
        [!
        recorded(data,optimal_A(Aopt),_),
        recorded(data,limit_A(Asl),_),
        Ast is Asl+1,recorda(data,stage_A(Ast),_), %initialization
        removeallb(absorb),
        get_split_flow(List,V,CList,0),
        cls,
        write('                    ┌──────────────────┐'),nl,
        write('                    | Gas Absorption |'),nl,
        write('                    └──────────────────┘'),nl,
        nl,nl,nl,
        write('  Solvent being added to stream .... '),nl,
        write('  Component Assignment Matrix being recalculated'),nl,
        write('    with equilibrium ratios from Henry`s law.'),nl,nl,
        recorded(data,solvent(S),_),
        make_hlist_from_clist(CList,HList),
        quicksort(HList,HHCList),
        sort_a_list_by_hhlist(CList,HHCList,CLList),
        make_henlist_from_list(List,CList,HL),
        quicksort(HL,HHL),
        sort_a_list_by_hhlist(List,HHL,LList),
        append(LList,[solvent],NList),
        append(CLList,[S],NCList),
        nl,nl,write('  Calculation complete. '),pause,
        cls,current_window(_,two),
        print_cam_driver(LList,CList,solvent),
        cls,current_window(_,four),

        /*******************************************************
        *Identify the desired separation and calculate the    *
        *initial value for the invcrement of the volume.      *
        *******************************************************/

        cls,current_window(_,four),
        cls,
        write('                    ┌──────────────────┐'),nl,
        write('                    | Gas Absorption |'),nl,
        write('                    └──────────────────┘'),nl,
        identify_desired_separation(NList,TopList,BotList),
        recorded(data,key_component(dga,KC),_),
        recorded(data,henry(KC,KConst),_),
        calc_min_L(Amin,List,CList,V,KConst,TopList,BotList)
```

```
        !],

        /**************************************************
        * Start searching for a solution. Kremser is the  *
        * main functor of the module.                     *
        **************************************************/
        for(Amin,Asl,A),
        erase_term(data,flow(solvent,_,_)),
        [!
        L is A*KConst*V,
        kremser_dga(LList,NCList,[S],L,V,KConst,TopList,BotList),
        current_window(_,four)
        !],
    fail.

        /**************************************************
        * When the each loop is finished, it fails .      *
        * The Absorb clause continues in the following code*
        * When a solution is found, it is asserted in the *
        * database of the whole program. Then the recovery *
        * column is calculated. The stripping module is   *
        * entered or the recovery is assumed feasible and *
        * the split sharp (choice of the user)            *
        **************************************************/

absorb(List,CList,_,_,_,_):-
        choose_sep_dga,
        recorded(sep,dseparator(Num,_,_,_,_),_),
        NNum is Num + 1,
        recorded(data,sep_CA(X),_),
        retrieveb(absorb,X,dseparator_dga(N,A,NCList,TopList,TopAFList,BotList,BotAFList)),
        recorda(sep,dseparator(NNum,NCList,TopAFList,NCList,BotAFList),_),
        recovery_dga,
        cls,nl,nl,
        write(' The absorption and recovery column '),nl,
        write(' are specified the results are: '),nl,nl,
        write('  Column        Separation'),nl,
        write(' ------        ----------'),nl,nl,
        write('  Absorber      '),write(TopList),write('/'),write(BotList),nl,
        delete_element(solvent,BotList,RTop),
        write('  Recovery      '),write(RTop),write('/'),write([solvent]),
        pause.

absorb(List,CList,_,_,_,_):-
        tmove(5,1),
        write('As no solution has been found the module ends here.'),
        pause.


/*********************************************************************
* End of the main part of the module. The following functors are used *
* by it.                                                              *
*********************************************************************/
/**************************************************
* This clause calculates the minimum solvent molar *
* flowrate according to the method explained in   *
* "Distillation", Van Winkle.                     *
**************************************************/

calc_min_L(Amin,List,CList,V,KConst,TopList,BotList):- %%%%verif
```

```
        recorded(data,solvent(a),_),
        write('Do you know the total volumetric flowrate'),
        send_menu_msg(activate(yes_no,(17,32)),Decision),!,
        Decision=yes,nl,nl,
        write('Enter this flowrate'),
        read(VolFlow),
        delete_element(solvent,BotList,Top_no_S_List),
        sum_component_flow(CList,BotFlow,Top_no_S_List),
        sum_component_flow(CList,TopFlow,TopList),
        VolFlowTop is TopFlow/VolFlow*100,
        VolFlowBot is BotFlow/VolFlow*100,
        recorded(data,key_component(dga,KC),_),
        sum_component_flow([KC],KCFlow,List),
        sum_component_flow([KC],KCBotFlow,BotList),
        ES is KCBotFlow/KCFlow*100,
        Lmin is ES*(VolFlowBot+VolFlowTop)/KConst-VolFlowBot,
        Amin is Lmin/KConst/V.

        /*******************************************************
        * If the solvent is not a or b the minimum absorp- *
        * tion factor is assumed to come from the .fac file*
        *******************************************************/

calc_min_L(Amin,_,_,_,_,_,_):-
        recorded(data,minimum_A(Amin),_).

/*******************************************************
* This clause is equivalent to a loop. It enables  *
* the program to increment the flowrate of the     *
* lean gas.                                        *
*******************************************************/

for(X,Y,X):-X>Y,!.
for(Y,X,Y):-true.
for(X,Y,Z):-inc2(X,X1),for(X1,Y,Z).

inc2(X,X1):-recorded(data,step(Incstep),_),X1 is X+Incstep.

/*******************************************************
* The following clause are used to manipulate the*
* lists. (Sorting...)                              *
*******************************************************/

make_hlist_from_clist([],[]).
make_hlist_from_clist([HC|TC],[HH|TH]):-
      recorded(data,henry(HC,HH),_),
      make_hlist_from_clist(TC,TH).

sort_a_list_by_hhlist(_,[],[]).
sort_a_list_by_hhlist(List,[HH|TH],[HL|TL]):-
        recorded(data,henry(HL,HH),_),
        member(HL,List),
        sort_a_list_by_hhlist(List,TH,TL).

make_henlist_from_list([],_,[]).
make_henlist_from_list([HL|TL],CList,[HH|TH]):-
        product_flow(HL,PFlow,CList),
        henry_prod_flow(HL,HPFlow,CList),
        adjust_hl(HPFlow,PFlow,HH),
        recordz(data,henry(HL,HH),_),
        make_henlist_from_list(TL,CList,TH).
```

```
henry_prod_flow(Head,HPFlow,CList):-
        flow_setP(Head,CList,HFlowList),
        sum_flist(HFlowList,HPFlow).

flow_setP(_,[],[]).
flow_setP(Head,[HC|TC],[HHF|THF]):-
        recorded(data,flow(Head,HC,Flow),_),
        recorded(data,henry(HC,Hval),_),
        HHF is Hval * Flow,
        flow_setP(Head,TC,THF).

adjust_hl(0,_,0).
adjust_hl(Pi2List,PiList,HlAdj):-
        HlAdj is Pi2List / PiList.

identify_desired_separation(NList,TopList,BotList):-
        recorded(data,split_product(Prod),_),
        split_above(NList,Prod,TopList,BotList).

/*****************************************************
*          KREMSER CLAUSE                  *
*****************************************************/

kremser_dga(List,CList,[S],L,V,KConst,TopList,BotList):-
        recorded(data,henry(KC,KConst),_),
        recorded(data,limit_A(Asl),_),
        recorded(data,optimal_A(Aopt),_),
        recorded(data,optimal_N(Nopt),_),
        A is L/(KConst * V),
%       write('V='),write(V),write('L='),write(L),write('A='),write(A),
        component_flow(List,KC,KCFlow),
        Yi_in is KCFlow / V,
        component_flow(TopList,KC,KCTFlow),
        sum_component_flow(CList,TTFlow,TopList),
        calc_Yi_out(KCFlow,KCTFlow,TTFlow,Yi_out),
        recorded(data,mole_fraction(solvent,KC,XKC_in),_),
        !,
        calc_number_stage_dga(A,Yi_in,Yi_out,KConst,XKC_in,N),
%       write(' N='),write(N),nl,
        !,
        check_every_dga(List,CList,L,RL,V,KConst,N,TopList,BotList,DBact,DBspec),
        tmove(5,1),
        write(' Absorption appears feasible for this separation.'),nl,
        write('  Component recovery estimates are as follows:'),nl,nl,
        write('                Specified      Estimated'),nl,
        write('  Component       d/b            d/b'),nl,
        write(' ---------    -------     -------'),nl,
        print_specifications(CList,DBact,DBspec),pause,cls,
        solvent_flow_record(List,S,RL),
        current_window(_,five),
        cls,
        write('                          ┌──────────────────┐ '),nl,
        write('                          | Gas Absorption | '),nl,
        write('                          └──────────────────┘ '),nl,
        nl,nl,nl,
        write(' The heuristic analysis is as follows:'),nl,nl,
        write('  The absorption factor (L/kV) is: '),
        W2 is round(A,2),write(W2),nl,
        write('  Heuristic optimum for absorption factor: A = '),write(Aopt),nl,nl,
        write('  The number of theoretical stages: '),
        W1 is round(N,1),write(W1),nl,
```

```
        write('  Heuristic optimum number of stages:'),write(Nopt),nl,nl,
        current_window(_,seven),cls,
        diag_dga(N,L,NewL,RL,KConst,V),
        current_window(_,five),
        calc_dev(DBact,DBspec,Dev),
        store_sep_dga(Dev,N,A,List,CList,TopList,BotList,DBact).

/*****************************************************
* This clause calculates the molar fractions in the*
* output vapor. If yi_out is 0, N becomes infinite *
* then we assume yi_out to be 2%.                   *
*****************************************************/

calc_Yi_out(KCFlow,KCTFlow,TTFlow,Yi_out):-
        KCTFlow is 0,
        Yi_out is 0.02 * KCFlow / TTFlow.

calc_Yi_out(_,KCTFlow,TTFlow,Yi_out):-
        Yi_out is KCTFlow / TTFlow.

/*****************************************************
* This clause calculates the number of theoretical *
* stages, it takes care of the case A=1             *
*****************************************************/

calc_number_stage_dga(A,Yi_in,Yi_out,KConst,XKC_in,N):-
        A=\=1,
        R is (Yi_in-Yi_out)/(Yi_in-KConst*XKC_in),
        (R-A)/(R-1) > 0,
        N is log((R-A)/(R-1))/log(A) - 1.0.

calc_number_stage_dga(A,Yi_in,Yi_out,KConst,XKC_in,N):-
        A=:=1,
        N is (Yi_in-Yi_out)/(Yi_out-KConst*XKC_in).

/*****************************************************
* This functor succeeds only when N>=Nopt, and      *
*     Amin < A < Asl (<Ast for which N<Nopt)        *
* A feasible separation is added to the top of the *
* database.                                         *
*****************************************************/

check_every_dga(List,CList,L,RL,V,KConst,N,TopList,BotList,DBact,DBspec):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
        check_A_opt(L,V,KConst,NewL),
        !,
        check_in_spec_dga(List,CList,L,NewL,RL,V,KConst,N,TopList,BotList,DBact,DBspec).

        /*****************************************************
        * This functor succeeds only when N<Nopt,equivalent*
        * to      Ast < A and if  A < Asl                   *
        * A feasible separation is added to the bot of the *
        * database, then the first separation on the data- *
        * base is the 'least worse' because N closest to   *
        * Nopt and L minimum.                               *
        *****************************************************/

check_every_dga(List,CList,L,RL,V,KConst,N,TopList,BotList,DBact,DBspec):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N<Nopt,
```

```
        recorded(data,limit_A(Asl),_),
        A is L/(KConst*V),
        store_Ast(A),
        sum_component_flow(CList,TTFlow,TopList),
        !,
        components_within_spec_dga(List,CList,L,V,N,TopList,BotList,TTFlow,DBact,DBspec),
        RL is L.

store_Ast(A):-
        [!recorded(data,stage_A(Ast),Ref)!],
        A < Ast,        % Ast is initialized to Asl
        NewAst is A,replace(Ref,stage_A(NewAst)).
store_Ast(A).

/*****************************************************
* These clauses verify if for a given L, the        *
* corresponding A is or not less than A optimal.    *
*****************************************************/


check_A_opt(L,V,KConst,NewL):-
    A is L/(V*KConst),
    [!recorded(data,optimal_A(Aopt),_)!],
    A < Aopt,
    NewL is Aopt*KConst*V.

check_A_opt(L,V,KConst,NewL):-  %verif
        A is L/(KConst*V),
        [!recorded(data,optimal_A(Aopt),_)!],
        A >=Aopt,
        NewL is L.

/*****************************************************
* This clause succeeds if in spec case 1 and 3      *
*****************************************************/
check_in_spec_dga(List,CList,L,NewL,RL,V,_,N,TopList,BotList,DBact,DBspec):-
        sum_component_flow(CList,TTFlow,TopList),

components_within_spec_dga(List,CList,L,V,N,TopList,BotList,TTFlow,DBact,DBspec),
        RL is L.

  /*****************************************************
  * This clause succeeds if not in with current L     *
  *  and in spec with this imposed L=Aopt*... Case 2  *
  *****************************************************/


check_in_spec_dga(List,CList,L,NewL,RL,V,KConst,N,TopList,BotList,DBact,DBspec):-
        [!recorded(data,optimal_A(Aopt),_)!],
        NewL is Aopt*KConst*V,
    sum_component_flow(CList,TTFlow,TopList),
    components_within_spec_dga(List,CList,NewL,V,N,TopList,BotList,TTFlow,DBact,DBspec),
    RL is NewL.


/*****************************************************
* Clause which decides if the d/b's are in the      *
* range ±10% of the specification.                  *
*****************************************************/

components_within_spec_dga(_,[],_,_,_,_,_,_,[],[]).
```

```
components_within_spec_dga(List,[HC|TC],L,V,N,TopList,BotList,TTFlow,[HDa|TDa],[HDs|TDs]):-
        record_comp_flow_in_solvent([solvent],[HC],L),
        component_flow(TopList,HC,Dspec),
        component_flow(BotList,HC,Bspec),
        d_over_b(Dspec,Bspec,DBspec),
        component_flow(List,HC,FFlow),
        Yi_in_actual is FFlow / V,
        recorded(data,henry(HC,HConst),_),
        A is L / (HConst * V),
        recorded(data,mole_fraction(solvent,HC,Xi_in),_),
        calc_Yi_out_actual(A,Yi_in_actual,HConst,Xi_in,N,FFlow,TFlow,Yi_out_actual),
        Dact is Yi_out_actual * V,
        Bact is FFlow - Dact,
        d_over_b(Dact,Bact,DBact),
        recorded(data,error(Er),_),
        !,
        DBact < (1+Er) * DBspec,
        DBact > (1-Er) * DBspec,
        HDa is DBact,
        HDs is DBspec,
        components_within_spec_dga(List,TC,L,V,N,TopList,BotList,TTFlow,TDa,TDs).

/*****************************************************
* Utility clauses                                    *
*****************************************************/
solvent_flow_record([],S,L):-
        recordz(data,flow(solvent,S,L),_).
solvent_flow_record([H|T],S,L):-
        recordz(data,flow(H,S,0),_),
        solvent_flow_record(T,S,L).

record_comp_flow_in_solvent([solvent],[],_).
record_comp_flow_in_solvent([solvent],[HC|TC],L):-
        recorded(data,mole_fraction(solvent,HC,X),_),
        LX is L*X,
        recorda(data,flow(solvent,HC,LX),_),
        record_comp_flow_in_solvent([solvent],TC,L).

/*****************************************************
* Calculation of yi_out requires many different      *
* cases to avoid infinite values or ln(y<=0).        *
*****************************************************/

calc_Yi_out_actual(A,Yi_in_actual,HConst,Xi_in,N,_,_,Yi_out_actual):-
        abs(A-1)>1e-8,
        abs((N+1)*ln(A))<10,
        Yi_out_actual is Yi_in_actual-(Yi_in_actual-Xi_in*HConst)*(exp((N+1)*ln(A))-A)/(exp((N + 1)* ln(A)) - 1).
        calc_Yi_out_actual(A,Yi_in_actual,HConst,Xi_in,N,FFlow,TTFlow,Yi_out_actual):-
        abs(A-1)>1e-8,
        (N+1)*ln(S)>0,
        Yi_out_actual is Xi_in*HConst+0.02*FFlow/TTFlow.

calc_Yi_out_actual(A,Yi_in_actual,HConst,Xi_in,N,_,_,Yi_out_actual):-
        abs(A-1)>1e-8,
        Yi_out_actual is Yi_in_actual-(Yi_in_actual-Xi_in*HConst)*A.

calc_Yi_out_actual(A,Yi_in_actual,HConst,Xi_in,N,_,_,Yi_out_actual):-
        Yi_out_actual is (Yi_in_actual+Xi_in*HConst*N)/(N+1).

/*****************************************************
* Diagnose clauses :                                 *
```

```
* 4 different cases exists but they are not all    *
* feasible or pratical                              *
*****************************************************/

diag_dga(N,L,NewL,RL,KConst,V):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
        [!recorded(data,optimal_A(Aopt),_)!],
        NewL is Aopt*V*KConst,
        RL is NewL,
        write(' L is imposed to a   '),nl,
        write(' higher optimum value.'),nl,
        write('   N is minimized.    '),nl,
        write('       [case 1]       '),pause.

diag_dga(N,L,NewL,RL,KConst,V):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
    [!recorded(data,optimal_A(Aopt),_)!],
        NewL is Aopt*V*KConst,
        RL is L,
        RL/KConst/V < Aopt,
        write(' L is lower than the '),nl,
        write('  optimum value N is '),nl,
        write('   higher than the   '),nl,
        write('        optimum      '),nl,
        write('        [case 2]     '),pause.

diag_dga(N,L,NewL,RL,KConst,V):-
        [!recorded(data,optimal_N(Nopt),_)!],
      N>=Nopt,
        NewL is L,
      [!recorded(data,limit_A(Asl),_)!],
        RL < Asl*V*KConst,
        write(' L is higher than the'),nl,
        write(' optimum value.      '),nl,nl,
        write('    The minimal      '),nl,
        write(' solution for N is   '),nl,
        write(' greater than   the '),nl,
        write(' optimum of '),write(Nopt),write('.'),nl,nl,
        write('         [case 3]'),pause.


diag_dga(N,L,NewL,RL,KConst,V):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N<Nopt,
        write(' L is much higher than '),nl,
        write(' the optimum value.    '),nl,
        write(' N is lower  than  the '),nl,
        write(' optimum (Nopt='),write(Nopt),write('). '),nl,
        write(' The  separation  is  '),nl,
        write(' unpractical  because '),nl,
        write(' N is too low, and not'),nl,
        write(' economical because V '),nl,
        write(' is too high:  A>Ast.'),
        write('         [case 4]'),pause.

/*****************************************************
* Calculates the average deviation in the doverb 's*
* to be used in the CESS.                           *
*****************************************************/
```

```
calc_dev(DBact,DBspec,Dev):-
        sum_dev(DBact,DBspec,Sum),
        length(DBspec,L),
        Dev is Sum/L.

sum_dev([],[],0).
sum_dev([HA|TA],[HB|TB],Sum):-
        sum_dev(TA,TB,SumTail),
        Sum is (abs(HA-HB)/HA+SumTail).

/*******************************************************
* Intermediate storage of the separations obtained *
* before final choice and end of the module       *
*******************************************************/

store_sep_dga(Dev,N,A,List,CList,TopList,BotList,DBact):-
        recorded(data,solvent(S),_),
        append(List,[solvent],NList),
        append(CList,[S],NCList),
        component_flow_set(TopList,NCList,TopFList),
        component_flow_set(BotList,NCList,BotFList),
        component_flow_set(List,CList,FList),
    actual_flows(FList,DBact,Top_no_s_AFList,Bot_no_s_AFList),
        recorded(data,flow(solvent,S,L),_),
        append(Top_no_s_AFList,[0],TopAFList),
        append(Bot_no_s_AFList,[L],BotAFList),
        calc_CA(Dev,N,A,CA),
        recordb(absorb,CA,dseparator_dga(N,A,NCList,TopList,TopAFList,BotList,BotAFList)).

actual_flows([],[],[],[]).
actual_flows([HF|TF],[DBA|TDBA],[HTAF|TTAF],[HBAF|TBAF]):-
        BA is 1/(1+DBA),
        DA is 1-BA,
        HTAF is DA*HF,
        HBAF is BA*HF,
        actual_flows(TF,TDBA,TTAF,TBAF).

/*******************************************************
* To decide which separation is the best we calcu- *
* late the Coefficient for Absorption.            *
*       It is a function of the average deviation*
* ,the number of stages and the absorption factor *
*******************************************************/

calc_CA(Dev,N,A,CA):-
        recorded(data,optimal_N(Nopt),_),
        recorded(data,optimal_A(Aopt),_),
        CAD is -ln(Dev),
        CAN is -ln( N*N/2-(Nopt-2)*N+(Nopt-2)*(Nopt-2)/2+1 )+2*sqrt(2)*atan( (N-(Nopt-2))/sqrt(2) ),
        CAS is -ln( A*A/2-(Aopt+2)*A+(Aopt+2)*(Aopt+2)/2+1 )-2*sqrt(2)*atan( (A-(Aopt+2))/sqrt(2) ),
    CA is round(exp(CAD+CAN+CAS),0).

/*******************************************************
* We need to choose the best separation among     *
* those generated previously and stored in        *
* dseparator_dga.                                 *
*******************************************************/

choose_sep_dga:-
        [!recorded(data,minimum_A(Amin),_),
```

```
            recorded(data,stage_A(Ast),_),
            recorded(data,limit_A(Asl),_),
            recorded(data,optimal_A(Aopt),_),
            write(' The range of Absorption Factor is:'),nl,
            write(Amin),write(' - '),write(Asl),nl,
            current_window(CW,seven),cls,
            diag_dga_range(Amin,Aopt,Ast,Asl),
            current_window(_,CW)!],
            findall(X,retrieveb(absorb,X,_),ListX),
            quicksort(ListX,[HX|TX]),
            cls,nl,
            write('      The Best Separation According to the Analysis is:'),
            write_sep_dga([HX|TX]).
choose_sep_dga:-fail.

write_sep_dga([]).
write_sep_dga([HX|TX]):-
            retrieveb(absorb,HX,dseparator_dga(N,A,NCList,TopList,TopAFList,BotList,BotAFList)),
            nl,write(' The absorption column '),
      write(' is specified the results are: '),nl,nl,
      write('  Column        Separation'),nl,
      write(' ———      ————————'),nl,nl,
      write('  Absorber      '),write(TopList),write('/'),write(BotList),
            nl,nl,
            write('Components:'),write(NCList),nl,
            write('Top       :'),
            write_round(TopAFList,1),nl,
            write('Bot       :'),write_round(BotAFList,1),nl,nl,nl,
            write(' The design parameters are:'),nl,
            NR is round(N,2),
            write(' Number of theoretical stages:'),write(NR),nl,nl,
            write(' Absorption factor:'),write(A),nl,nl,
            pause,
            write('Do you Accept this one, if not the other solutions will be displayed'),
      send_menu_msg(activate(yes_no2,(10,60)),Decision),!,
            solve_choice_dga(Decision,[HX|TX]).

write_round([],_).
write_round([HL|TL],C):-
      NewC is (8*C) + 10,
            tget(R,_), tmove(R,NewC),
            HLt is round(HL,2),
            write(HLt),
            inc(C,IC),
            write_round(TL,IC).
write_round(_,_).

solve_choice_dga(Decision,[HX|TX]):-
            Decision=yes,
            recordz(data,sep_CA(HX),_).

solve_choice_dga(Decision,[HX|TX]):-
            Decision=no,cls,write_sep_dga(TX).

/*****************************************************
* Here, the program diagnoses if the range of abso-*
* rption factors entered by the user is likely to  *
* contain all the possible solutions.              *
*****************************************************/
```

```
                    /*case1*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Asl < Aopt,
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        write(' If you want to use an '),nl,
        write(' optimal value for A,  '),nl,
        write(' you must not have     '),nl,
        write(' the limit A smaller   '),nl,
        write(' than the optimal value'),nl,
        write(' for A:'),
        write(Aslt),write(' > '),write(Aoptt),write('.'),pause.

                    /*case2*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Amin > Asl,
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        write(' The upper limit Asl is'),nl,
        write(' smaller than the lower'),nl,
        write(' limit Amin:'),nl,
        write(Amint),write(' < '),write(Aslt),write('.'),nl,
        write(' You must have Amin < Asl'),
        write(' for any calculation to'),nl,
        write(' occur.'),pause.

                    /*case3*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Amin < Aopt,
        Aopt < Ast,
        Ast < Asl,
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        recorded(data,optimal_N(Nopt),_),
        Noptt is round(Nopt,2),
        write(' No more solutions are '),nl,
        write('      expected in     '),nl,
        write('   the studied range. '),
        pause.

                    /*case4*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Amin < Aopt,
        Aopt < Asl,
        [!recorded(data,optimal_N(Nopt),_)!],
        Asl < Ast,  %Ast actually does not exist because Asl is to small to reach it, Ast is initialized to Asl+1
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        Noptt is round(Nopt,2),
        write(' Solutions may exist   '),nl,
        write(' for A > '),write(Aslt),write('. With a'),nl,
        write(' number of stages still'),nl,
        write(' higher than '),write(Noptt),write('. You'),nl,
        write(' should increase Asl.'),
        pause.

                    /*case5*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Amin < Ast,
        Ast < Aopt,
        Aopt < Asl,
        recorded(data,optimal_N(Nopt),_),
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        Noptt is round(Nopt,2),
        write(' The  optimal  A ,  '),nl,
```

```
        write(' Aoptt = '),write(Aoptt),write(', should'),nl,
        write(' be  smaller  than '),nl,
        write(' Ast='),write(Astt),write(' because'),nl,
        write(' Aopt ought to be where'),nl,
        write(' N is greater than '),write(Noptt),write('.'),
        pause.
                /*case6*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Amin>=Aopt,
        Amin=:=Ast, %the first value of A will instantiate Ast
        recorded(data,optimal_N(Nopt),_),
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        Noptt is round(Nopt,2),
        write(' The  initial  value,'),nl,
        write(' Amin='),write(Amint),write(', is too high.'),
        write(' Only solutions with N '),nl,
        write(' smaller  than  '),write(Noptt),nl,
        write(' will be generated.'),nl,
        write(' Worse, the optimal A  ').nl,
        write(' Aopt='),write(Aoptt),write(', is too'),nl,
        write(' small to be studied. '),nl,
        write(' It should be greater '),nl,
        write(' than '),write(Amint),write('.'),
        pause.

                /*case7*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Amin>=Aopt,
        Amin<Ast,
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        write(' The  minimum  value  '),nl,
        write(' Amin='),write(Amint),write(' should be '),nl,
        write(' smaller if you want  '),nl,
        write(' to use the optimal   '),nl,
        write(' value Aopt='),write(Aoptt),write('.'),nl,
        pause.

                /*case8*/
diag_dga_range(Amin,Aopt,Ast,Asl):-
        Amin<Aopt,
        Amin =:= Ast, %the first value of A will instantiate Ast
        recorded(data,optimal_N(Nopt),_),
        Amint is round(Amin,2),Aoptt is round(Aopt,2),Astt is round(Ast,2),Aslt is round(Asl,2),
        Noptt is round(Nopt,2),
        write(' The initial value of A'),nl,
        write(' Amin='),write(Amint),write(' is too high.'),nl,
        write(' Only solutions with N'),nl,
        write(' smaller than '),write(Noptt),write(' will '),nl,
        write(' be generated.'),
        pause.

diag_dga_range(_,_,_,_):-
        write('No diagnosis. Error.'),pause.

d_over_b(0,_,0.02040816327).
d_over_b(_,0,49.0).
d_over_b(D,B,DB):-
        D/B>49.0,
        DB is 49.0.
d_over_b(D,B,DB):-
        D/B<0.02040816327,
```

```
        DB is 0.02040816327.
d_over_b(D,B,DB):-
        DB is D/B.

print_specifications([],[],[]).
print_specifications([HC|TC],[DBA|TDBA],[DBS|TDBS]):-
        BA is 1/(1+DBA),
        DA is 1 - BA,
        BS is 1/(1+DBS),
        DS is 1 - BS,
        tab(4),write(HC),tget(R,C),tmove(R,18),
        WDS is round(DS,3), WBS is round(BS,3),
        write(WDS),write('/'),write(WBS),
        tmove(R,34),
        WDA is round(DA,3), WBA is round(BA,3),
        write(WDA),write('/'),write(WBA),nl,
        print_specifications(TC,TDBA,TDBS).


/*****************************************************
* Recovery clause to separate the solvent from    *
* the bottom. Call of the Stripping module        *
* possible.                                       *
*****************************************************/

recovery_dga:-
        [![!current_window(Cur,eight),
    recorded(sep,dseparator(Rum,NCList,TopAFList,NCList,BotAFList),_),
        cls,
        write(' The recovery column is going to be calculated.'),nl,
        write(' The split can be sharp and assumed feasible or'),nl,
        write(' a stripping simulation can be held.'),nl,
        write(' Do you want the stripping simulation ?'),nl,
        send_menu_msg(activate(no_yes,(17,32)),Decision),cls!],
        solve_decision(Decision),nl,nl,
        write(' Please enter the data:'),nl,
    nl,
        readsolvent(s,Name),nl,
        write(' Are Henry''s law constants available for the system using '),
        recorded(data,lean_gas(LG),_),
    write(LG),write(' ? '),nl,
        send_menu_msg(activate(yes_no,(17,32)),Available)!],
        [!Available=yes,
        recorded(data,solvent(S),_),
        input_henrys_law(NCList),cls,nl,
        readflowreco(NCList,BotAFList,S,LG),nl,
        readkey(s),nl,
        recorda(data,initial_set([ps1,ps2]),_),
        recorda(data,initial_components(NCList),_),
        recorda(data,split_product(ps2),_),
        recorded(data,initial_set(SList),_),
        recorded(data,initial_components(SCList),_)!],
        current_window(_,five),cls,
        readdata_default(rec),
        synthesize_process(SList,SCList,[s]).

recovery_dga:-
        write('recov sharp'),
        cls,
        write('The recovery of the solvent is supposed to be feasible and sharp'),nl,
    recorded(sep,dseparator(Rum,NCList,TopAFList,NCList,BotAFList),_),
```

```
        recorded(data,solvent(S),_),
            RRum is Rum + 1,
            recorda(bypass,dstreambypass(RRum,'none',[],[]),_),
        recorda(sep,dstreamin(RRum,NCList,BotAFList),_),
            reverse(BotAFList,[L|B_Rev_L]),
        recorda(data,flow(solvent,S,L),_),
            delete_element(S,NCList,CList),
            record_comp_flow_in_solvent([solvent],CList,L),
            append([0],B_Rev_L,Top_Rev_RFList),
            reverse(Top_Rev_RFList,TopRFList),
            component_flow_set([solvent],NCList,BotRFList),
            recorda(sep,dseparator(RRum,NCList,TopRFList,NCList,BotRFList),_),
            write(' The results for the recovery  COLUMN #'),write(RRum),
            write(' are now available'),pause,
            current_window(_,five).

readflowreco(NCList,BotAFList,S,LG):-
            reverse(BotAFList,[L|B_Rev_L]),
            delete_element(S,NCList,CList),
            append([0],B_Rev_L,Top_Rev_RFList),
            reverse(Top_Rev_RFList,TopRFList),
            write(' The specified flow rates are:'),nl,
            write('      To the top of the recovery column:'),nl,
            write('      Components:'),write(NCList),nl,
            write('      Flows:'),write_round(TopRFList,1),nl,
            write('  To the bottom of the recovery column:'),nl,
            write('      Components:'),write(NCList),nl,
            write('      Flows: Zeros except for '),write(S),write(':'),write(L),nl,nl,
            delete_element(S,NCList,CList),
            make_botlist(CList,BotFList),
            append(BotFList,[L],BotRFList),
            record_component_flow([ps2],NCList,TopRFList),
            record_component_flow([ps1],NCList,BotRFList),
            record_mol_fract(NCList).

make_botlist([],[]).
make_botlist([HC|TC],[HF|TF]):-
            HF=0,
            make_botlist(TC,TF).

record_component_flow([P],[],[]).
record_component_flow([P],[HC|TC],[HF|TF]):-
            recorda(data,flow(P,HC,HF),_),
            record_component_flow([P],TC,TF).

record_mol_fract([]).
record_mol_fract([HC|TC]):-
            recorda(data,mole_fraction(lean_gas,HC,0),_),
            record_mol_fract(TC).

solve_decision(no):-!,fail.
solve_decision(yes).

input_henrys_law([]).
input_henrys_law([H|T]):-
            write(' What is the Henry''s law constant of '),write(H),write(' ? '),
            read(HVal),
            recorda(data,henry(H,HVal),_),
            input_henrys_law(T).
```

STRIPPING MODULE (STRIPPIN.ARI)

```
/*******************************************************
*STRIPPING  MODULE -- This module implements a short*
*cut method for the evaluation of S and N for a      *
* stripping  column with recovery, using Douglas,    *
*Keller and Perry heuristics,  and Kremser equation.*
*The main goal is called 'STRIPPING' this version    *
*needs 1 database file: containing the flows and     *
*Henry law constants. 2/4                            *
*******************************************************/
:-segment(strseg).
:-public stripping/6:far.
:-visible retrieveb/3.
:-extrn
            append/3:far,
            append_v/3:far,
            component_flow/3:far,
            component_flow_list/4:far,
            component_flow_set/3:far,
            decide_difficult_or_easy/2:far,
            delete_element/3:far,
            delete_elements/3:far,
            equal/2:far,
            erase_term/2:far,
            ffactor/3:far,
            first_prod/2:far,
            flow_set/3:far,
            flow_set_c/3:far,
            get_split_flow/4:far,
        largest_flow/3:far,
            last_prod/2:far,
            list_member/2:far,
            max/3:far,
            max_flist/2:far,
            max_flow/3:far,
            member/2:far,
            pause/0:far,
            pause_escape/0:far,
            positive_component_flow_list/2:far,
            print_cam_driver/3:far,
            product_flow/3:far,
            product_flow_match/4:far,
            product_flow_set/3:far,
            quicksort/2:far,
            reverse/2:far,
            selected_positive_component_flow_list/3:far,
            set_above/3:far,
            set_below/3:far,
            split_above/4:far,
            split_below/4:far,
            sub_list/2:far,
            sum_component_flow/3:far,
            sum_flist/2:far,
            two_highest_flows/3:far,
                        make_hlist_from_clist/2:far,
                        sort_a_list_by_hhlist/3:far,
                        make_henlist_from_list/3:far,
```

```
                print_specifications/3:far,
                identify_desired_separation/3:far,
                input_henrys_law/1:far,
                d_over_b/3:far,
            actual_flows/4:far,
            write_round/2:far.


stripping(List,CList,TopList,TopCList,BotList,BotCList):-

        /*****************************************************
        *Instantiate the knowledge of the database to the   *
        *variable used in the module. Then transform and    *
        *create lists for the CAM representation            *
        *****************************************************/
        [!
        recorded(data,optimal_S(Sopt),_),
        recorded(data,limit_S(Slg),_),
        Sst is Slg+1,recorda(data,stage_S(Sst),_), %initialization
        removeallb(strip),
        get_split_flow(List,L,CList,0),
        cls,
        write('                          ┌──────────┐ '),nl,
        write('                          | Stripping | '),nl,
        write('                          └──────────┘ '),nl,
        nl,nl,nl,
        write('  Lean Gas being added to stream ....  '),nl,
        write('  Component Assignment Matrix being recalculated'),nl,
        write('    with equilibrium ratios from Henry''s law.'),nl,nl,
        recorded(data,lean_gas(LG),_),
        make_hlist_from_clist(CList,HList),
        quicksort(HList,HHCList),
        sort_a_list_by_hhlist(CList,HHCList,CLList),
        make_henlist_from_list(List,CList,HL),
        quicksort(HL,HHL),
        sort_a_list_by_hhlist(List,HHL,LList),
        append([lean_gas],LList,NList),
        append([LG],CLList,NCList),
        nl,nl,write('  Calculation complete. '),pause,
        cls,current_window(_,two),
        print_cam_driver(LList,CLList,lean_gas),

        /*****************************************************
        *Identify the desired separation and calculate the  *
        *initial value for the invcrement of the volume.    *
        *****************************************************/

        cls,current_window(_,four),
        cls,
        write('                          ┌──────────┐ '),nl,
        write('                          | Stripping | '),nl,
        write('                          └──────────┘ '),nl,
        identify_desired_separation(NList,TopList,BotList),
        recorded(data,key_component(s,KC),_),
        recorded(data,henry(KC,KConst),_),
        calc_min_V_s(Smin,List,CList,L,KConst,TopList,BotList)
         !],

        /*****************************************************
        * Start searching for a solution. Kremser is the    *
```

```
        * main functor of the module.                    *
        ****************************************************/

    for(Smin,Slg,S),
    erase_term(data,flow(lean_gas,_,_)),
        [!
    V is S*L/KConst,
        kremser_s(LList,NCList,[LG],V,L,KConst,TopList,BotList),
        current_window(_,four)
        !],
    fail.

        /*****************************************************
        *When each loop is finished, it fails.              *
        *The Stripping clause continues in the following    *
        *code.                                              *
        *When a solution is found, it is asserted in the    *
        *database of the whole program. Then the recovery   *
        *column is calculated. The absorption module is     *
        *entered or the recovery is assumed feasible and    *
        *the split sharp (choice of the user)               *
        *****************************************************/

stripping(List,CList,_,_,_,_):-
        choose_sep_s,
        recorded(sep,dseparator(Num,_,_,_,_),_),
        NNum is Num + 1,
        recorded(data,sep_CS(X),_),
        retrieveb(strip,X,dseparator_s(N,S,NCList,TopList,TopAFList,BotList,BotAFList)),
        recorda(sep,dseparator(NNum,NCList,TopAFList,NCList,BotAFList),_),
        recovery_s,
        cls,nl,nl,
        write(' The stripping and lean gas column '),nl,
        write(' are specified the results are: '),nl,nl,
        write('  Column        Separation'),nl,
        write(' ------        ----------'),nl,nl,
        write('  Stripper      '),write(TopList),write('/'),write(BotList),nl,
        delete_element(lean_gas,TopList,RTop),
        write('  Recovery      '),write(RTop),write('/'),write([lean_gas]),
        pause.

stripping(List,CList,_,_,_,_):-
        tmove(5,1),
        write('As no solution has been found the module ends here.'),
        pause.



/*************************************************************************
* End of the main part of the module. The following functors are used *
* by it.                                                               *
*************************************************************************/

/*****************************************************
* This clause calculates the minimum steam molar     *
* flowrate according to the method explained in      *
* "Distillation", Van Winkle.                        *
*****************************************************/
calc_min_V_s(Smin,List,CList,L,KConst,TopList,BotList):-
        recorded(data,lean_gas(a),_),
        write('Do you know the total volumetric flowrate'),
```

```
        send_menu_msg(activate(yes_no,(17,32)),Decision),!,
        Decision=yes,nl,nl,
        write('Enter this flowrate'),
        read(VolFlow),
        delete_element(lean_gas,TopList,Top_no_LG_List),
        sum_component_flow(CList,TopFlow,Top_no_LG_List),
        sum_component_flow(CList,BotFlow,BotList),
        VolFlowTop is TopFlow/VolFlow*100,
        VolFlowBot is BotFlow/VolFlow*100,
        recorded(data,key_component(s,KC),_),
        sum_component_flow([KC],KCFlow,List),
        sum_component_flow([KC],KCTopFlow,TopList),
        ES is KCTopFlow/KCFlow*100,
        Vmin is ES*(VolFlowBot+VolFlowTop)/KConst-VolFlowTop,
        Smin is KConst*Vmin/L.

    /*****************************************************
    * If the lean gas is not steam the minimum strip-  *
    * ping factor is assumed to be 0.4                  *
    *****************************************************/

calc_min_V_s(Smin,_,_,_,_,_,_):-
        recorded(data,minimum_S(Smin),_).

/*****************************************************
* This clause is equivalent to a loop. It enables  *
* the program to increment the flowrate of the     *
* lean gas.                                         *
*****************************************************/

for(X,Y,X):-X>Y,!.
for(Y,X,Y):-true.
for(X,Y,Z):-inc2(X,X1),for(X1,Y,Z).

inc2(X,X1):-recorded(data,step(Incstep),_),X1 is X+Incstep.

/*****************************************************
*            KREMSER CLAUSE                         *
*****************************************************/

kremser_s(List,CList,[LG],V,L,KConst,TopList,BotList):-
        recorded(data,henry(KC,KConst),_),
        recorded(data,limit_S(Slg),_),
        recorded(data,optimal_N(Nopt),_),
        recorded(data,optimal_S(Sopt),_),
        S is (KConst * V) / L,
%       write(' L='),write(L),write(' V='),write(V),write('S='),write(S),
        component_flow(List,KC,KCFlow),
        Xi_in is KCFlow / L,
        component_flow(BotList,KC,KCBFlow),
        sum_component_flow(CList,TBFlow,BotList),
        calc_Xi_out(KCFlow,KCBFlow,TBFlow,Xi_out),
        recorded(data,mole_fraction(lean_gas,KC,YKC_in),_),
        !,
        calc_number_stage_s(S,Xi_in,Xi_out,KConst,YKC_in,N),
%       write(' N='),write(N),nl,
        !,
        check_every_s(List,CList,V,RV,L,KConst,N,TopList,BotList,DBact,DBspec),
        tmove(5,1),
        write(' Stripping appears feasible for this separation.'),nl,
        write('  Component recovery estimates are as follows:'),nl,nl,
```

```
        write('                Specified      Estimated'),nl,
        write(' Component        d/b             d/b'),nl,
        write(' ----------     --------      --------'),nl,
        print_specifications(CList,DBact,DBspec),pause,cls,
        lean_gas_flow_record(List,LG,RV),
        current_window(_,five),
        cls,
        write('                              ┌──────────────┐ '),nl,
        write('                              | Stripping | '),nl,
        write('                              └──────────────┘ '),nl,
        nl,nl,nl,
        write(' The heuristic analysis is as follows:'),nl,nl,
        write('    The stripping factor (kV/L) is: '),
        W2 is round(S,2),write(W2),nl,
        write('    Heuristic optimum for stripping factor: S = '),
        write(Sopt),nl,nl,
        write('    The number of theoretical stages: '),
        W1 is round(N,1),write(W1),nl,
        write('    Heuristic optimum number of stages:'),write(Nopt),nl,nl,
        current_window(_,seven),cls,
        diag_s(N,V,NewV,RV,KConst,L),
        current_window(_,five),
        calc_dev(DBact,DBspec,Dev),
        store_sep_s(Dev,N,S,List,CList,TopList,BotList,DBact).

/*******************************************************
* This clause calculates the molar fractions in the*
* output liquid. If xi_out is 0, N becomes infinite*
* then we assume xi_out to be 2%.                  *
*******************************************************/


calc_Xi_out(KCFlow,KCBFlow,TBFlow,Xi_out):-
        KCBFlow is 0,
        Xi_out is 0.02 * KCFlow / TBFlow.

calc_Xi_out(_,KCBFlow,TBFlow,Xi_out):-
        Xi_out is KCBFlow / TBFlow.

/*******************************************************
* This clause calculates the number of theoretical *
* stages, it takes care of the case S=1            *
*******************************************************/

calc_number_stage_s(S,Xi_in,Xi_out,KConst,YKC_in,N):-
    S=\=1,
    R is (Xi_in-Xi_out)/(Xi_in-KConst*YKC_in),
        (R-S)/(R-1) > 0,
    N is log((R-S)/(R-1))/log(S) - 1.0.

calc_number_stage_s(S,Xi_in,Xi_out,KConst,YKC_in,N):-
        S=:=1,
        N is (Xi_in-Xi_out)/(Xi_out-YKC_in/KConst).

/*******************************************************
* This functor succeeds only when N>=Nopt, and     *
*     Smin < S < Slg (<Sst for which N<Nopt)       *
* A feasible separation is added to the top of the *
* database.                                        *
*******************************************************/
```

```
check_every_s(List,CList,V,RV,L,KConst,N,TopList,BotList,DBact,DBspec):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N >=Nopt,
        check_S_opt(V,L,KConst,NewV),
        !,
        check_in_spec_s(List,CList,V,NewV,RV,L,KConst,N,TopList,BotList,DBact,DBspec).

        /*****************************************************
        * This functor succeeds only when N<Nopt,equivalent*
        * to      Sst < S and if  S < Slg                   *
        * A feasible separation is added to the bot of the *
        * database, then the first separation on the data- *
        * base is the 'least worse' because N closest to   *
        * Nopt and V minimum.                               *
        *****************************************************/

check_every_s(List,CList,V,RV,L,KConst,N,TopList,BotList,DBact,DBspec):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N<Nopt,
        recorded(data,limit_S(Slg),_),
        S is KConst*V/L,
        store_Sst(S),
        sum_component_flow(CList,TBFlow,BotList),
        !,
        components_within_spec_s(List,CList,V,L,N,TopList,BotList,TBFlow,DBact,DBspec),
        RV is V.

store_Sst(S):-
        [!recorded(data,stage_S(Sst),Ref)!],
        S < Sst,        % Sst is initialized to Slg
        NewSst is S, replace(Ref,stage_S(NewSst)).
store_Sst(S).



/*****************************************************
* These clauses verify if for a given V, the       *
* corresponding S is or not less than S optimal.   *
*****************************************************/


check_S_opt(V,L,KConst,NewV):-
      S is V*KConst/L,
      [!recorded(data,optimal_S(Sopt),_)!],
      S < Sopt,
      NewV is Sopt*L/KConst.

check_S_opt(V,L,KConst,NewV):-
        S is V*KConst/L,
        [!recorded(data,optimal_S(Sopt),_)!],
      S >=Sopt,
      NewV is V.

/*****************************************************
* This clause succeeds if in spec. case 1 and 3.   *
*****************************************************/
check_in_spec_s(List,CList,_,NewV,RV,L,_,N,TopList,BotList,DBact,DBspec):-
        sum_component_flow(CList,TBFlow,BotList),

components_within_spec_s(List,CList,NewV,L,N,TopList,BotList,TBFlow,DBact,DBspec),
        RV is NewV.
```

```
    /*******************************************************
    * This clause succeeds if not in with current V    *
    *  and in spec with this imposed V=Sopt*... Case 2 *
    *******************************************************/

check_in_spec_s(List,CList,V,NewV,RV,L,KConst,N,TopList,BotList,DBact,DBspec):-
      [!recorded(data,optimal_S(Sopt),_)!],
          NewV*KConst/L =:= Sopt,
      sum_component_flow(CList,TBFlow,BotList),

components_within_spec_s(List,CList,V,L,N,TopList,BotList,TBFlow,DBact,DBspec),
      RV is V.


/*******************************************************
* Clause which decides if the d/b's are in the     *
* range ±10% of the specification.                 *
*******************************************************/

components_within_spec_s(_,[],_,_,_,_,_,_,[],[]).
components_within_spec_s(List,[HC|TC],V,L,N,TopList,BotList,TBFlow,[HDa|TDa],[HDs|TDs]):-
          record_comp_flow_in_lean_gas([lean_gas],[HC],V),
          component_flow(TopList,HC,Dspec),
          component_flow(BotList,HC,Bspec),
          d_over_b(Dspec,Bspec,DBspec),
          component_flow(List,HC,FFlow),
          Xi_in_actual is FFlow / L,
          recorded(data,henry(HC,HConst),_),
          S is (HConst * V)/L,
          recorded(data,mole_fraction(lean_gas,HC,Yi_in),_),
          calc_Xi_out_actual(S,Xi_in_actual,HConst,Yi_in,N,FFlow,TFlow,Xi_out_actual),
          Bact is Xi_out_actual * L,
          Dact is FFlow - Bact,
          d_over_b(Dact,Bact,DBact),
          recorded(data,error(Er),_),
          !,
          DBact < (1+Er) * DBspec,
          DBact > (1-Er) * DBspec,
          HDa is DBact,
          HDs is DBspec,
          components_within_spec_s(List,TC,V,L,N,TopList,BotList,TBFlow,TDa,TDs).

/*******************************************************
* Utility clauses                                  *
*******************************************************/
lean_gas_flow_record([],LG,V):-
          recordz(data,flow(lean_gas,LG,V),_).
lean_gas_flow_record([H|T],LG,V):-
          recordz(data,flow(H,LG,0),_),
          lean_gas_flow_record(T,LG,V).

record_comp_flow_in_lean_gas([lean_gas],[],_).
record_comp_flow_in_lean_gas([lean_gas],[HC|TC],V):-
          recorded(data,mole_fraction(lean_gas,HC,Y),_),
          VX is V*Y,
          recorda(data,flow(lean_gas,HC,VX),_),
          record_comp_flow_in_lean_gas([lean_gas],TC,V).

/*******************************************************
* Calculation of xi_out requires many different    *
```

```
* cases to avoid infinite values or ln(x<=0).      *
*****************************************************/

calc_Xi_out_actual(S,Xi_in_actual,HConst,Yi_in,N,_,_,Xi_out_actual):-
        abs(S-1)>1e-8,
        abs((N+1)*ln(S))<10,
        Xi_out_actual is Xi_in_actual-(Xi_in_actual-Yi_in/HConst)*(exp((N+1)*ln(S))-S)/(exp((N + 1)* ln(S)) - 1).

calc_Xi_out_actual(S,Xi_in_actual,HConst,Yi_in,N,FFlow,TBFlow,Xi_out_actual):-
        abs(S-1)>1e-8,
        (N+1)*ln(S)>0,
        Xi_out_actual is Yi_in/HConst+0.02*FFlow/TBFlow.

calc_Xi_out_actual(S,Xi_in_actual,HConst,Yi_in,N,_,_,Xi_out_actual):-
        abs(S-1)>1e-8,
        Xi_out_actual is Xi_in_actual-(Xi_in_actual-Yi_in/HConst)*S.

calc_Xi_out_actual(S,Xi_in_actual,HConst,Yi_in,N,_,_,Xi_out_actual):-
        Xi_out_actual is (Xi_in_actual+Yi_in/HConst*N)/(N+1).

/*****************************************************
* Diagnose clauses :                                 *
* 4 different cases exists but they are not all      *
* feasible or pratical                               *
*****************************************************/

diag_s(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
        recorded(data,optimal_S(Sopt),_),
        NewV is Sopt*L/KConst,
        RV is NewV,
        write(' V is imposed to a   '),nl,
        write(' higher optimum value.'),nl,
        write('   N is minimized.     '),nl,
        write('       [case 1]        '),pause.

diag_s(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
    [!recorded(data,optimal_S(Sopt),_)!],
        NewV is Sopt*L/KConst,
        RV is V,
        RV*KConst/L < Sopt,
        write(' V is lower than the '),nl,
        write('  optimum value N is '),nl,
        write('   higher than the   '),nl,
        write('        optimum      '),nl,
        write('        [case 2]     '),pause.

diag_s(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
      N>=Nopt,
        NewV is V,
        [!recorded(data,limit_S(Slg),_)!],
        RV < Slg*L/KConst,
        write(' V is higher than the'),nl,
        write(' optimum value.      '),nl,nl,
```

```
        write('    The minimal      '),nl,
        write(' solution for N is   '),nl,
        write(' greater than    the '),nl,
        write(' optimum of '),write(Nopt),write('.'),nl,nl,
        write('          [case 3]'),pause.


diag_s(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
        N<Nopt,
        write(' V is much higher than '),nl,
        write(' the optimum value.    '),nl,
        write(' N is lower  than  the '),nl,
        write(' optimum (Nopt='),write(Nopt),write('). '),nl,
        write(' The  separation  is   '),nl,
        write(' unpractical  because  '),nl,
        write(' N is too low, and not'),nl,
        write(' economical because V  '),nl,
        write(' is too high:  S>Sst.'),
        write('          [case 4]'),pause.

/*******************************************************
* Calculates the average deviation in the doverb 's*
* to be used in the CESS.                              *
*******************************************************/

calc_dev(DBact,DBspec,Dev):-
        sum_dev(DBact,DBspec,Sum),
        length(DBspec,L),
        Dev is Sum/L.

sum_dev([],[],0).
sum_dev([HA|TA],[HB|TB],Sum):-
        sum_dev(TA,TB,SumTail),
        Sum is (abs(HA-HB)/HA+SumTail).

/*******************************************************
* Intermediate storage of the separations obtained *
* before final choice and end of the module        *
*******************************************************/

store_sep_s(Dev,N,S,List,CList,TopList,BotList,DBact):-
        recorded(data,lean_gas(LG),_),
        append([lean_gas],List,NList),
        append([LG],CList,NCList),
        component_flow_set(TopList,NCList,TopFList),
        component_flow_set(BotList,NCList,BotFList),
        component_flow_set(List,CList,FList),
        actual_flows(FList,DBact,Top_no_lg_AFList,Bot_no_lg_AFList),
        recorded(data,flow(lean_gas,LG,V),_),
        append([V],Top_no_lg_AFList,TopAFList),
        append([0],Bot_no_lg_AFList,BotAFList),
        calc_CS(Dev,N,S,CS),
        recordb(strip,CS,dseparator_s(N,S,NCList,TopList,TopAFList,BotList,BotAFList)).

/*******************************************************
* To decide which separation is the best we calcu- *
* late the Coefficient for Stripping.              *
*       It is a function of the average deviation*
* ,the number of stages and the stripping factor   *
```

```
*****************************************************/

calc_CS(Dev,N,S,CS):-
        recorded(data,optimal_N(Nopt),_),
        recorded(data,optimal_S(Sopt),_),
        CSD is -ln(Dev),
        CSN is -ln( N*N/2-(Nopt-2)*N+(Nopt-2)*(Nopt-2)/2+1 )+2*sqrt(2)*atan( (N-(Nopt-2))/sqrt(2) ),
        CSS is -ln( S*S/2-(Sopt+2)*S+(Sopt+2)*(Sopt+2)/2+1 )-2*sqrt(2)*atan( (S-(Sopt+2))/sqrt(2) ),
        CS is round(exp(CSD+CSN+CSS),0).


/*****************************************************
* We need to choose the best separation among      *
* those generated previously and stored in         *
* dseparator_s.                                    *
*****************************************************/


choose_sep_s:-
        [!recorded(data,minimum_S(Smin),_),
        recorded(data,stage_S(Sst),_),
        recorded(data,limit_S(Slg),_),
        recorded(data,optimal_S(Sopt),_),
        write(' The range of Stripping Factor is:'),nl,
        write(Smin),write(' - '),write(Slg),nl,
        current_window(CW,seven),cls,
        diag_s_range(Smin,Sopt,Sst,Slg),
        current_window(_,CW)!],
        findall(X,retrieveb(strip,X,_),ListX),
        quicksort(ListX,[HX|TX]),
        cls,nl,
        write('      The Best Separation According to the Analysis is:'),
        write_sep_s([HX|TX]).
choose_sep_s:-fail.

write_sep_s([]).
write_sep_s([HX|TX]):-
        retrieveb(strip,HX,dseparator_s(N,S,NCList,TopList,TopAFList,BotList,BotAFList)),
        nl,write(' The stripping column '),
    write(' is specified the results are: '),nl,nl,
    write('   Column        Separation'),nl,
    write(' ---------      ----------'),nl,nl,
    write('  Stripper      '),write(TopList),write('/'),write(BotList),
        nl,nl,
        write('Components:'),write(NCList),nl,
        write('Top       :'),
        write_round(TopAFList,1),nl,
        write('Bot       :'),write_round(BotAFList,1),nl,nl,nl,
        write(' The design parameters are:'),nl,
        NR is round(N,2),
        write(' Number of theoretical stages:'),write(NR),nl,nl,
        write(' Stripping factor:'),write(S),nl,nl,
        pause,
        write('Do you Accept this one, if not the other solutions willbe displayed'),
        send_menu_msg(activate(yes_no2,(10,60)),Decision),!,
        solve_choice_s(Decision,[HX|TX]).

solve_choice_s(Decision,[HX|TX]):-
        Decision=yes,
        recordz(data,sep_CS(HX),_).
```

```
solve_choice_s(Decision,[HX|TX]):-
        Decision=no,cls,write_sep_s(TX).

/*****************************************************
* Here, the program diagnoses if the range of stri-*
* pping factors entered by the user is likely to   *
* contain all the possible solutions.              *
*****************************************************/

                    /*case1*/
diag_s_range(Smin,Sopt,Sst,Slg):-
        Slg<Sopt,
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        write(' If you want to use an '),nl,
        write(' optimal value for S,  '),nl,
        write(' you must not have the '),nl,
        write(' limit  Slg  smaller   '),nl,
        write(' than the optimal value'),nl,
        write(' Sopt:'),write(Slgt),write(' > '),write(Soptt),write('.'),
        pause.

                    /*case2*/
diag_s_range(Smin,Sopt,Sst,Slg):-
        Smin>Slg,
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        write(' The upper limit Slg is'),nl,
        write(' smaller than the lower'),nl,
        write(' limit Smin:'),write(Smint),write(' < '),write(Slgt),write('.'),nl,
        write(' You must have Smin<Slg'),nl,
        write(' for any calculation to'),nl,
        write(' occur.'),pause.

                    /*case3*/
diag_s_range(Smin,Sopt,Sst,Slg):-
        Smin<Sopt,
        Sopt<Sst,
        Sst<Slg,
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        [!recorded(data,optimal_N(Nopt),_)!],
        Noptt is round(Nopt,2),
        write(' No more solutions are '),nl,
        write('      expected in     '),nl,
        write('    the studied range. '),
        pause.
                    /*case4*/
diag_s_range(Smin,Sopt,Sst,Slg):-
        Smin<Sopt,
        Sopt<Slg,
        [!recorded(data,optimal_N(Nopt),_)!],
        Slg<Sst,   %Sst actually does not exist because Slg is to small to reach it, Sst is initialized to Slg+1
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        Noptt is round(Nopt,2),
        write(' Solutions may exist   '),nl,
        write(' for S>'),write(Slgt),write('. With a'),nl,
        write(' number of stages still'),nl,
        write(' higher than '),write(Noptt),write('. You'),nl,
        write(' should increase Slg.'),
        pause.

                    /*case5*/
diag_s_range(Smin,Sopt,Sst,Slg):-
```

```
        Smin<Sst,
        Sst<Sopt,
        Sopt<Slg,
        [!recorded(data,optimal_N(Nopt),_)!],
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        Noptt is round(Nopt,2),
        write(' The  optimal  E ,  '),nl,
        write(' Soptt  =  '),write(Soptt),write(', should'),nl,
        write(' be  smaller  than  '),nl,
        write(' Sst='),write(Sstt),write(' because'),nl,
        write(' Sopt ought to be where'),nl,
        write(' N is greater than '),write(Noptt),write('.'),
        pause.

                /*case6*/
diag_s_range(Smin,Sopt,Sst,Slg):-
        Smin>=Sopt,
        Smin=:=Sst, %the first value of E will instantiate Sst
        recorded(data,optimal_N(Nopt),_),
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        Noptt is round(Nopt,2),
        write(' The  initial  value,'),nl,
        write(' Smin='),write(Smint),write(', is too high.'),
        write(' Only solutions with N '),nl,
        write(' smaller  than  '),write(Noptt),nl,
        write(' will be generated.'),nl,
        write(' Worse, the optimal E  '),nl,
        write(' Sopt='),write(Soptt),write(', is too'),nl,
        write(' small to be studied. '),nl,
        write(' It should be greater '),nl,
        write(' than '),write(Smint),write('.'),
        pause.

                /*case7*/
diag_s_range(Smin,Sopt,Sst,Slg):-
        Smin>=Sopt,
        Smin<Sst,
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        write(' The  minimum  value  '),nl,
        write(' Smin='),write(Smint),write(' should be '),nl,
        write(' smaller if you want  '),nl,
        write(' to use the optimal   '),nl,
        write(' value Sopt='),write(Soptt),write('.'),nl,
        pause.

                /*case8*/
diag_s_range(Smin,Sopt,Sst,Slg):-
        Smin<Sopt,
        Smin =:= Sst, %the first value of E will instantiate Sst
        recorded(data,optimal_N(Nopt),_),
        Smint is round(Smin,2),Soptt is round(Sopt,2),Sstt is round(Sst,2),Slgt is round(Slg,2),
        Noptt is round(Nopt,2),
        write(' The initial value of E'),nl,
        write(' Smin='),write(Smint),write(' is too high.'),nl,
        write(' Only solutions with N'),nl,
        write(' smaller than '),write(Noptt),write(' will '),nl,
        write(' be generated.'),
        pause.

diag_s_range(_,_,_,_):-
        write('No diagnosis. Error.'),pause.
```

```
/*****************************************************
* Recovery clause to separate the lean gas from     *
* the top. Call of the Absorb module to be imple-   *
* -mented                                           *
*****************************************************/

recovery_s:-
        write('The recovery of the lean gas is supposed to be feasible and sharp'),nl,
        recorded(sep,dseparator(Rum,NCList,TopAFList,NCList,BotAFList),_),
        recorded(data,lean_gas(LG),_),
        RRum is Rum + 1,
        recorda(bypass,dstreambypass(RRum,'none',[],[]),_),
        recorda(sep,dstreamin(RRum,NCList,TopAFList),_),
        [V|TF] = TopAFList,
        recorda(data,flow(lean_gas,LG,V),_),
        delete_element(S,NCList,CList),
        record_comp_flow_in_lean_gas([lean_gas],CList,V),
        append([0],TF,BotRFList), %bot flows of recovery column
        component_flow_set([lean_gas],NCList,TopRFList),
        recorda(sep,dseparator(RRum,NCList,TopRFList,NCList,BotRFList),_),
        write(' The results for the RECOVERY  COLUMN #'),write(RRum),
        write(' are now available').
```

LIQUID-LIQUID EXTRACTION MODULE (LLE.ARI)

```
/*****************************************************
*LLE  MODULE --       This module implements a short*
*cut method for the evaluation of E and N for a     *
* counter-current liquid-liquid extraction column.  *
* It uses heuristics form Keller, Perry and         *
* heuristics,  and Kremser equation.                *
*The main goal is called 'lle' this version needs   *
*needs 1 database file: containing the flows and    *
*Henry law constants. 1/30 fix the diag clauses     *
*****************************************************/

:-segment(lleseg).
:-public lle/6:far.
:-visible retrieveb/3.
:-extrn
            append/3:far,
            append_v/3:far,
            component_flow/3:far,
            component_flow_list/4:far,
            component_flow_set/3:far,
            decide_difficult_or_easy/2:far,
            delete_element/3:far,
            delete_elements/3:far,
            equal/2:far,
            erase_term/2:far,
            ffactor/3:far,
            first_prod/2:far,
            flow_set/3:far,
            flow_set_c/3:far,
            get_split_flow/4:far,
      largest_flow/3:far,
            last_prod/2:far,
            list_member/2:far,
            max/3:far,
            max_flist/2:far,
            max_flow/3:far,
            member/2:far,
            pause/0:far,
            pause_escape/0:far,
            positive_component_flow_list/2:far,
            print_cam_driver/3:far,
            product_flow/3:far,
            product_flow_match/4:far,
            product_flow_set/3:far,
            quicksort/2:far,
            reverse/2:far,
            selected_positive_component_flow_list/3:far,
            set_above/3:far,
            set_below/3:far,
            split_above/4:far,
            split_below/4:far,
            sub_list/2:far,
            sum_component_flow/3:far,
            sum_flist/2:far,
            two_highest_flows/3:far,
                        make_hlist_from_clist/2:far,
```

```
                sort_a_list_by_hhlist/3:far,
                make_henlist_from_list/3:far,
                print_specifications/3:far,
                identify_desired_separation/3:far,
                input_henrys_law/1:far,
                d_over_b/3:far,
            actual_flows/4:far,
            write_round/2:far.

lle(List,CList,TopList,TopCList,BotList,BotCList):-

        /*******************************************************
        * Instantiate the knowledge of the database to the  *
        * variable used in the module. Then transform and   *
        * create lists for the CAM representation           *
        *******************************************************/
        [!
        recorded(data,optimal_E(Eopt),_),
        recorded(data,limit_E(Esl),_),
        Est is Esl+1,recorda(data,stage_E(Est),_), %initialization
        removeallb(lle),
        get_split_flow(List,L,CList,0),
        cls,
        write('                    ┌──────────────────────────┐'),nl,
        write('                    | Liquid-Liquid Extraction |'),nl,
        write('                    └──────────────────────────┘'),nl,
        write('  Lean Gas being added to stream ....  '),nl,
        write('  Component Assignment Matrix being recalculated'),nl,
        write('    with equilibrium ratios from Henry`s law.'),nl,nl,
        recorded(data,solvent(S),_),
        make_hlist_from_clist(CList,HList),
        quicksort(HList,HHCList),
        sort_a_list_by_hhlist(CList,HHCList,CLList),
        make_henlist_from_list(List,CList,HL),
        quicksort(HL,HHL),
        sort_a_list_by_hhlist(List,HHL,LList),
        append([solvent],LList,NList),
        append([S],CLList,NCList),
        nl,nl,write('  Calculation complete. '),pause,
        cls,current_window(_,two),
        print_cam_driver(LList,CLList,solvent),

        /*******************************************************
        *Identify the desired separation and calculate the *
        *initial value for the invcrement of the volume.   *
        *******************************************************/

        cls,current_window(_,four),
        cls,
        write('                    ┌──────────────────────────┐'),nl,
        write('                    | Liquid-Liquid Extraction |'),nl,
        write('                    └──────────────────────────┘'),nl,
        nl,nl,
        identify_desired_separation(NList,TopList,BotList),
        recorded(data,key_component(lle,KC),_),
        recorded(data,henry(KC,KConst),_),
        calc_min_V_lle(Emin,List,CList,L,KConst,TopList,BotList)
        !],

        /*******************************************************
        * Start searching for a solution. Kremser is the    *
```

```
        * main functor of the module.                    *
        ****************************************************/
        for(Emin,Esl,E),
        erase_term(data,flow(solvent,_,_)),
        [!
    V is E*L/KConst,
        kremser_lle(LList,NCList,[S],V,L,KConst,TopList,BotList),
        current_window(_,four)
        !],
    fail.


        /****************************************************
        * When each loop is finished, it fails.            *
        * The LLE clause continues in the following        *
        * code.                                            *
        * When a solution is found, it is recorded in the  *
        * database of the whole program. Then the recovery *
        * column is calculated. The absorption module is   *
        * entered or the recovery is assumed feasible and  *
        * the split sharp (choice of the user)             *
        ****************************************************/

lle(List,CList,_,_,_,_):-
        choose_sep_lle,
        recorded(sep,dseparator(Num,_,_,_,_),_),
        NNum is Num + 1,
        recorded(data,sep_CE(X),_),
        retrieveb(lle,X,dseparator_lle(N,E,NCList,TopList,TopAFList,BotList,BotAFList)),
        recorda(sep,dseparator(NNum,NCList,TopAFList,NCList,BotAFList),_),
        recovery_lle,
        cls,nl,nl,
        write(' The lle and le an gas column '),nl,
        write(' are specified the results are: '),nl,nl,
        write('   Column        Separation'),nl,
        write(' ---------      ----------'),nl,nl,
        write('  LLE      '),write(TopList),write('/'),write(BotList),nl,
        delete_element(solvent,TopList,RTop),
        write('  Recovery      '),write(RTop),write('/'),write([solvent]),
        pause.

lle(List,CList,_,_,_,_):-
        tmove(5,1),
        write('As no solution has been found the module ends here.'),
        pause.

/*********************************************************************
* End of the main part of the module. The following functors areused *
* by it.                                                             *
*********************************************************************/
/****************************************************
* This clause calculates the minimum steam molar   *
* flowrate according to the method explained in    *
* "Distillation", Van Winkle.                      *
****************************************************/
calc_min_V_lle(Emin,List,CList,L,KConst,TopList,BotList):-
        recorded(data,solvent(a),_),
        write('Do you know the total volumetric flowrate'),
        send_menu_msg(activate(yes_no,(17,32)),Decision),!,
        Decision=yes,nl,nl,
        write('Enter this flowrate'),
```

```
        read(VolFlow),
        delete_element(solvent,TopList,Top_no_S_List),
        sum_component_flow(CList,TopFlow,Top_no_S_List),
        sum_component_flow(CList,BotFlow,BotList),
        VolFlowTop is TopFlow/VolFlow*100,
        VolFlowBot is BotFlow/VolFlow*100,
        recorded(data,key_component(s,KC),_),
        sum_component_flow([KC],KCFlow,List),
        sum_component_flow([KC],KCTopFlow,TopList),
        ES is KCTopFlow/KCFlow*100,
        Vmin is ES*(VolFlowBot+VolFlowTop)/KConst-VolFlowTop,
        Emin is KConst*Vmin/L.

        /*******************************************************
        * If the lean gas is not steam the minimum lle     *
        * factor is assumed to be 1.3                      *
        *******************************************************/

calc_min_V_lle(Emin,_,_,_,_,_,_):-
        recorded(data,minimum_E(Emin),_).

/*******************************************************
* This clause is equivalent to a loop. It enables  *
* the program to increment the flowrate of the     *
* lean gas.                                        *
*******************************************************/

for(X,Y,X):-X>Y,!.
for(Y,X,Y):-true.
for(X,Y,Z):-inc2(X,X1),for(X1,Y,Z).

inc2(X,X1):-recorded(data,step(Incstep),_),X1 is X+Incstep.

/*******************************************************
*            KREMSER CLAUSE                    *
*******************************************************/

kremser_lle(List,CList,[S],V,L,KConst,TopList,BotList):-
        recorded(data,henry(KC,KConst),_),
        recorded(data,limit_E(Esl),_),
        recorded(data,optimal_E(Eopt),_),
        recorded(data,optimal_N(Nopt),_),
        E is (KConst * V) / L,
%       write(' L='),write(L),write(' V='),write(V),write('E='),write(E),
        component_flow(List,KC,KCFlow),
        Xi_in is KCFlow / L,
        component_flow(BotList,KC,KCBFlow),
        sum_component_flow(CList,TBFlow,BotList),
        calc_Xi_out(KCFlow,KCBFlow,TBFlow,Xi_out),
        recorded(data,mole_fraction(solvent,KC,YKC_in),_),
        !,
        calc_number_stage_lle(E,Xi_in,Xi_out,KConst,YKC_in,N),
%       write(' N='),write(N),nl,
        !,
        check_every_lle(List,CList,V,RV,L,KConst,N,TopList,BotList,DBact,DBspec),
        tmove(5,1),
        write(' LLE appears feasible for this separation.'),nl,
        write('  Component recovery estimates are as follows:'),nl,nl,
        write('                Specified      Estimated'),nl,
        write('  Component        d/b            d/b'),nl,
        write(' ————————    ————————    ————————'),nl,
```

```
        print_specifications(CList,DBact,DBspec),pause,cls,
        solvent_flow_record(List,S,RV),
        current_window(_,five),
        cls,
        write('                    ┌──────────────────────────────┐'),nl,
        write('                    | Liquid-Liquid Extraction |'),nl,
        write('                    └──────────────────────────────┘'),nl,
        nl,nl,nl,
        write(' The heuristic analysis is as follows:'),nl,nl,
        write(' The LLE factor (kV/L) is: '),
        W2 is round(E,2),write(W2),nl,
        write('     Heuristic optimum for LLE factor:  E = '),
        write(Eopt),nl,nl,
        write('     The number of theoretical stages: '),
        W1 is round(N,1),write(W1),nl,
        write('     Heuristic optimum number of stages:'),write(Nopt),nl,nl,
        current_window(_,seven),cls,
        diag_lle(N,V,NewV,RV,KConst,L),
        current_window(_,five),
        calc_dev(DBact,DBspec,Dev),
        store_sep_lle(Dev,N,E,List,CList,TopList,BotList,DBact).

/*******************************************************
* This clause calculates the molar fractions in the*
* output liquid. If xi_out is 0, N becomes infinite*
* then we assume xi_out to be 2%.                  *
*******************************************************/


calc_Xi_out(KCFlow,KCBFlow,TBFlow,Xi_out):-
        KCBFlow is 0,
        Xi_out is 0.02 * KCFlow / TBFlow.

calc_Xi_out(_,KCBFlow,TBFlow,Xi_out):-
        Xi_out is KCBFlow / TBFlow.

/*******************************************************
* This clause calculates the number of theoretical *
* stages, it takes care of the case E=1            *
*******************************************************/

calc_number_stage_lle(E,Xi_in,Xi_out,KConst,YKC_in,N):-
    E=\=1,
    R is (Xi_in-Xi_out)/(Xi_in-KConst*YKC_in),
        (R-E)/(R-1) > 0,
    N is log((R-E)/(R-1))/log(E) - 1.0.

calc_number_stage_lle(E,Xi_in,Xi_out,KConst,YKC_in,N):-
        E=:=1,
        N is (Xi_in-Xi_out)/(Xi_out-YKC_in/KConst).

/*******************************************************
* This functor succeeds only when N>=Nopt, and     *
*     Emin < E < Esl (<Est for which N<Nopt)       *
* A feasible separation is added to the top of the *
* database.                                        *
*******************************************************/

check_every_lle(List,CList,V,RV,L,KConst,N,TopList,BotList,DBact,DBspec):-
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
```

```
        check_E_opt(V,L,KConst,NewV),
        !,
            check_in_spec_lle(List,CList,V,NewV,RV,L,KConst,N,TopList,BotList,DBact,DBspec).

            /*****************************************************
            * This functor succeeds only when N<Nopt,equivalent*
            * to      Est  <  E and if  E  <  Esl                *
            * A feasible separation is added to the bot of the *
            * database, then the first separation on the data- *
            * base is the 'least worse' because N closest to   *
            * Nopt and V minimum.                               *
            *****************************************************/

check_every_lle(List,CList,V,RV,L,KConst,N,TopList,BotList,DBact,DBspec):-
            [!recorded(data,optimal_N(Nopt),_)!],
            N<Nopt,
        [!recorded(data,limit_E(Esl),_)!],
            E is KConst*V/L,
            store_Est(E),
            sum_component_flow(CList,TBFlow,BotList),
            !,
            components_within_spec_lle(List,CList,V,L,N,TopList,BotList,TBFlow,DBact,DBspec),
        RV is V.

store_Est(E):-
        [!recorded(data,stage_E(Est),Ref)!],
            E  <  Est,        % Est is initialized to Esl
            NewEst is E, replace(Ref,stage_E(NewEst)).
store_Est(E).



/*****************************************************
* These clauses verify if for a given V, the        *
* corresponding E is or not less than E optimal.    *
*****************************************************/


check_E_opt(V,L,KConst,NewV):-
        E is V*KConst/L,
        [!recorded(data,optimal_E(Eopt),_)!],
        E  <  Eopt,
        NewV is Eopt*L/KConst.

check_E_opt(V,L,KConst,NewV):-
            E is V*KConst/L,
            [!recorded(data,optimal_E(Eopt),_)!],
        E >=Eopt,
        NewV is V.

/*****************************************************
* This clause succeeds if in spec. Case 1 and 3     *
*****************************************************/
check_in_spec_lle(List,CList,V,_,RV,L,_,N,TopList,BotList,DBact,DBspec):-
            sum_component_flow(CList,TBFlow,BotList),

components_within_spec_lle(List,CList,V,L,N,TopList,BotList,TBFlow,DBact,DBspec),
            RV is V.

    /*****************************************************
    * This clause succeeds if not in with current V     *
```

```
    * and in spec with this imposed V=Sopt*... Case 2 *
    ****************************************************/

check_in_spec_lle(List,CList,V,NewV,RV,L,KConst,N,TopList,BotList,DBact,DBspec):-
      [!recorded(data,optimal_E(Eopt),_)!],
          NewV is Eopt*L/KConst,
      sum_component_flow(CList,TBFlow,BotList),
      components_within_spec_lle(List,CList,NewV,L,N,TopList,BotList,TBFlow,DBact,DBspec),
      RV is NewV.

/****************************************************
* Clause which decides if the d/b's are in the     *
* range ±10% of the specification.                 *
****************************************************/

components_within_spec_lle(_,[],_,_,_,_,_,_,[],[]).
components_within_spec_lle(List,[HC|TC],V,L,N,TopList,BotList,TBFlow,[HDa|TDa],[HDs|TDs]):-
          record_comp_flow_in_solvent_lle([solvent],[HC],V),
          component_flow(TopList,HC,Dspec),
          component_flow(BotList,HC,Bspec),
          d_over_b(Dspec,Bspec,DBspec),
          component_flow(List,HC,FFlow),
          Xi_in_actual is FFlow / L,
          recorded(data,henry(HC,HConst),_),
          E is (HConst * V)/L,
          recorded(data,mole_fraction(solvent,HC,Yi_in),_),
          calc_Xi_out_actual(E,Xi_in_actual,HConst,Yi_in,N,FFlow,TFlow,Xi_out_actual),
          Bact is Xi_out_actual * L,
          Dact is FFlow - Bact,
          d_over_b(Dact,Bact,DBact),
          recorded(data,error(Er),_),
          !,
          DBact < (1+Er) * DBspec,
          DBact > (1-Er) * DBspec,
          HDa is DBact,
          HDs is DBspec,
          components_within_spec_lle(List,TC,V,L,N,TopList,BotList,TBFlow,TDa,TDs).

/****************************************************
* Utility clauses                                  *
****************************************************/
solvent_flow_record([],S,V):-
          recordz(data,flow(solvent,S,V),_).
solvent_flow_record([H|T],S,V):-
          recordz(data,flow(H,S,0),_),
          solvent_flow_record(T,S,V).

record_comp_flow_in_solvent_lle([solvent],[],_).
record_comp_flow_in_solvent_lle([solvent],[HC|TC],V):-
          recorded(data,mole_fraction(solvent,HC,Y),_),
          VX is V*Y,
          recorda(data,flow(solvent,HC,VX),_),
          record_comp_flow_in_solvent_lle([solvent],TC,V).

/****************************************************
* Calculation of xi_out requires many different    *
* cases to avoid infinite values or ln(x<=0).      *
****************************************************/

calc_Xi_out_actual(E,Xi_in_actual,HConst,Yi_in,N,_,_,Xi_out_actual):-
          abs(E-1)>1e-8,
```

```
        abs((N+1)*ln(E))<10,
        Xi_out_actual is Xi_in_actual-(Xi_in_actual-Yi_in/HConst)*(exp((N+1)*ln(E))-E)/(exp((N + 1)* ln(E)) - 1).

calc_Xi_out_actual(E,Xi_in_actual,HConst,Yi_in,N,FFlow,TBFlow,Xi_out_actual):-
        abs(E-1)>1e-8,
        (N+1)*ln(E)>0,
        Xi_out_actual is Yi_in/HConst+0.02*FFlow/TBFlow.

calc_Xi_out_actual(E,Xi_in_actual,HConst,Yi_in,N,_,_,Xi_out_actual):-
        abs(E-1)>1e-8,
        Xi_out_actual is Xi_in_actual-(Xi_in_actual-Yi_in/HConst)*E.

calc_Xi_out_actual(E,Xi_in_actual,HConst,Yi_in,N,_,_,Xi_out_actual):-
        Xi_out_actual is (Xi_in_actual+Yi_in/HConst*N)/(N+1).

/*****************************************************
* Diagnose clauses :                                 *
* 4 different cases exists but they are not all      *
* feasible or pratical                               *
*****************************************************/

diag_lle(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
        [!recorded(data,optimal_E(Eopt),_)!],
        NewV is Eopt*L/KConst,
        RV is NewV,
        write(' V is imposed to a   '),nl,
        write(' higher optimum value.'),nl,
        write('   N is minimized.    '),nl,
        write('       [case 1]       '),pause.

diag_lle(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
        N>=Nopt,
    [!recorded(data,optimal_E(Eopt),_)!],
        NewV is Eopt*L/KConst,
        RV is V,
        RV*KConst/L < Eopt,
        write(' V is lower than the '),nl,
        write('  optimum value N is '),nl,
        write('   higher than the   '),nl,
        write('        optimum      '),nl,
        write('        [case 2]     '),pause.

diag_lle(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
    N>=Nopt,
        NewV is V,
    [!recorded(data,limit_E(Esl),_)!],
        RV < Esl*L/KConst,
        write(' V is higher than the'),nl,
        write(' optimum value.      '),nl,nl,
        write('    The minimal      '),nl,
        write(' solution for N is   '),nl,
        write(' greater than    the '),nl,
        write(' optimum of '),write(Nopt),write('.'),nl,nl,
        write('         [case 3]'),pause.
```

```
diag_lle(N,V,NewV,RV,KConst,L):-
        cls,
        [!recorded(data,optimal_N(Nopt),_)!],
        N<Nopt,
        write(' V is much higher than '),nl,
        write(' the optimum value.    '),nl,
        write(' N is lower than the '),nl,
        write(' optimum (Nopt='),write(Nopt),write('). '),nl,
        write(' The separation is '),nl,
        write(' unpractical because '),nl,
        write(' N is too low, and not'),nl,
        write(' economical because V '),nl,
        write(' is too high: E>Est.'),
        write('          [case 4]'),pause.

/*****************************************************
* Calculates the average deviation in the doverb 's*
* to be used in the CEEE.                           *
*****************************************************/

calc_dev(DBact,DBspec,Dev):-
        sum_dev(DBact,DBspec,Sum),
        length(DBspec,L),
        Dev is Sum/L.

sum_dev([],[],0).
sum_dev([HA|TA],[HB|TB],Sum):-
        sum_dev(TA,TB,SumTail),
        Sum is (abs(HA-HB)/HA+SumTail).

/*****************************************************
* Intermediate storage of the separations obtained *
* before final choice and end of the module        *
*****************************************************/

store_sep_lle(Dev,N,E,List,CList,TopList,BotList,DBact):-
        recorded(data,solvent(S),_),
        append([solvent],List,NList),
        append([S],CList,NCList),
        component_flow_set(TopList,NCList,TopFList),
        component_flow_set(BotList,NCList,BotFList),
        component_flow_set(List,CList,FList),
        actual_flows(FList,DBact,Top_no_s_AFList,Bot_no_s_AFList),
        recorded(data,flow(solvent,S,V),_),
        append([V],Top_no_s_AFList,TopAFList),
        append([0],Bot_no_s_AFList,BotAFList),
        calc_CE(Dev,N,E,CE),
        recordb(lle,CE,dseparator_lle(N,E,NCList,TopList,TopAFList,BotList,BotAFList)).


/*****************************************************
* To decide which separation is the best we calcu- *
* late the Coefficient for LLE.                     *
*       It is a function of the average deviation*
* ,the number of stages and the lle factor        *
*****************************************************/

calc_CE(Dev,N,E,CE):-
        recorded(data,optimal_N(Nopt),_),
        recorded(data,optimal_E(Eopt),_),
        CED is -ln(Dev),
```

```
        CEN is -ln( N*N/2-(Nopt-2)*N+(Nopt-2)*(Nopt-2)/2+1 )+2*sqrt(2)*atan( (N-(Nopt-2))/sqrt(2) ),
        CEE is -ln( E*E/2-(Eopt+2)*E+(Eopt+2)*(Eopt+2)/2+1 )-2*sqrt(2)*atan( (E-(Eopt+2))/sqrt(2) ),
        CE is round(exp(CED+CEN+CEE),0).


/*****************************************************
* We need to choose the best separation among      *
* those generated previously and stored in         *
* dseparator_lle.                                  *
*****************************************************/

choose_sep_lle:-
        [!recorded(data,minimum_E(Emin),_),
        recorded(data,stage_E(Est),_),
        recorded(data,limit_E(Esl),_),
        recorded(data,optimal_E(Eopt),_),
        write(' The range of LLE Factor is:'),nl,
        write(Emin),write(' - '),write(Esl),nl,
        current_window(CW,seven),cls,
        diag_lle_range(Emin,Eopt,Est,Esl),
        current_window(_,CW) !],
        findall(X,retrieveb(lle,X,_),ListX),
        quicksort(ListX,[HX|TX]),
        cls,nl,
        write('      The Best Separation According to the Analysis is:'),
        write_sep([HX|TX]).
choose_sep_lle:-fail.

write_sep([]).
write_sep([HX|TX]):-
        retrieveb(lle,HX,dseparator_lle(N,E,NCList,TopList,TopAFList,BotList,BotAFList)),
        nl,write(' The lle column '),
    write(' is specified the results are: '),nl,nl,
    write('   Column        Separation'),nl,
    write(' ———————      ———————'),nl,nl,
    write('  LLE      '),write(TopList),write('/'),write(BotList),
        nl,nl,
        write('Components:'),write(NCList),nl,
        write('Top       :'),
        write_round(TopAFList,1),nl,
        write('Bot       :'),write_round(BotAFList,1),nl,nl,nl,
        write(' The design parameters are:'),nl,
        NR is round(N,2),
        write(' Number of theoretical stages:'),write(NR),nl,nl,
        write(' LLE factor:'),write(E),nl,nl,
        pause,
        write('Do you Accept this one, if not the other solutions will be displayed'),
    send_menu_msg(activate(yes_no2,(10,60)),Decision),!,
        solve_choice(Decision,[HX|TX]).

solve_choice(Decision,[HX|TX]):-
        Decision=yes,
        recordz(data,sep_CE(HX),_).

solve_choice(Decision,[HX|TX]):-
        Decision=no,cls,write_sep(TX).

/*****************************************************
* Here, the program diagnoses if the range of lle  *
* factors entered by the user is likely to contain *
* all the possible solutions.                      *
```

```
**************************************************/

                    /*case1*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Esl<Eopt,
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        write(' If you want to use an '),nl,
        write(' optimal value for E, '),nl,
        write(' you must not have the '),nl,
        write(' limit Elg smaller '),nl,
        write(' than the optimal value'),nl,
        write(' Eopt:'),write(Eslt),write(' > '),write(Eoptt),write('.'),
        pause.

                    /*case2*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Emin>Esl,
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        write(' The upper limit Esl is'),nl,
        write(' smaller than the lower'),nl,
        write(' limit Emin:'),write(Emint),write(' < '),write(Eslt),write('.'),nl,
        write(' You must have Emin<Esl'),nl,
        write(' for any calculation to'),nl,
        write(' occur.'),pause.

                    /*case3*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Emin<Eopt,
        Eopt<Est,
        Est<Esl,
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        [!recorded(data,optimal_N(Nopt),_)!],
        Noptt is round(Nopt,2),
        write(' No more solutions are '),nl,
        write('     expected in    '),nl,
        write('   the studied range. '),
        pause.
                    /*case4*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Emin<Eopt,
        Eopt<Esl,
        [!recorded(data,optimal_N(Nopt),_)!],
        Esl<Est,  %Est actually does not exist because Esl is to small to reach it, Est is initialized to Esl+1
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        Noptt is round(Nopt,2),
        write(' Solutions may exist '),nl,
        write(' for E>'),write(Eslt),write('. With a'),nl,
        write(' number of stages still'),nl,
        write(' higher than '),write(Noptt),write('. You'),nl,
        write(' should increase Esl.'),
        pause.

                    /*case5*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Emin<Est,
        Est<Eopt,
        Eopt<Esl,
        [!recorded(data,optimal_N(Nopt),_)!],
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        Noptt is round(Nopt,2),
        write(' The  optimal  E ,  '),nl,
```

```
        write(' Eoptt = '),write(Eoptt),write(', should'),nl,
        write(' be  smaller  than '),nl,
        write(' Est='),write(Estt),write(' because'),nl,
        write(' Eopt ought to be where'),nl,
        write(' N is greater than '),write(Noptt),write('.'),
        pause.

                /*case6*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Emin>=Eopt,
        Emin=:=Est, %the first value of E will instantiate Est
        recorded(data,optimal_N(Nopt),_),
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        Noptt is round(Nopt,2),
        write(' The  initial  value,'),nl,
        write(' Emin='),write(Emint),write(', is too high.'),
        write(' Only solutions with N '),nl,
        write(' smaller  than  '),write(Noptt),nl,
        write(' will be generated.'),nl,
        write(' Worse, the optimal E  '),nl,
        write(' Eopt='),write(Eoptt),write(', is too'),nl,
        write(' small to be studied. '),nl,
        write(' It should be greater '),nl,
        write(' than '),write(Emint),write('.'),
        pause.

                /*case7*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Emin>=Eopt,
        Emin<Est,
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        write(' The  minimum  value  '),nl,
        write(' Emin='),write(Emint),write(' should be '),nl,
        write(' smaller if you want  '),nl,
        write(' to use the optimal   '),nl,
        write(' value Eopt='),write(Eoptt),write('.'),nl,
        pause.

                /*case8*/
diag_lle_range(Emin,Eopt,Est,Esl):-
        Emin<Eopt,
        Emin =:= Est, %the first value of E will instantiate Est
        recorded(data,optimal_N(Nopt),_),
        Emint is round(Emin,2),Eoptt is round(Eopt,2),Estt is round(Est,2),Eslt is round(Esl,2),
        Noptt is round(Nopt,2),
        write(' The initial value of E'),nl,
        write(' Emin='),write(Emint),write(' is too high.'),nl,
        write(' Only solutions with N'),nl,
        write(' smaller than '),write(Noptt),write(' will '),nl,
        write(' be generated.'),
        pause.

diag_lle_range(_,_,_,_):-
        write('No diagnosis. Error'),pause.

/*****************************************************
* Recovery clause to separate the solvent from    *
* the top. Call of the washing module to be imple- *
* -mented                                          *
*****************************************************/
recovery_lle:-
```

```
    write('The recovery of the solvent is supposed to be feasible and sharp'),nl,
recorded(sep,dseparator(Rum,NCList,TopAFList,NCList,BotAFList),_),
recorded(data,solvent(S),_),
    RRum is Rum + 1,
    recorda(bypass,dstreambypass(RRum,'none',[],[]),_),
    recorda(sep,dstreamin(RRum,NCList,TopAFList),_),
[V|TF] = TopAFList,
recorda(data,flow(solvent,S,V),_),
    delete_element(S,NCList,CList),
    record_comp_flow_in_solvent_lle([solvent],CList,V),
append([0],TF,BotRFList), %bot flows of recovery column
component_flow_set([solvent],NCList,TopRFList),
    recorda(sep,dseparator(RRum,NCList,TopRFList,NCList,BotRFList),_),
    write(' The results for the RECOVERY  COLUMN #'),
    write(RRum),write(' are now available').
```

UTILITY MODULE (UTILITY.ARI)

```
:-segment(utiseg).
:-public
        append/3:far,
        append_v/3:far,
        component_flow/3:far,
        component_flow_list/4:far,
        component_flow_set/3:far,
        decide_difficult_or_easy/2:far,
        delete_element/3:far,
        delete_elements/3:far,
        equal/2:far,
        erase_term/2:far,
        ffactor/3:far,
        first_prod/2:far,
        flow_set/3:far,
        flow_set_c/3:far,
        get_split_flow/4:far,
        largest_flow/3:far,
        last_prod/2:far,
        list_member/2:far,
        max/3:far,
        max_flist/2:far,
        max_flow/3:far,
        member/2:far,
        pause/0:far,
        pause_escape/0:far,
        positive_component_flow_list/2:far,
        print_cam_driver/3:far,
        product_flow/3:far,
        product_flow_match/4:far,
        product_flow_set/3:far,
        quicksort/2:far,
        reverse/2:far,
        selected_positive_component_flow_list/3:far,
        set_above/3:far,
        set_below/3:far,
        split_above/4:far,
        split_below/4:far,
        sub_list/2:far,
        sum_component_flow/3:far,
        sum_flist/2:far,
        two_highest_flows/3:far.

:-visible member/2.


retractall(X):-
        call(X),
        retract(X),
        fail.
retractall(_).

quicksort([],[]).
quicksort([X|Tail],Sorted):-
    split_sort(X,Tail,Small,Big),
```

```
    quicksort(Small,SortedSmall),
    quicksort(Big,SortedBig),
    append(SortedSmall,[X|SortedBig],Sorted).

split_sort(_,[],[],[]).
split_sort(X,[Y|Tail],[Y|Small],Big):-
    X < Y,!,
    split_sort(X,Tail,Small,Big).
split_sort(X,[Y|Tail],Small,[Y|Big]):-
    split_sort(X,Tail,Small,Big).

quicksort_inc([],[]).
quicksort_inc([X|Tail],Sorted):-
    split_sort_inc(X,Tail,Small,Big),
    quicksort_inc(Small,SortedSmall),
    quicksort_inc(Big,SortedBig),
    append(SortedSmall,[X|SortedBig],Sorted).

split_sort_inc(_,[],[],[]).
split_sort_inc(X,[Y|Tail],[Y|Small],Big):-
    X > Y,!,
    split_sort_inc(X,Tail,Small,Big).
split_sort_inc(X,[Y|Tail],Small,[Y|Big]):-
    split_sort_inc(X,Tail,Small,Big).

max(A,B,A):- A > B,!.
max(A,B,B):- A =< B,!.

first_prod([],[]).
first_prod([X|_],[X]).

last_prod([],[]).
last_prod([X],[X]):-!.
        last_prod([_|Tail],X):-
    last_prod(Tail,X).

pause:-
        create_popup(label('Pause',center,14,0),(22,52),(24,79),(14,14)),
        write(' Press Enter to continue.'),
        get0(_),
        exit_popup.

pause_escape:-
    nl,
    write('Pause. Press enter to procees, or ESC to'),nl,
    write('consider other alternatives'),nl,
    read(_).

ffactor(Top,Bot,Ratio):-
    Top < Bot,!,
    Ratio is Top/Bot.
ffactor(Top,Bot,Ratio):-
    Top >= Bot,!,
    Ratio is Bot/Top.

split_above(List,Prod,SetAboveIncludingProd,SetBelowProd):-
    set_above(List,Prod,SetAboveProd),
    set_below(List,Prod,SetBelowProd),
    append(SetAboveProd,[Prod],SetAboveIncludingProd).

split_below(List,Prod,SetAboveProd,[Prod|SetBelowProd]):-
```

```
        set_above(List,Prod,SetAboveProd),
        set_below(List,Prod,SetBelowProd).

set_above(List,Prod,ListAbove):-
        append(ListAbove,[Prod|_],List),!.

set_below(List,Prod,ListBelow):-
        set_above(List,Prod,ListAbove),
        append(ListAbove,[Prod|ListBelow],List),!.

largest_flow(List,Lprod,CList):-
        product_flow_set(List,FlowList,CList),
        max_flist(FlowList,Lflow),
        product_flow_match(List,Lflow,Lprod,CList).

two_highest_flows(List.MFlow,SFlow):-
        max_flist(List,MFlow),
        delete_element(MFlow,List,NList),
        max_flist(NList,SFlow),!.

product_flow(Head,PFlow,CList):-
        flow_set(Head,CList,PFlowList),
        sum_flist(PFlowList,PFlow).

product_flow_help(_,PFlow,[],PFlow):-!.
product_flow_help(Prod,PFlow,[HeadC|TailC],RefFlow):-
        recorded(data,flow(Prod,HeadC,NFlow),_),!,
        NRefFlow is RefFlow + NFlow,
        product_flow_help(Prod,PFlow,TailC,NRefFlow).

product_flow_match([Prod|_],PFlow,Prod,CList):-
        product_flow(Prod,Flow,CList),
        Flow is PFlow,!.
product_flow_match([_|Tail],PFlow,Prod,CList):-
        product_flow_match(Tail,PFlow,Prod,CList).

product_flow_set([],[],_).
product_flow_set([Head|Tail],[HeadFlow|TailFlow],CList):-
        product_flow(Head,HeadFlow,CList),
        product_flow_set(Tail,TailFlow,CList).

flow_set(_,[],[]).
flow_set(Prod,[HeadProd|TailProd],[HeadFlow|TailFlow]):-
        recorded(data,flow(Prod,HeadProd,HeadFlow),_),!,
        flow_set(Prod,TailProd,TailFlow).

append_v([],List,List).
append_v([Head|Tail],List1,[Head|List]):-
        append_v(Tail,List1,List).

max_flist([X],X).
max_flist([Int1,Int2|Tail],Max):-
            max_flow(Int1,Int2,Mflow),
            max_flist([Mflow|Tail],Max).

max_flow(F1,F2,F1):-
        F1 >= F2,!.
max_flow(F1,F2,F2):-
        F1 < F2,!.

sum_flist([],0).
```

```
sum_flist([Head|Tail],Sum):-
        sum_flist(Tail,Sumtail),
        Sum is Head + Sumtail.

append([],List,List).
append([Head|Tail],List1,[Head|List]):-
        append(Tail,List1,List).

reverse([],[]).
reverse([Head|Tail],List):-
        reverse(Tail,List1),
        append(List1,[Head],List).

equal(X,X).

sub_list(Sublist,List):-
            append(_,List1,List),
        append(Sublist,_,List1),!.

positive_component_flow_list(List,CList):-
            recorded(data,initial_components(Components),_),!,
        component_flow_set(List,Components,FlowList),
        component_flow_list(Components,FlowList,[],CList).

selected_positive_component_flow_list(List,CList,CompList):-
            recorded(data,initial_components(Components),_),!,
        component_flow_set(List,Components,FlowList),
        component_flow_list(Components,FlowList,[],RefCompList),
        build_list(RefCompList,CList,[],CompList),!.

build_list([],_,C,C):- !.
build_list([Head|Tail],CList,RefBuildList,CompList):-
        member(Head,CList),
        append(RefBuildList,[Head],NewRefBuildList),
        build_list(Tail,CList,NewRefBuildList,CompList),!.
build_list([Head|Tail],CList,RefBuildList,CompList):-
        not(member(Head,CList)),
        build_list(Tail,CList,RefBuildList,CompList),!.

component_flow_list([],[],CList,CList).
component_flow_list([HR|TR],[HF|TF],TempCList,CList):-
        HF > 0,
        append(TempCList,[HR],NewCList),!,
        component_flow_list(TR,TF,NewCList,CList).
component_flow_list([_|TR],[HF|TF],TempCList,CList):-
        HF =< 0,!,
        component_flow_list(TR,TF,TempCList,CList).

component_flow_set(_,[],[]).
component_flow_set(List,[Head|Tail],[HeadFlow|TailFlow]):-
        component_flow(List,Head,HeadFlow),
        component_flow_set(List,Tail,TailFlow).

get_split_flow([],TFlow,_,TFlow):-!.
get_split_flow([Head|Tail],TFlow,CList,RefFlow):-
        !,product_flow_help(Head,PFlow,CList,0),
        NRefFlow is RefFlow + PFlow,
        get_split_flow(Tail,TFlow,CList,NRefFlow).

component_flow(List,Head,CFlow):-
        flow_set_c(Head,List,CFlowList),
```

```
        sum_flist(CFlowList,CFlow).

flow_set_c(_,[],[]).
flow_set_c(Comp,[HeadProd|TailProd],[HeadFlow|TailFlow]):-
        recorded(data,flow(HeadProd,Comp,HeadFlow),_),!,
        flow_set_c(Comp,TailProd,TailFlow).

delete_elements([],Y,Y).
delete_elements([Head|Tail],X,Y):-
            delete_element(Head,X,Z),
        delete_elements(Tail,Z,Y).

delete_element(X,[X|Tail],Tail).
delete_element(X,[Y|Tail],[Y|Tail1]):-
        delete_element(X,Tail,Tail1),!.

sum_component_flow([],SumCompFlow,_):-
            SumCompFlow is 0.
sum_component_flow([Head|Tail],SumCompFlow,List):-
        component_flow(List,Head,HeadFlow),
        sum_component_flow(Tail,RunningSum,List),
        SumCompFlow is RunningSum + HeadFlow.

list_member([],_):-!.
list_member([Head|Tail],List):-
        member(Head,List),
        list_member(Tail,List).

member(X,[X|_]):- !.
member(X,[_|Tail]):-
        member(X,Tail).

decide_difficult_or_easy(Delta,'easy'):-
        Delta > 10.0,!.
decide_difficult_or_easy(Delta,'difficult'):-
        Delta =< 10.0,!.

erase_term(Db,Term):-
            recorded(Db,Term,Ref),
            erase(Ref),
            fail.
erase_term(Db,_).

print_cam_driver(List,CList,'fresh'):-
                cls,nl,
          write(' Component Assignment Matrix is complete.'),nl,
          write(' Do you wish to print it on the screen?'),
                send_menu_msg(activate(yes_no,(17,32)),Decision),cls,
                Decision = yes,nl,nl,
          print_six_element_list(CList,CList,List),!.
print_cam_driver(_,_,'fresh').

print_cam_driver(List,CList,'bypass'):-
          cls,nl,
          write(' Component Assignment Matrix has been recalculated due to bypass.'),nl,
          write(' Do you wish to print it on the screen?'),nl,
                send_menu_msg(activate(yes_no,(17,32)),Decision),cls,
                Decision = yes,nl,nl,
          print_six_element_list(CList,CList,List),!.
print_cam_driver(_,_,'bypass').
```

```
print_cam_driver(List,CList,solvent):-
        cls,nl,
        write(' Component Assignment Matrix has been recalculated due to solvent.'),nl,
        write(' Do you wish to print it on the screen?'),nl,
            send_menu_msg(activate(yes_no,(17,32)),Decision),cls,
            Decision = yes.nl,nl,
        print_six_element_list(CList,CList,List), pause, !.
print_cam_driver(_,_,solvent).

print_cam_driver(List,CList,lean_gas):-
        cls,nl,
        write(' Component Assignment Matrix has been recalculated due to lean gas.'),nl,
        write(' Do you wish to print it on the screen?'),nl,
            send_menu_msg(activate(yes_no,(17,32)),Decision),cls,
            Decision = yes.nl,nl,
        print_six_element_list(CList,CList,List), pause, !.
print_cam_driver(_,_,lean_gas).


print_six_element_list(CList,[E1,E2,E3,E4,E5,E6|Tail],List):-
        print_cam(CList,List,[E1,E2,E3,E4,E5,E6]),
        print_six_element_list(CList,Tail,List).
print_six_element_list(CList,Remain,List):-
        print_cam(CList,List,Remain).

print_cam(_,_,[]).
print_cam(KList,List,CList):-
        write(' Products  Total Flow     Components'),nl,
        reverse(List,Rlist),
        write(' ------ -------  ----------'),nl,
            tab(19),
        print_component_line(CList,1),
        print_cam_line(KList,Rlist,CList),
        nl.

print_component_line([],_):- nl,nl.
print_component_line([Head|Tail],C):-
            NewC is (8*C) + 18,
            tget(R,_), tmove(R,NewC),
            write(Head),
            inc(C,IC),
        print_component_line(Tail,IC).

print_cam_line(_,_,[]).
print_cam_line(_,[],_).
print_cam_line(KList,[Head|Tail],CList):-
        product_flow(Head,Flow,KList),
        tab(3),write(Head).tab(8),write(Flow),
            print_row_flows(Head,CList,1),nl,
        print_cam_line(KList,Tail,CList).

print_row_flows(_,[],_).
print_row_flows(Prod,[Head|Tail],C):-
            NewC is (8*C) + 18,
            tget(R,_), tmove(R,NewC),
        recorded(data,flow(Prod,Head,Flow),_),
            WFlow is round(Flow,2),write(WFlow),
            inc(C,IC),
        print_row_flows(Prod,Tail,IC).
```

MENU DRIVER (MENU.ARI)

```
begin_menu(yes_no,13,colors((78,78),(14,78),(14,71),(12,78))).
     item($Decision $,
    [item($   ~Yes   $,yes),
     item($   ~No    $,no) ] ).
end_menu(yes_no).

begin_menu(yes_no2,13,colors((14,14),(14,78),(14,71),(12,14))).
     item($Decision $,
    [item($   ~Yes   $,yes),
     item($   ~No    $,no) ] ).
end_menu(yes_no2).

begin_menu(no_yes,13,colors((78,78),(14,78),(14,71),(12,78))).
     item($Decision $,
    [item($   ~No    $,no),
     item($   ~Yes   $,yes) ] ).
end_menu(no_yes).

begin_menu(separations,28,colors((78,78),(14,78),(14,71),(12,78))).
     item($   Separation Methods   $,
    [item($Ordinary ~Distillation$,od,unchecked),
     item($Dilute Gas ~Absorption$,dga,unchecked),
     item($~Stripping$,s,unchecked),
     item($~Liquid-Liquid Extraction$,lle,unchecked),
                  break,
     item($~Bulk Gas Absorption$,bga,greyed),
     item($~Extractive Distillation$,ed,greyed),
    item($A~zeotropic Distillation$,ad,greyed)
                        ]
       ).
end_menu(separations).

begin_menu(data_loading,23,colors((78,78),(14,78),(14,71),(12,78))).
     item($Data Loading Choice$,
    [item($   From ~File      $,1),
     item($   From ~Terminal  $,2) ] ).
end_menu(data_loading).

begin_menu(bypass,12,colors((78,78),(14,78),(14,71),(12,78))).
     item($ Answer $,
    [item($ ~100    $,'100'),
     item($  ~90    $,'90'),
     item($~Manual  $,'manual')] ).
end_menu(bypass).


begin_menu(terminate,29,colors((78,78),(14,78),(14,71),(12,78))).
     item($Further Program Execution$,
    [item($       ~Terminate          $,terminate),
     item($   ~Synthesize Another    $,synthesize_another)]).
end_menu(terminate).

begin_menu(split,19,colors((78,78),(14,78),(14,71),(12,78))).
     item($     Answer    $,
    [item($    ~Split      $,split),
     item($  ~Do Not Split  $,notsplit)]).
```

```
end_menu(split).

begin_menu(technique,20,colors((78,78),(14,78),(14,71),(12,78))).
     item($Search Technique$,
    [item($ ~Heuristically  $,heuristically),
     item($    ~Manually    $,manually)]).
end_menu(technique).

begin_menu(display,24,colors((78,78),(14,78),(14,71),(12,78))).
     item($     SST Display    $,
    [item($  ~Do Not Display    $,donotdisplay),
     item($  ~Display To Screen $,displaytoscreen),
     item($        ~Print       $,print)]).
end_menu(display).

begin_menu(remove,16,colors((78,78),(14,78),(14,71),(12,78))).
     item($   Answer   $,
    [item($   ~Remove   $,remove),
     item($  ~Maintain  $,maintain)]).
end_menu(remove).
```

VITA

Jean-Christophe Brunet was born on March 26, 1968, in Fontainebleau, France. He received his "Baccalaureat Serie C" (Math and Physics) in 1985, he prepared for the entrance examination for the "Grandes Ecoles" during 1985 to 1988. In Fall 1988, he joined the Chemical Engineering Department at the "Université de Technologie de Compiègne" (U.T.C.), France. While studying at U.T.C., he participated in the U.T.C. - Virginia Tech exchange program to enroll in the Master Program in Chemical Engineering at VPI & SU in August 1990. He received his french "Diplôme D'Ingénieur" (5-year program) in March 1992. After completing the requirements for an Master's Degree at Virginia Tech, he will seek employment in France after getting married to Miss Sylvaine Chardon on September, 5, 1992.

Jean-Christophe Brunet