REFERENCE ONLY

UNIVERSITY OF LONDON THESIS

e PhD Year 2005 Name of Author O'GNADY, A.N.F

COPYRIGHT

This is a thesis accepted for a Higher Degree of the University of London. It is an unpublished typescript and the copyright is held by the author. All persons consulting the thesis must read and abide by the Copyright Declaration below.

COPYRIGHT DECLARATION

I recognise that the copyright of the above-described thesis rests with the author and that no quotation from it or information derived from it may be published without the prior written consent of the author.

LOAN

Theses may not be lent to individuals, but the University Library may lend a copy to approved libraries within the United Kingdom, for consultation solely on the premises of those libraries. Application should be made to: The Theses Section, University of London Library, Senate House, Malet Street, London WC1E 7HU.

REPRODUCTION

University of London theses may not be reproduced without explicit written permission from the University of London Library. Enquiries should be addressed to the Theses Section of the Library. Regulations concerning reproduction vary according to the date of acceptance of the thesis and are listed below as guidelines.

A. Before 1962. Permission granted only upon the prior written consent of the author. (The University Library will provide addresses where possible).

B. 1962 - 1974. In many cases the author has agreed to permit copying upon completion of a Copyright Declaration.

C. 1975 - 1988. Most theses may be copied upon completion of a Copyright Declaration.

D. 1989 onwards. Most theses may be copied.

This thesis comes within category D.

☑ This copy has been deposited in the Library of UCL

☐ This copy has been deposited in the University of London Library, Senate House, Malet Street, London WC1E 7HU.

C:\Documents and Settings\lproctor.ULL\Local Settings\Temporary Internet Files\OLK36\Copyright - thesis.doc

Automated design of separation processes using implicit enumeration and interval analysis

Andrew Robert Francis O'Grady

A thesis submitted for degree of
Doctor of Philosophy
in Chemical Engineering

June 2004

Department of Chemical Engineering
University College London

UMI Number: U593076

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U593076
Published by ProQuest LLC 2013. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Acknowledgements

I would like to thank Dr. Eric Fraga for pointing me in the right direction when lost and encouraging me when everything seemed hopeless. Thanks also go to Professor David Bogle for his useful comments and suggestions.

I am grateful to all the staff and students of the Computer Aided Process Engineering group for their great company and sound advice. The financial support provided by the U.K. Engineering and Physical Sciences Research Council and the Department of Chemical Engineering, UCL is also gratefully acknowledged.

Above all, thanks to Claire for staying awake while proof reading this and being so understanding over the years.

Abstract

This thesis concerns the automated synthesis of separation processes. A single multi-component stream is to processed to give one or more pure component product streams. A list of units are available for the task and the aim is to find the optimal flowsheet structure in terms of cost. Implicit enumeration (IE) has been used to tackle the synthesis problem. The main advantage of this approach is that IE does not require the development of a superstructure.

A disadvantage of using IE is that it is necessary to discretise the values of unit operating conditions in order for there to be a finite search space (Fraga et al., 2000). The user may not have any idea of the effect of the discretisations on the *quality* of the solution. In addition, the optimal solution may be missed between the discrete values chosen. The purpose of this work is to address these issues.

Interval analysis is used to bound the effects of this discretisation. This allows the cost of each particular flowsheet to be bounded based on the level of discretisation used. The technique is demonstrated by bounding the effect of discretisation on the synthesis of distillation flowsheets. The use of runs with progressively finer uniform discretisation lead to the isolation of the optimal structure.

This result leads to the development of an adaptive algorithm that changes the discretisation profile in response to bounding information from downstream in the search. The algorithm operates recursively and isolates the optimal process structure for each stream encountered. This builds up to the isolation of the overall optimal process structure for the

feed process stream. The effectiveness and performance of the new algorithm are evaluated using two very different separation problems. The first is a distillation sequence and the second a separation of a protein from a biological stream.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Process synthesis can be defined as the systematic generation of flowsheets for a chemical process. The aim is to optimise the logical structure of processing units. The objective function is usually financial but may include a measure of the environmental impact of the process. The synthesis procedure takes place at the very early stages of plant design but can have large cost implications. If poor decisions are made at this stage, then later cost analysis of the resulting detailed flowsheet may mean that the process is deemed economically infeasible.

The synthesis problem can take one of several forms which include:

- Determination of the optimal heat exchanger network configuration for a given flowsheet structure. This may be based on pinch technology (Linnhoff and Hindmarsh, 1983), mathematical programming or a combination of the two.
- Optimisation of the structure of a reactor network including identification of suitable recycle streams.

- Separation network synthesis (SNS)
- Mass exchange network optimisation.
- Utility system optimisation.

These issues may be addressed together where whole plant-wide optimisation is attempted. It is possible to tackle the synthesis problem in several different ways. Traditionally synthesis has been carried out by the use of heuristics (Douglas, 1988). Another approach is the evolution of an existing flowsheet via small modifications (Stephanopoulos and Westerberg, 1976). Increasingly mathematical algorithms have been developed (Grossmann, 1996). Some methods combine both heuristic and mathematical approaches (Daichendt and Grossmann, 1998).

In terms of mathematical programming, the detail of the mathematical models used can be increased as the subsystem being synthesised becomes more specific. This is due to the decrease in problem size if subsystems are considered individually. For example, it is generally possible to use much more detailed models in the optimisation of heat integration on an existing flowsheet than if the flowsheet were to be synthesised and heat integration considered simultaneously. Therefore, there is often a trade off between the need to consider as wide a level of plant synthesis as possible and the detail of the models used to simulate this.

1.1 Motivation

This thesis is concerned with the development of methods that can be used to find the optimal flowsheet structure for a process given a feedstock, a set of units and product

specifications. The techniques used are based upon an implicit enumeration (IE) procedure where a search graph is simultaneously created and traversed. In the graph, units are represented by nodes and streams by edges. An existing method is embodied by the Jacaranda automated process synthesis package (Fraga et al., 2000). In this package streams, units and the search algorithm itself, are implemented in an object oriented framework. The merits of this and various other approaches to the synthesis problem are discussed in detail in Chapter 2.

An advantage of the IE approach is that the search graph is generated automatically. This prevents an engineer from imparting any preconceptions to the design potentially yielding a novel process structure. As a result, the possibility of yielding a novel process structure is increased. A drawback of this approach is that in order for enumeration to take place, continuous unit variables are discretised. The size of the search space is further reduced by the discretisation of continuous stream variables such as pressure and flowrate. This is combined with dynamic programming to allow the re-use of solutions. The user has no idea of the effect of these discretisations on the quality of the solutions obtained. In addition, the ranked list of solutions returned is not guaranteed to contain the optimal process structure.

The object of the work described in this thesis is to develop techniques that retain the advantages of the implicit enumeration search, while allowing the optimal process structure to be isolated automatically. The different stages of development of such an algorithm are described.

1.2 Interval analysis

Since the discretisation of unit variables means that the structures returned by the existing method may not be optimal, it is necessary to quantify the effects of discretisation. Interval analysis can be used for this purpose. Carrying out unit designs using interval arithmetic produces bounds on the cost of each design. The bounds are based on the interval values of the unit variables and feed stream properties. Chapter 3 explains the concept of interval analysis and how it can be applied to the design of processing units.

Chapter 4 describes how intervals may be used to represent the unit variable pressure in a distillation column. This results in distillation column designs with bounds on capital and operating costs. This is applied to a hydrocarbon separation problem and results in bounds on the cost of the structures returned. Runs are carried out at various levels of discretisation. At a sufficiently fine level of discretisation, it is demonstrated that the *best* structure can be isolated on the basis of cost bounds.

Chapter 5 develops the application of interval analysis further by describing component flow rates using intervals. This is an important step, as whether or not a component is present within a stream determines how the stream is processed and affects the overall process structure.

1.3 An adaptive algorithm

The results of implementation of the ideas in chapters 4 and 5 give assurances on the quality of the solutions obtained. By carrying out successive runs it is also possible to isolate a structure based on cost bounds. Such a structure must be optimal as long as

interval parameters do not cross feasibility boundaries during the search. However, during a particular run, there is no guarantee that this is true. In addition, successive runs using uniform discretisation is a time consuming and inefficient process

Chapter 6 describes an algorithm that combines an interval box splitting algorithm with the implicit enumeration search. The discretisation profiles of unit variables are changed for each stream encountered, by successive splitting of the unit variable intervals. This is carried out until the optimal structure for processing the stream has been found. If an interval parameter, resulting from a unit design, crosses a feasibility boundary then a unit variable interval is split until crossing does not occur. The search space is thoroughly investigated and nothing is discounted until it is shown to be either sub-optimal or infeasible.

Chapter 7 applies the new algorithm to the benzene separation synthesis problem attempted in chapter 5. The adaptive algorithm is able to isolate the optimal structure using around 40 times less processor time than if uniform discretisation that reached the same resolution were used. The algorithm is also applied to the synthesis of a biological separation process in order to demonstrate its applicability to various types of problems. The algorithm yields a different structure from the previous non-interval discrete approach. This shows that the optimal solution may be missed due to discretisation and supports the application of the new algorithm to such problems.

Chapter 2

Separation synthesis methods

2.1 Introduction

This chapter gives an overview of work in the field of process synthesis with the specific discussion of previous applications to separation network synthesis problems. Process synthesis techniques can be broadly classified as one of two approaches: (1) Heuristic methods and (2) Algorithmic methods. The former rely on previous experience of similar problems. The latter employ some kind of logical search procedure in order to find the optimal structure. The two approaches can be combined in order to discount some structures from the search. The merits of the different approaches are discussed and this discussion gives rise to the motivation for the work described by this thesis.

2.2 Terminology

Distillation is the one of the most studied technologies in terms of separation synthesis and part of this thesis is concerned with the synthesis of distillation sequences. The following is a list of terminology that can be used to describe distillation synthesis.

Light key: In a distillation unit this is defined as the lightest component which may be present in the bottom product in significant amounts.

Heavy key: This is the heaviest component that may be present in the top product in significant amounts. *Light* and *heavy* refer to the relative volatilities of the feed components.

Semi-sharp separation: This gives 100% recovery of components lighter than the *light key* to the top product and 100% recovery of components heavier than the *heavy key* to the bottom product. The key components are distributed between the two product streams depending on the percentage key recovery.

Sharp separation: Sharp separation is an idealised situation where 100% key separation is assumed. This leads to negligible amounts of the *light key* in the bottom product and neglible amounts of the *heavy key* in the top product.

Non-sharp separation: All components may distribute between the two product streams.

Divider: This splits a process stream into two or more streams each with the same fractional composition and is analogous to a fork in the pipework.

Blender: This mixes two streams to yield one product stream.

2.3 Heuristic and artificial intelligence (AI) approaches

An example of a heuristic approach is hierarchical decomposition (Douglas, 1988). It breaks the problem down into five basic levels:

1. Batch versus continuous operation
2. Input-Output structure of the flowsheet
3. Recycle structure and reactor design
4. Separation systems
5. Heat exchanger networks

From the second decision on-wards, the economic potential is examined. It may be decided that further work is not justified on this basis. This screening hierarchy is based upon heuristics and engineering insight to converge on a design. This approach cannot rigorously ensure an optimal design and the reliance on heuristics may mean that novel structures are not considered at all. Interactions of variables at different decision levels are not taken into account which may cause optimal designs to be missed. However, it is the most widely used design methodology. This may be due to the intuitive nature of the procedure and the lack of a general purpose process synthesis package. PIP (Process Invention Procedure), a computer implementation of hierarchical decomposition is described by Kirkwood et al. (1988). Another attempt at implementing artificial intelligence is as part of the PROSYN package as described by Schembecker et al. (1994).

Evolutionary techniques use the previous experience of the designer to make small changes to an existing flowsheet. Stephanopoulos and Westerberg (1976) propose a set of rules,

by which to make modifications to create a *neighbouring* flowsheet. In addition, means by which to compare the flowsheets are suggested. These ideas are applied to multicomponent separation problems. This strategy has been combined with distillation synthesis heuristics (Seader and Westerberg, 1977). The heuristics help with the determination of the starting flowsheet and the strategy of applying the evolutionary rules.

A problem with using heuristics is that the rules sometimes conflict with each other. In addition, terms such as *large* or *high* are often used in the heuristics that are ambiguous. Djouad et al. (1997) use fuzzy set theory to aid the decision procedure. Four heuristic rules for separation by distillation are made quantifiable and each is given a weighting. For each possible split, the values for each rule are calculated. The degree of compatibility between each rule is considered before a decision is made.

An investigation compared AI to mathematical optimization (Best et al., 1987) in the solution of multicomponent separation by distillation. Distillation synthesis has well established heuristics, so it is well suited to the application of AI. The study showed that the AI generated flowsheets were significantly more expensive than those obtained by mathematical optimization when applied to a range of test problems. Other processes do not have such established heuristics so may perform worse in a similar comparison. The lack of optimality in the use of heuristics motivates the application of mathematical optimization to process synthesis problems.

2.4 Algorithmic methods

In contrast to the use of heuristics, algorithmic approaches are designed to search the possible structures in order to obtain the optimal solution. Consequently, these approaches tend to be much more time consuming and are only practical by the use of computers.

2.4.1 Optimisation by mathematical programming

One method of mathematical programming uses optimisation techniques to select the configuration and operating conditions of processing units based on what is called a superstructure. The optimisation is formulated as a mixed-integer problem (Grossmann, 1985). The superstructure is intended to represent all combinations of available unit operations and possible interconnections. In the problem formulation, the existence or absence of a particular unit is represented by 1 or 0 respectively. This approach results in a mixed integer nonlinear programming (MINLP) problem. Given a superstructure, there are a number of algorithms available to solve the MINLP. These include

- Branch and bound (Gupta and Ravindran, 1985). In this method the continuous nonlinear program (NLP) relaxation is solved. If the relaxed discrete variables happen to take integer values then the search is stopped. Otherwise, a tree search of the integer variables is carried out. Lower bounds produced from relaxed NLP problems are compared with the current upper bound. A particular path is discounted if the lower bound at any point is greater than the upper bound. A new upper bound will result if all discrete variables take integer values.

- Outer Approximation (Duran and Grossmann, 1986) where mixed integer linear programs(MILP) and NLP subproblems are solved successively. This type of NLP problem corresponds to a particular discrete combination of the integer variables, that arises from the solution of the MILP master problem. The NLPs yield upper bounds and the MILPs lower bounds on the solution.

- Generalised Benders decomposition (Geoffrion, 1972) uses a similar strategy to the outer approximation method. The methods differ in the way that the MILP problem is constructed.

- Extended cutting plane method (Westerlund and Pettersson, 1995). This is a geometrical method which can guarantee optimality when applied to pseudo-convex functions.

The superstructure can be represented by the *state task network* (STN) (Kondili et al., 1993). It is an example of a finite automaton (Kohavi, 1978). It recognises that feed streams undergo a set of transformations within a process. These transformations yield various intermediate states. A unit operation then carries out the *task* of converting a material from one set of states to another. One or more pieces of equipment may be assigned to each task or one piece of equipment may be used to carry out multiple tasks. A variation on this system is the *state equipment network representation* (SEN) (Smith, 1996) where the superstructure is represented by the possible states of the process and the equipment that can be used to convert between the states. In this representation, the number and type of pieces of equipment may be specified but it is necessary to state all the possible states that may result from a piece of equipment. An example is the synthesis of a sharp distillation sequence in order to separate a four component mixture. In the STN representation, each possible split (task) for every possible stream (state) is represented in the superstructure. This is accomplished by the use of mixers and dividers. State ABCD is first divided into three intermediate states. The first is processed by the task A/BCD, the second by AB/CD and the third by ABC/D. Hence the first produces the states A and BCD. A is a pure product state and BCD is split into two for further processing. In the SEN representation, it is identified that three columns are required to sharply split a four component mixture into pure components. The superstructure consists of these three columns with each having options of the tasks that it may perform. The output streams are represented by the states that correspond to the possible separation tasks. The column accepting ABCD may perform the tasks A/BCD, AB/CD or ABC/D. Consequently the state of the top stream is A,AB,ABC and the state of the bottom stream is BCD,CD,D.

These multiple states are then split into single states for further processing.

These two approaches were applied to the formation of heat-integrated shortcut distillation sequence superstructures (Yeomans and Grossmann, 1999). These were then formulated and solved as MILPs. It was shown that these two approaches are complementary to one another. SEN generated superstructures were later solved using rigorous tray-by-tray calculations (Yeomans and Grossmann, 2000).

It was proposed by Sargent (1998) that the STN representation can be combined with hierarchical decomposition for the synthesis of distillation systems. It is suggested that resulting design possibilities are examined for feasibility before moving to improved models. This approach is applicable to both ideal and azeotropic systems.

Linke and Kokossis (2003a) present a framework for generating superstructures for the combination of reaction and separation processes. The superstructure is generated from the combination of generic synthesis units. The reactor/mass exchange unit is compartmentalised into each phase present in the system. Each compartment can then exchange mass across a phase boundary or diffusion barrier. Recycle can occur between compartments if technically possible. The separation task unit performs a set of feasible separation tasks according to an order of separation based on a physical property. All combinations of separators that correspond to this operation are incorporated in the superstructure. This generic approach allows a wide range of processing technologies to be investigated including reactive distillation.

An advantage of the superstructure approach is that it is able to tackle many different types of synthesis problem. However, the optimization of a superstructure by the solution of a MINLP does not generally ensure a globally optimal solution for process synthesis problems. This is because most methods assume convexity to ensure global optimality. The functions involved are often non-convex leading to the presence of multiple local optima.

Zhu and Kuno (2003) recently presented a method to deal with these non-convexities in a MINLP and ensure global optimality. They propose a combination of generalised Benders decomposition and branch and bound, using convex quadratic under-estimators.

If a given superstructure is solved using a global optimization algorithm,this does not ensure that the global optimum for the synthesis problem has been found. In order for the global optimum to be ensured, it must be guaranteed that the superstructure contains all possible unit configurations and connections. Without a systematic approach to superstructure formation, the design is constrained by the imagination and insight of the engineer formulating the problem.

2.4.2 Automatic generation of the superstructure

A method of systematically generating the superstructure is presented by Friedler et al. (1993). The bi-partite graph (P-graph) is introduced because a conventional graph representation of a process structure is shown not to uniquely describe one particular alternative. Bi-partite means that the vertices of the graph are partitioned into two sets with no two vertices of the same set being adjacent (Friedler et al., 1993). The two sets, in the case of process synthesis, are unit operations and materials.

The synthesis problem is posed mathematically using set theory and an algorithm is described that rigorously forms what is termed the maximal structure. This is defined as the union of all possible solution structures.

It is necessary to define the raw materials available, the required products and the set of operations that can be used. An operation is defined by a set containing two subsets. One subset contains the set of inputs and the other the set of outputs. Therefore, it is necessary to list all the possible intermediate materials in order to define the unit operations. For

example, to define a reactor converting the materials A and B to C, the notation (A, B, C) would be used to define the unit operation. The product stream could be of variable composition depending upon the amounts of unreacted A and B. If this difference were to be represented, another unit would have to be defined (A, B, D). Where material D would be used to represent a material containing C with significant amounts of A and B. Similarly, only sharp separators can be used, as in order to define the outputs, it is necessary to assume that there are no residual amounts of bottom product in the top stream or top products in the bottom stream. Otherwise, it would be necessary to define a different unit operation for each possible top and bottom product composition.

The approach is rigorous in the sense that it yields a maximal structure that accounts for all feasible connections between units and materials. However, it is still limited by the necessity to define all possible outputs from the operating units that may be used. This task remains the responsibility of the person formulating the problem. Recently, the P-graph approach has been applied to the synthesis of azeotropic distillation systems (Feng et al., 2003).

2.4.3 Multi-component, multi product problems (MCP problems)

The P-graph technique has been applied to a class of separation network synthesis (SNS) problems, for which there was previously no method to create a rigorous superstructure. The goal was to synthesise a process where multiple multi-component feed streams yield multiple multi-component product streams (MCP problem) (Kovacs et al., 1999).

The P-graph approach was applied to the global optimization of some SNS problems that had been attempted previously (Kovacs et al., 2000). The objective was to minimise the sum of mass load multiplied by degree of difficulty of each separation. This allowed the

problem to be formulated as a linear program assuring the global optimum. In many cases a better solution was found than had previously been published. For the multi-component product problem, it is necessary to include dividers, blenders and recycle loops in the superstructure in order to prevent excluding some of the solution space (Kovacs et al., 1993). The use of mixers and dividers is often essential for this type of problem as the product specifications are not attainable without dividing and blending.

An algorithm that ensures the globally optimal separation sequence for the MCP problem, also assuming sharp splits, is presented by Wehe and Westerberg (1987). This approach is based on a superstructure and linear programming and gives the global solution for a given superstructure. The non-linearities introduced by splitters in a three component separation are reduced, by analysis, to two linear programs. For more components, the resulting non-linear programs are relaxed providing a linear lower bound.

The multi-component product problem has been considered, allowing non-sharp separation by Aggarwal and Floudas (1990). A superstructure is devised allowing for distribution of components between the top and bottom streams. There is one column in the superstructure for each of the adjacent separation key combinations. Initial shortcut simulations are used to determine the lower bound on key recoveries. The purpose of this is to ensure that there is no significant distribution of non-key components between product streams. Further simulations are used to develop a cost model for each of the columns. These span the range of feed flowrates, compositions and key recoveries. The problem is formulated as an MINLP and solved using a procedure that can search for and identify the global optimum, but does not assure that it will be found (Floudas et al., 1989). In the test problems, four out of the five solutions were assured to be global.

A solution method for the multicomponent product problem that does not require the generation of a superstructure is presented by McCarthy et al. (1998) and McCarthy (2000).

This method searches an implicitly created solution graph. It allows non-sharp separations, splitting and blending of process streams. Discretisation of stream and unit properties are necessary in order to keep the search space finite.

2.4.4 Hybrid methods

Hybrid methods combine two or more synthesis techniques in order to search for solutions.

Wahnschafft et al. (1991) describe a software system called SPLIT that aids the design of processes for the separation of non-ideal mixtures. In this case the major hurdle is the generation of feasible solutions rather than the pruning of weak alternatives. It uses a *blackboard* system to access various knowledge sources. Potential separation strategies are tested across the operating range by simulation. Alternative flowsheets are compared by combining inputs from the available knowledge sources. The user is able to influence the direction of the search by discounting some alternatives. Potential alternatives can then be incorporated into a superstructure for MINLP optimization.

Another issue in separation synthesis is technology that should be considered when a flowsheet is designed. A strategy to address this problem is presented by Jaksland et al. (1995). Physico-chemical properties of components in the feed stream are compared in order to identify the most appropriate separation technologies. A set of separation *tasks* are then identified. Finally, estimates of appropriate operating conditions are generated.

Bek-Pedersen and Gani (2004) present a set of algorithms for distillation design based on the driving force of the separation. The algorithms deal with situations ranging from design of a single column to the design of a distillation sequence. The idea is that performing the separation with the largest driving force first leads to the minimum energy

requirement for the separation. This approach only accounts for the operating costs of separation and capital costs are not considered.

2.4.5 Stochastic methods

Stochastic methods use random changes in flowsheet structure in order to search for the optimal solution. In general, these methods will provide a global optimum in infinite time.

Simulated annealing

Simulated annealing (Kirkpatrick et al., 1983) has been applied to separation synthesis (Floquet et al., 1994). Simulated annealing (SA) is based on an analogy with the cooling of a molten material. If the cooling is carried out quickly, there will be irregularities in the structure of the crystals that are formed. The slower the material is cooled the less irregularities there will be in the structure. The more orderly the structure, the lower the energy level of the crystal. A perfectly formed crystal represents the lowest possible energy. In process synthesis, the configuration of the crystal corresponds to a feasible solution structure and the energy to the cost. The general algorithm works by gradually reducing the start temperature with time. The structure of the initial feasible structure is encoded. The encoding is then altered randomly subject to certain rules that ensure a feasible solution. The cost of this structure is evaluated. If it is smaller than the original, the solution is accepted. If not, it is accepted with a probability of $e^{\frac{-\Delta E}{T}}$. Thus, changes that result in cost increases are more likely to be accepted early on when the temperature is greater. These moves prevent the algorithm from becoming stuck in the area of a local optimum. A basic SA algorithm is shown in figure 2.1.

input T_{stop}
input T_{start}
input A, the number of moves per annealing
input S, the starting structure
Calc.E, the objective function value of structure S
Temp. $T = T_{start}$
while $T < T_{stop}$ and success = true **do**
  success = false
  for $i = 1$ to A **do**
    Select a new random structure, S^*
    Calc. objective function value of S^*, E^*
    $\Delta E = E^* - E$
    if $\Delta E < 0$ **then**
      $S = S^*$
      success = true
    else
      if random()$< e^{\frac{-\Delta E}{T}}$ **then**
        $S = S^*$
        success = true
      end if
    end if
  end for
  $T = Tf$
end while
print results

The function random() yields a random number between 0 and 1 and f is a factor between 0 and 1.

Figure 2.1: A simulated annealing algorithm

The procedure for separations synthesis outlined by Floquet et al. (1994) is applied to separations using distillation. Both simple, two product, and more complex, side stream dis-

tillation columns are allowed for in the encoding. However, only sharp splits are allowed. The algorithm is used to solve a large 16 component hydrocarbon separation problem, in which the feed is to be separated into pure components. There are 5.9×10^{11} possible structural combinations for this problem if two and three product distillation columns are allowed. A saving of around 50% over the inital flowsheet cost is reported.

A SA method that allows rigorous distillation models to be used is presented by Marcoulaki et al. (2001). The separators used are simple one feed two product columns. In addition, non-sharp separations are allowed by discretizing the recovery fractions in steps of 1% between 70% and 90%. This leads to about 1.2×10^{14} possible flowsheet structures for the 15 component hydrocarbon separation synthesis problem attempted. This size of problem would be prohibitively expensive, in terms of computer time, if each possible flowsheet were to be evaluated.

Linke and Kokossis (2003b) compare the synthesis of reaction/separation processes using SA and a tabu search. The Tabu search is another stochastic technique where new structures are selected that are in the neighbourhood of the current structure. In simulated annealing this is a random move and the search direction is guided by the success of the new structure and the stage of the search. The Tabu search determines the direction by *remembering* recently tried modifications and not allowing such changes for a certain number of iterations. It was found that Tabu searches tended to take significantly shorter paths than SA to arrive at solutions of similar quality.

Genetic algorithms

Another approach to the separation synthesis problem is the application of genetic algorithms (GA) (Wang et al., 1998). Genetic algorithms are based on an analogy with

Darwinian evolution in nature. A population is composed of a number of individuals. The most successful individuals are those lowest in cost. The more successful individuals are allowed to breed and transfer their characteristics to offspring. In addition random mutations of the population can occur which may or may not be beneficial. This has been applied to sharp distillation separation sequencing by encoding the possible structures. The flowsheets are optimised for annual operating cost with heat integration included.

Neither genetic algorithms nor simulated annealing can guarantee the globally optimal solution in a finite time. This is because the solution space is not systematically explored. Even though a superstructure is not required explicitly, certain assumptions must be made about the solution structures when the encoding procedure is devised. However, these techniques can be applied to large combinatorial problems and reduce the possibility of becoming stuck in local optima.

2.4.6 Implicit enumeration

The implicit enumeration approach to process synthesis dispenses with the need for the prior development of a superstructure. Consequently the user does not need to impart as many preconceived ideas on the development of the process as may happen when developing a superstructure or the encoding system for a stochastic algorithm. Implicit enumeration may be more likely to yield a radically different, and therefore patentable, structure (Johns, 2001). However, in terms of separation synthesis, the approach is mainly applicable to single feed problems where the desired products are pure components. This is due to the difficulty of introducing recycles and handling multiple feed streams. The automated generation of recycle streams has been demonstrated by Fraga (1998). The technique is based on identifying structures that partially meet the product requirements.

These structures have a recycle stream incorporated if any units are involved in conversion rather than just separation.

Potential for novel solutions

The following describes the potential of implicit enumeration to the separation synthesis problem. Methods that require the definition of a superstructure usually only allow one distillation column per possible split point between *light* and *heavy* keys. For a five component mixture to be separated into pure components, a minimum of four distillation columns are required. The rigorous superstructure generation approach (Kovacs et al., 2000) allows more columns than this but these are present in order to solve the multi-component product problem that involves dividers and blenders.

It may be the case that the optimal solution to a problem requiring pure component products involves two different columns that carry out the same *light/heavy* key separation task. The two columns would share the burden of the separation task but each would require a less fine separation. This division of separation duty could prove to be a more cost effective solution. This could not be incorporated into the P-graph maximal structure generation methodology (Friedler et al., 1993) as only sharp separations are allowed.

The Jacaranda process synthesis system uses implicit enumeration to solve the synthesis problem. Previous approaches to implicit enumeration are described by Johns and Romero (1979) and Fraga and McKinnon (1994). Using this approach, solutions are created and evaluated simultaneously. It does not have the problem of settling in local optima due to non-convexities as the discretised space is searched systematically. This allows the whole search space to be traversed. The following section describes the Jacaranda implementation of implicit enumeration in more detail.

2.5 The automated synthesis package Jacaranda

Jacaranda (Fraga et al., 2000) is implemented in the object oriented programming language, Java. Objects are used to represent streams and unit models. Both stream variables and unit model operating variables are mapped to discrete space. This is carried out in order to make the search space, that is to be enumerated, finite. A technique called dynamic programming is used to reuse solutions to problems that have already been encountered. This can dramatically increase the efficiency of the procedure.

2.5.1 The search algorithm

The algorithm is based on a depth first traversal of the superstructure graph. This graph is itself generated as it is searched. The problem can be described by equation 2.1 (Fraga and McKinnon, 1994).

$$f(s) = \min_{u} \left(c(u,s) + \sum_{i=1}^{n_p} f(p_i) \right) \tag{2.1}$$

u is the possible range of units that may be used to process a stream, s and $c(u,s)$ is the cost of processing s to yield n_p product streams. The function, $f()$ is the cost of the solution to a subproblem. Equation 2.1 is evaluated recursively until a stream meets a product specification. At this point the problem associated with the stream has been solved. The user may specify if it may be desirable to process the stream further or to stop if a stream meets any one of the product specifications. The costs of the alternatives for a particular stream are compared and a list of the best solutions is created.

The feed stream

In practice, in order to optimise the synthesis of a particular process, Jacaranda requires a feed stream, a list of unit operations available and a list of product specifications. The component flows of the feed are discretised to be a number of basic units of flow. The value of the *base* level for each component is user specified. The user also specifies a stream pressure range and a number of discrete levels available. The feed stream pressure is mapped to the nearest discrete value. The program then attempts to process the feed stream with one of the available unit operations. This list includes product tanks that each represent a product specification.

Unit designs

The particular unit model may define one or more discrete design alternatives. The range and number of possible values of unit variables are also user specified. For example, the operating pressure of a distillation column may be allowed within a certain range. The user specifies a number of discrete values that the operating pressure may take. Each of these pressure levels represents an alternative design for the unit. Each of these designs are carried out for the feed stream yielding values for operating cost, capital cost and any other value that is incorporated into the unit model. Each design produces output streams which are recursively treated in the same way as the feed. Thus the search graph is simultaneously created and searched. When a stream meets a product specification the solution is passed up a level of recursion where a list of the best solutions is compiled. The user may specify n the number of ranked solutions to be stored (the *nbest list* solutions. The solutions may be ranked on the basis of one or more criteria. In this way solutions are passed back to the base of the tree until all the alternatives for the processing of the feed

stream have been attempted and a list of the best flowsheet structures can be identified. The algorithm is shown in figure 2.2.

```
function solve(problem p)
  boolean solved = false
  initialise nbest, an empty solutions list
  stream F = p.feed
  if F processed already then
    solved = true
    retrieve nbest solutions
  end if
  if solved = false then
    for each unit type do
      for each design alternative d do
        create node N(F,u,d)
        for Each d.product o do
          create problem p(o)
          solve(p)
        end for
        attempt to insert N.solution into nbest
      end for
    end for
  end if
end function solve
```

Figure 2.2: The Jacaranda search algorithm

Solution processing

The solution to a node is inserted into the *nbest* list if the value of the objective function for this node is lower than for one or more current members of the list. It can be specified that only one *similar* solution may be present in the list. The level of detail at which the solutions are compared can also be specified (Fraga, 1996). This choice changes the level of detail of the solution encoding. Unit type and alternative can be included in this encoding or the comparison may be solely based on solution structure.

2.5.2 Re-use of solutions

Jacaranda allows solutions to streams to be re-used using dynamic programming. For a detailed explanation of this procedure see Fraga (1996). When a particular stream is solved at any point in the search graph, the solution is stored. This storage method relies on a string encoding that is unique to that stream. The string is made up of the number of basic flow units of each component and the discrete pressure level. If this stream is encountered at another point in the search, the solution is then recalled saving computer time. Thus, the stream discretisation aids in the re-use of solutions. Generally the more coarse the level of stream discretisation, the more likely a particular encoding will appear elsewhere in the search.

2.5.3 The consequences of discretisation

The discretisations used in the procedure described above have several implications. The level of these discretisations is set by the user. In some cases, user intervention may be appropriate. For example, some components may be more important than others for

environmental or economic reasons. The *base* flow rate of these components would be set to lower values than the others. In addition, the engineer may know the pressure range of operation for distillation columns and base the range and level of discretisation on this knowledge. See Laing and Fraga (1997) for a discussion on the iterative use of an automated procedure with particular emphasis on user interaction. Typically, however, the engineer may have no insight on the level of discretisation required for a given synthesis problem. Furthermore, the solutions generated give no indication of the effect of the discretisation on the effectiveness of the search procedure.

The solutions produced will consist of units and stream products. Each unit variable value will be at one of the discrete levels set when the synthesis problem was formulated. The best solution for a given level of discretisation may not represent the optimal flowsheet structure in continuous space as potentially good values for the discretised variables may be missed between the discrete levels chosen.

2.6 Summary

This chapter has described various methods that may be applied to the separation synthesis problem. Heuristic methods such as hierarchical decomposition are still the most widely applied, but are likely to yield sub-optimal flowsheets. This has led to the development of various algorithmic techniques.

Mathematical programming strategies are able to tackle a wide range of types of synthesis problems but generally do not ensure global optimality. In addition, such approaches require a superstructure to be constructed beforehand. This process in itself could result in novel solutions being missed. The P-graph approach can be used to rigorously generate

a superstructure but the designer still has to define all possible outputs from operating units, again leading to the possibility of missing optimal structures.

Another major strategy is the use of stochastic optimisation methods such as simulated annealing and genetic algorithms. An advantage of these approaches is their ability to tackle combinatorially large synthesis problems and their ability to escape from local optima. These methods do not require the prior generation of a superstructure but it is necessary to make assumptions about the nature of the solution when devising an encoding method. Another drawback is that the globally optimal solution cannot be guaranteed in finite time.

Implicit enumeration has the advantage that no prior assumptions about the structure are necessary. All that is required by the Jacaranda package is a feed stream, a list of units and set of product specifications. This increases the likelihood of the generation of novel solutions. A disadvantage of using implicit enumeration is the need to discretise continuous unit variables to yield a finite search space. Jacaranda also discretises stream flows and pressures which increases the efficiency of the dynamic programming. The user does not gain any information on the effect of chosen level of discretisation. It also means that the top-ranked solution is not necessarily optimal. This project is motivated by the goal of developing a procedure that has the benefits of implicit enumeration described above along with the assurance of the optimality of the solution. This leads to the idea that interval analysis can be applied in order to realise this goal. The costs of solutions yielded by the search procedure can be bounded by the application of interval analysis to unit and stream calculations. This information can then be used to discriminate between solutions based on their objective function bounds. The use of the properties of interval arithmetic can ultimately be used to isolate the globally optimal solution. The concept of intervals, interval arithmetic, and how it can be applied to the pure component separation synthesis problem, are discussed in the next chapter.

Chapter 3

Interval analysis

3.1 Introduction

This chapter explains the concepts of interval analysis in the context of the separation synthesis problem. Interval methods have been used previously within process engineering. For example, they have been applied to find all roots to an equation with mathematical certainty (Schnepper and Stadtherr, 1996). This interval approach was tested on several chemical engineering simulation problems. Interval analysis has also been applied to the global optimisation of selected flowsheets (Byrne and Bogle, 2000).

Interval mathematics was first introduced by Moore (1966). An interval is a closed bounded set of real numbers, $X = [a, b]$, where $a \leq x \leq b$. An interval of zero width (i.e. with the same values for both lower and upper bounds) is called a *degenerate* interval. In the discussion that follows, interval variables will be denoted by uppercase letters and real variables by lower case letters. The bounds of the interval are shown by square brackets enclosing the real lower bound followed by a comma and then the real upper bound e.g $[a, b]$.

When using the implicit enumeration search embodied by Jacaranda, unit and stream variables are discretised primarily to give a finite search space. As discussed previously, it would be useful to bound the effects of these discretisations using interval arithmetic. For example, the problem may involve separation by distillation. A solution describes a flowsheet structure containing a series of distillation columns each at a discrete operating pressure. The columns operate over a range of pressures between 1 and 10 bar. It is decided that 10 discrete pressure alternatives are to be used over this range. The discrete values are uniformly spaced. Hence, the discrete values would be accurate to the nearest bar. An alternative at 2 bar would actually represent a range of values with a lower bound of 1.5 bar and an upper bound of 2.5 bar. This range of values can be represented by the interval [1.5,2.5]. The same applies to all discretised variables, both in streams and units. If all these discretised variables are bounded in the same manner then interval arithmetic can be used to bound the effects on the objective function.

3.2 Interval arithmetic

A set of arithmetic operations can be defined for intervals that correspond to the operations on real numbers. If X and Y are both intervals, $X \mathrm{op} Y$ will yield an interval containing every possible number that can be calculated resulting from the operation of each $x \in X$ on each $y \in Y$. The following rules (Hansen, 1992) can be produced from this definition, given $X = [a, b]$ and $Y = [c, d]$:

$$X + Y = [a + c, b + d] \tag{3.1}$$

$$X - Y = [a - d, b - c] \tag{3.2}$$

$$X \times Y = [\min(ac, bc, ad, bd), \max(ac, bc, ad, bd)] \tag{3.3}$$

In order to divide the inverse of the denominator is calculated:

$$\frac{1}{Y} = \left[\frac{1}{d}, \frac{1}{c}\right] \tag{3.4}$$

$$\frac{X}{Y} = X \times \left(\frac{1}{Y}\right) \tag{3.5}$$

so long as $0 \notin Y$. If $0 \in Y$ then extended interval arithmetic can be used (Hansen, 1992). Rules for this situation are as follows.

$$\frac{X}{Y} = \begin{cases} [b/c, \infty] & \text{if } b \leq 0 \text{ and } d = 0 \\ [-\infty, b/d] \cup [b/c, \infty] & \text{if } b \leq 0 \text{ and } c < 0 < d \\ [-\infty, b/d] & \text{if } b \leq 0 \text{ and } c = 0 \\ [-\infty, \infty] & \text{if } a < 0 < b \\ [-\infty, a/c] & \text{if } a \geq 0 \text{ and } d = 0 \\ [-\infty, a/c] \cup [a/d, \infty] & \text{if } a \geq 0 \text{ and } c < 0 < d \\ [a/d, \infty] & \text{if } a \geq 0 \text{ and } c = 0 \end{cases} \tag{3.6}$$

Exponents can also be defined:

$$X^n = \begin{cases} [1,1] & \text{if } n = 0 \\ [a^n, b^n] & \text{if } a \geq 0 \text{ or if } a \leq 0 \text{ and } n \text{ is odd} \\ [b^n, a^n] & \text{if } b \leq 0 \text{ and } n \text{ is even} \\ [0, \max(a^n, b^n)] & \text{if } a \leq 0 \leq b \text{ and } n \text{ is even for } n = 0, 1, 2, ... \end{cases} \tag{3.7}$$

3.3 Interval functions

An interval function will yield an interval when applied to one or more interval arguments. An interval function, F, is said to be an interval extension of a real function, f, if

$$F(x) = f(x) \qquad \forall x \in \mathbb{R} \tag{3.8}$$

F is defined as an interval extension of f, if the results of evaluating them both over the same vector of degenerate intervals are equal.

The *natural* interval extension of a function f is to replace the variables of the real function with interval variables. There are, in fact, an infinite number of interval extensions of a function. An interval function is said to be *inclusion monotonic* if $X_i \subset Y_i, i = 1, ..., n$ implies that

$$F(X_1, \ldots, X_n) \subset F(Y_1, ..., Y_n) \tag{3.9}$$

Interval functions, containing a sequence of interval addition, subtraction, multiplication and division operators, are inclusion monotonic (Hansen, 1992) if the interval extension retains the same form when evaluating X and Y. The following example illustrates this

point (Caprani and Madsen, 1980). The function

$$f(x) = x(1 - x) \tag{3.10}$$

could be rewritten as

$$f(x) = c(1 - c) + (1 - 2c)(x - c) - (x - c)^2 \tag{3.11}$$

where c is a constant real number. If the real x is replaced by the interval X then equation 3.11 represents a set of interval extensions that differ by the value chosen for c. The *natural* interval extension of equation 3.10 can also be written. Evaluating the two forms for real values of x and any value of c always yields the same result. This does not hold for interval extensions of these functions. Let $X = [0, 1]$ and c is the midpoint of X, $c = m(X) = 0.5$. Evaluating $f(X)$ in the revised form represented by equation (3.11) yields $[0, 0.25]$. If X is replaced with $X' = [0, 0.9]$ and c with $c' = m(X') = 0.45$, $f(X') = [0, 0.2925]$. Inclusion monotonicity has not held: $X' \subset X$ but $f(X') \not\subset f(X)$. The reason that inclusion monotonicity fails is that the form of the function was different for each evaluation. Both the functions that were evaluated are interval extensions of equation (3.10) but they differ in form due to the different values of c.

If an interval function, $F(X_1, ..., X_n)$, is an inclusion monotonic interval extension of a real function $f(x_1, ..., x_n)$, then $F(X_1, ...X_n)$ contains all the possible values of $f(x_1, ..., x_n)$ for all $x_i \in X_i (i = 1, ..., n)$ (Hansen, 1992). This result will prove useful in bounding the value of the global optimum in an optimisation procedure.

3.4 Dependency

The interval returned by an interval function depends on the form that the function takes. For example,

$$\begin{aligned} F_1(X) &= X^2 - X - 3 \\ F_2(X) &= (X - 1/2)^2 - 3\frac{1}{4} \end{aligned}$$

are both interval extensions of

$$f(x) = x^2 - x - 3$$

yet they do not yield the same result when evaluated:

$$\begin{aligned} F_1([1,2]) &= [-4,0] \\ F_2([1,2]) &= [-3,-1] \end{aligned}$$

F_2 produces sharper bounds for the range of f over the interval $[1,2]$ than F_1. This is due to the *dependency* phenomenon associated with interval arithmetic. Generally, the more often a given variable occurs within a function, the wider the bounds become. In fact, F_2 yields the exact range of f for $X = [1,2]$ as X only occurs once in the function. When evaluating interval functions, dependency should be kept to a minimum so as to keep the bounds as sharp as possible.

3.5 Thick and thin functions

The term, *parameter*, will be used to refer to the constant values, either real or interval, within a function. The argument is the value of the function variable at which the function is evaluated. A *thick* function has interval valued parameters whereas a *thin* function has only real valued (or degenerate interval valued) parameters. A thin interval function evaluated on a degenerate interval argument will return a degenerate interval; a thick function would return an interval value.

The interval methods implemented will all involve thick functions. This is because a discretised variable can be represented as an interval spanning the possible range of real values that could have been mapped to that discrete value. *Thick* functions result when these intervals are used in design equations.

For example, the stream pressure discretisation regime may be the same as that described at the start of this chapter. 10 pressure levels, distributed uniformly between 1 and 10, are allowed. During discretisation a stream at 5.3 bar would be mapped to a discrete value of 5 bar. The set of real values that would be mapped to this discrete value is represented by the interval $[4.5, 5.5]$. In order to calculate the bubble point of the stream over this possible range of pressure, equation 3.12 must be solved for temperature:

$$\sum_{i=0}^{i_c} k(T,P)x_i = 1 \tag{3.12}$$

where k is the equilibrium constant and x_i is the liquid fraction of component i.

Pressure is an interval value due to discretisation. Since the function is to be solved for temperature over this range of pressure, the value of pressure in the function is a constant

interval. Even if the function is evaluated for a real value of temperature, the result would be an interval. Hence, the function is *thick.*

3.6 The nominal value

Earlier sections in this chapter have explained that intervals are usually defined by a lower and upper bound. In this work intervals are also defined in this way but with an additional real value which will be termed the *nominal* value. For an interval X, its associated *nominal* value, x_n must be between the lower and upper bounds of X.

$$x_n \in X \tag{3.13}$$

If the *nominal* value is included an interval may be written as $[a, n, b]$ where $a < n < b$. A *nominal* value is associated with each interval because intervals are to be used to cover the continuous space around discretised real values. The *nominal* value represents the discretised point around which its associated interval is constructed. In an arithmetic operation between two intervals the corresponding real operator is applied to their *nominal* values. A numerical example of division would be:

$$\frac{[2,3,4]}{[4,5,6]} = [\frac{1}{3}, \frac{3}{5}, 1]$$

The *nominal* values of the intervals are 3 and 5 and the *nominal* value of the resulting interval is $\frac{3}{5}$, the result of applying real division. The bounds are calculated using the interval arithmetic rules described by equations 3.3 to 3.5.

The nominal value allows real calculations to take place along with the interval calculations. This facility proves useful when analysing results and ultimately, is used in the development of the adaptive algorithm.

3.7 The root of a thick function

The root of thick functions must be located in order to implement Interval analysis within Jacaranda. If a thick function f is evaluated on a degenerate interval or real number x^*, the result will be an interval $f^I(x^*)$. Below are three ways to define whether or not x^* is a root of the function. Each has a different meaning as a root of a thick function.

1. $f(x^*) \cap [-\epsilon, \epsilon] \neq \phi$
2. $f(x^*) \subset [-\epsilon, \epsilon]$
3. $0 \in f(x^*)$

where ϵ is a tolerance used to define an interval, $[-\epsilon, \epsilon]$. A function evaluation within the bounds of $[-\epsilon, \epsilon]$ is approximated to be zero.

The first of these definitions is shown graphically in figure 3.1. This figure shows the upper and lower bounds for a hypothetical thick function with one variable plotted against x, along with a *nominal* value of the function. The function illustrates the case when there is only one root. An interval evaluation for a point x^*, that is within the solution interval, is indicated by $f(x^*)$. The interval root X is shown. All values of x within this interval, when evaluated by f, will yield an interval that has a non-zero intersection with the interval $[-\epsilon, \epsilon]$.

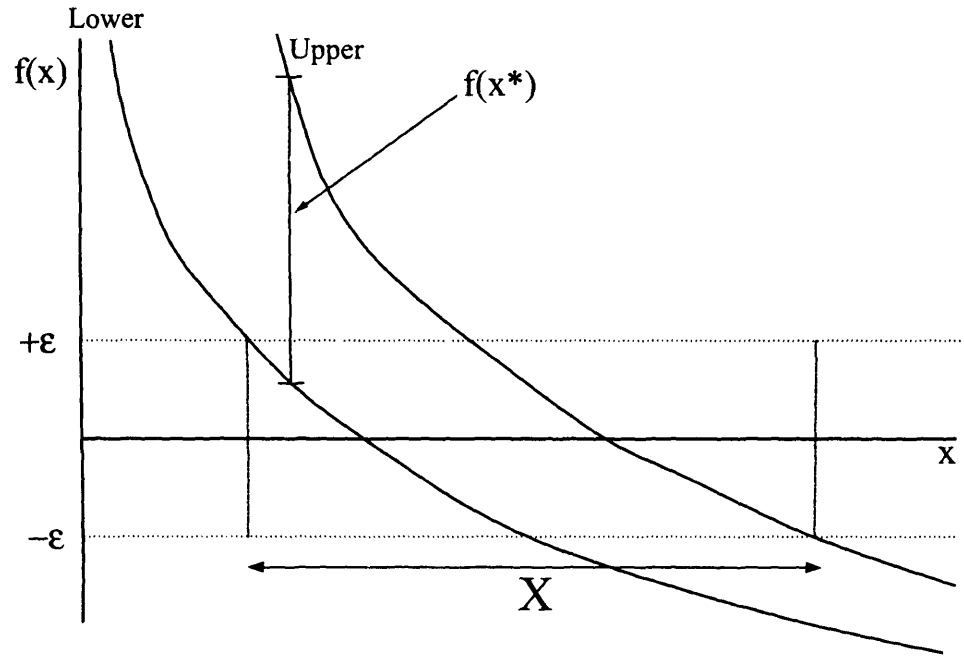

Figure 3.1: Illustration of the solution,X for the criterion that $f(x^*) \cap [-\epsilon, \epsilon] \neq \phi$

The worth of any of these definitions of a solution depends on what is required of the solution. For example, it may be required to solve a design equation where one or more physical properties are known to lie within one or more intervals. This causes some of the parameters of the design equation to be intervals and hence the equation is a thick function. If the root is found for a variable x, the solution will be an interval, X.

If the root is defined by the first criterion then every value of x within the solution interval will certainly lead to a design that satisfies the real design equation. However this will not be the case for all values of the interval parameters in the equation. Some combinations of parameter values will lead to designs and some will not. A given value of x may not yield a design as the whole of the interval $f^I(x)$ is not necessarily within the interval $[-\epsilon, \epsilon]$ under this criterion. Since one or more design parameters are intervals, there will be a viable design for one or more real values within the interval parameters, but not necessarily for all values. This definition of a root will be useful when investigating the range of possible designs, but one cannot be sure that the designs will be viable for all values of the interval parameters. However, costs resulting from unit designs are strictly bounded. This property is useful if certain designs and structures are to be discounted, on the basis of cost intervals, in an optimisation algorithm. In such a case it does not matter that some of the cost intervals may not lead to a design, as it is more important that no valid designs are missed. The contribution of such designs to the cost interval leads to a widening of the cost bounds and a potentially less efficient search but the bounds are still valid.

A graphical representation of the second definition is shown by figure 3.2. This is shown for the same hypothetical function as figure 3.1.This definition is much more restrictive than the previous and will lead to sharper bounds on X. It specifies that every $x \in X$, should yield an interval within the interval, $[-\epsilon, \epsilon]$, when the function f is evaluated on x. This means $f(X)$ has bounds with absolute values that are smaller than ϵ. From

the perspective of solving a design equation, this criterion gives more certainty in the solution. It yields the values of x within X for which a design will be viable for all real values within the interval design parameters.

This second definition may be useful in assessing the effect of the coarseness of discretisations. The width of interval parameters in a design equation result from the level of discretisation used. If a given discretisation does not yield a solution under this criterion then the discretisation could be made progressively finer until a solution is found. The confidence in the viability of the unit design would then only be dependent on the accuracy of the equations used. One drawback of this definition is the fact that the width of the interval parameters in the function may need to be narrow before there are any values of x for which $f^I(x)$ is within $[-\epsilon, \epsilon]$. In fact, there may not be any solutions using this criteria as the upper and lower bounds of the function may be too wide near the root. This definition could not be used when the costs of all possible structures are to be bounded, as feasible designs could be missed.

The third definition dispenses with the need for the concept of tolerance, ϵ. It is represented graphically for the same hypothetical function in figure 3.3. This criterion stipulates that for x to be within the solution interval, X, the evaluation of the function on x must yield an interval that contains zero within its bounds. An example of a value that meets this criteria is indicated in figure 3.3 by $f(x)$. This shows that the bounds do not need to be within $[-\epsilon, \epsilon]$. The width of a solution determined by this definition depends upon the gradient of the upper and lower bounds of the function as it crosses the x axis. This definition may yield wider or narrower solutions than the second. This depends upon the value of ϵ used in the second definition and the gradient of the function around $f(x) = 0$. Under definition 2, values of $f(x)$ for x within the solution interval, X, do not necessarily have to contain zero. Under those circumstances, the width of solutions may be wider than for definition 3. There will always be values of x that satisfy definitions

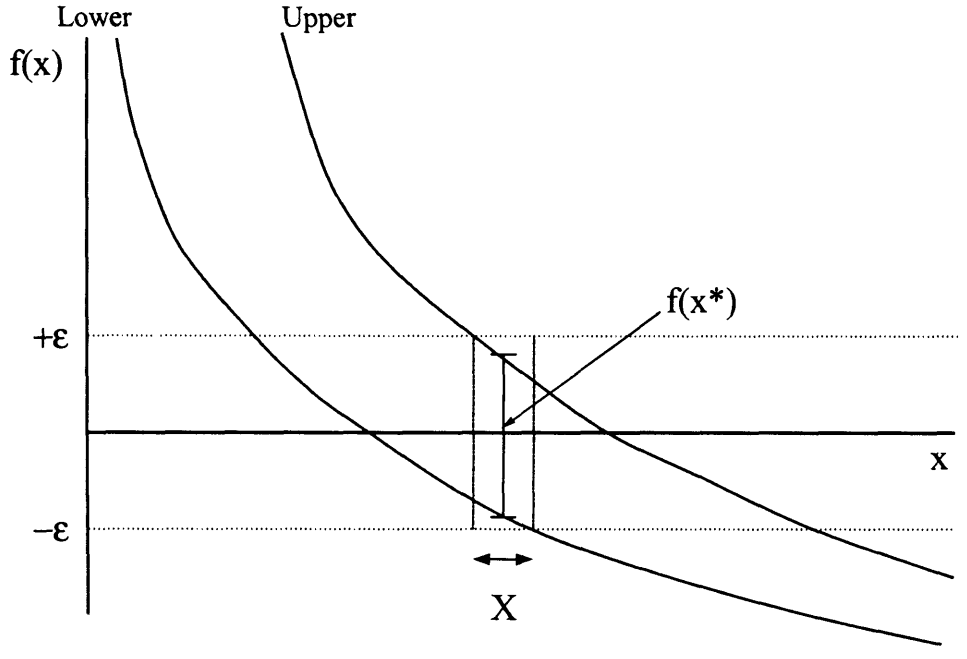

Figure 3.2: Illustration of the solution, X for the criterion that $f(x^*) \subset [-\epsilon, \epsilon]$

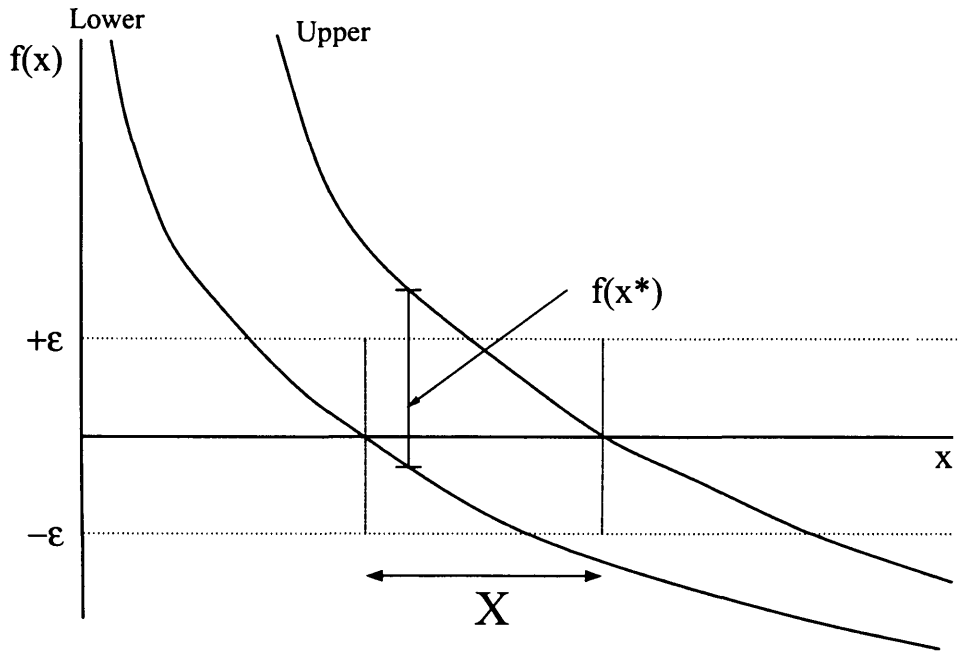

Figure 3.3: Illustration of the solution, X for the criterion that $0 \in f(x^*)$

1 and 3 as long a root of the function exists. A problem may occur using definition 3 if there are two roots close together and the lower bound of the function becomes only slightly negative. In this case, computer rounding error could cause the solution to be missed. As mentioned earlier, this discussion is concerned with equations that have single roots within the range of x being investigated. If this is the case then this problem will not arise. Nevertheless, this definition may cause some of the possible values of x to be missed. Machine rounding error may cause some solutions at the edge of X to be omitted as the interval value of the function at these points may have been rounded up or down so that it no longer contains a zero. In order to obtain the entire possible range of solutions it is necessary to introduce some value of tolerance to account for rounding error. This leads to definition 1.

Definition 1 will always give the widest solutions as the conditions for a value of x being accepted as a root are the most relaxed. For very small values of ϵ that would be used as an approximation to zero, definition 2 will give the narrowest solutions.

The first definition should be used to locate the root of a thick function in the optimisation algorithm being developed. This ensures that no design possibilities are dismissed. For the functions that are to be considered, it is reasonable to assume that there is only one root in the range that is being considered. Therefore, a bisection method is used to calculate the lower and upper bounds separately. If it is necessary to find multiple roots then the Interval Newton method (Moore, 1966) could be used.

3.8 Summary

This chapter has introduced how interval analysis can be applied to an implicit enumeration search for the optimal process structure. The basic concepts of interval arithmetic

and interval functions have been outlined. Using an interval extension of a real function gives bounds on the function value when interval variable values are applied. The functions that appear in the design of units during the search procedure may be *thick* functions, that is that they contain interval parameters. A method has been developed that will allow the root of such functions to be found. The selected solution criterion ensures that all possible solutions are bounded. The following chapter describes the application of these techniques to unit designs. This leads to the bounding of the effects of pressure discretisation in distillation designs.

Chapter 4

Application of interval analysis

4.1 Introduction

This chapter introduces how interval arithmetic can be applied to an implicit enumeration search for optimal process structures. The use of intervals allows the effects of discretisation on the objective function to be bounded. The first attempt at attaining this information is bounding the effects of pressure discretisation when designing a distillation separation sequence using distillation columns.

The process synthesis package Jacaranda, described in section 2.5, has been adapted in order to work with streams and units that use intervals to represent pressure rather than real numbers. The following sections describe the initial application of interval analysis to the separation synthesis problem. The core implicit enumeration procedure within Jacaranda can then be used to create and search the possible flowsheet structures. This technique is applied to a case study involving the separation of a five component hydrocarbon mixture and the results are presented.

4.2 Implementation

Within the Jacaranda framework, the discretisation of pressure, and indeed any other continuous variable, takes place in two distinct areas:

1. The distillation unit model.
2. The process streams.

The way that intervals method are applied to these two areas is described below.

4.2.1 Intervals, units and streams

Java is an object oriented programming language. This has allowed an Interval class to be constructed. The class has methods that correspond to each of the real arithmetic operations that might be used when designing a unit or during stream property calculations. This approach dispenses with the need to *hard-code* the rules of interval arithmetic into each of the calculations performed in units and streams.

Inclusion functions can be constructed from the natural extension of the corresponding real functions. The equations and variables that occurs in a unit model or stream based on real arithmetic are examined. If a real variable is to be represented by an interval, each time it occurs it is substituted by an instance of the Interval class. For example, in this chapter intervals are used to represent pressure ranges of distillation column operation. This means that the pressure of the column is represented by an instance of the Interval class. Calculations involving pressure now use the rules of interval arithmetic and result

in intervals. Ultimately an instance of Interval is produced that represents the range of possible costs of the column.

In practice, the direct substitution described above does not always occur. This is due to the dependency phenomenon where the same interval variable occurs more than once in an expression. In some cases it is possible to deal with this situation by rearranging the equation in order to reduce the occurrences to one. However, this is not always possible and some dependency may be unavoidable. This effect may be compounded when the results of two or more function evaluations are used in a third function. This situation is described by the expression:

$$Y = F(G(X), H(X))$$

If the functions G and H contain the interval, X then this leads to dependency when evaluating the value of Y. Where possible, G and H should be substituted into F and rearranged in order to minimise the occurrences of X.

4.2.2 The distillation unit model

The distillation model is based upon the Fenske (1932),

$$N_{min} = \frac{log\frac{y_{lk}x_{hk}}{x_{lk}y_{lk}}}{log\frac{\alpha_{lk}}{\alpha_{hk}}} \tag{4.1}$$

Underwood (1948) equations

$$\sum_{i=1}^{n} \frac{\alpha_i x_{fi}}{\alpha_i - \theta} + q - 1 = 0 \tag{4.2}$$

$$R_{min} + 1 = \sum_{i=1}^{n} \frac{\alpha_i y_i}{\alpha_i - \theta} \tag{4.3}$$

and the Gilliland (1940) correlation.

$$N = \frac{N_{min} + S}{1 - S} \tag{4.4}$$

where

$$S = 0.5309 - 0.5968 \left(\frac{R - R_{min}}{R+1} \right) - 0.908 \log_{10} \left(\frac{R - R_{min}}{R+1} \right)$$

if

$$\frac{R - R_{min}}{R+1} < 0.125$$

and

$$S = 0.6257 - 0.9868 \left(\frac{R - R_{min}}{R+1} \right) + 0.516 \left(\frac{R - R_{min}}{R+1} \right)^2 - 0.1738 \left(\frac{R - R_{min}}{R+1} \right)^3$$

otherwise. The values for this correlation are from Rathore et al. (1974).

The Fenske correlation is used to calculate the minimum number of stages, when the column is operating under total reflux. The Underwood correlation is used to calculate

the minimum reflux ratio and from this the actual reflux ratio is determined using a reflux rate factor. The Gilliland correlation is used to calculate the actual number of stages based on the results of the first two equations. Capital and operating cost models are provided by Rathore et al. (1974)

The column model assumes semi-sharp separation. Non-key components pass completely into the top and bottom products. The key components are split according to the fractional recovery specified. This was set to be 98% in all cases. Heat exchangers are costed based upon the heat transfer area required. Continuous utilities are available (Rathore et al., 1974). A constant temperature difference of $8.5\ K$ between utilities and the process streams is assumed in order to calculate heat exchanger areas.

The unit model is presented with a feed at a pressure within a certain interval. The column is allowed to operate within a range of 1 to 32 atm. Depending upon the level of discretisation selected this leads to a corresponding number of intervals spanning the pressure range. Coupled with a component selected as the light key, one of these pressure intervals defines a particular unit alternative. The design calculations are performed using interval analysis. The design generated yields interval values for the height, diameter and heat exchanger areas for all possible stream and distillation pressures. These values subsequently lead to capital and operating cost intervals for a particular column design. For each interval value determined, a *nominal* value is calculated. This is a result of the calculation carried out at the midpoint around which the interval is constructed and corresponds to the value of the discretised real value of the variable. If a unit design is successful, the *nominal* values of the design parameters and costs are feasible.

It is necessary to solve equation 4.2 to determine θ, a value between the relative volatilities of the keys. If interval analysis is used, this value is itself an interval,Θ. This is not only because the relative volatility of each component, α, varies with pressure but also because

the pressure of the feed to the column is an interval. The parameter, q, is a measure of the fraction of the feed that is vapour. This is calculated by comparing the enthalpy of the feed at its current pressure to its enthalpy at the column pressure. Since both the feed and the column are within certain pressure intervals, the enthalpy will also be an interval. Q will contain the range of possible real values of q.

Note, the range of α is not as sharp as theoretically possible because of dependency due to the interaction of pressure intervals. This will be discussed further in the next section.

The fact that vapour fraction and relative volatilities are intervals mean that equation 4.2 is a *thick* function. The solution is obtained by a bisection method. This is the most convenient option as it is known that Θ must be between the relative volatilities of the keys and there is only one root between these bounds. Real values of θ within the root interval, Θ must meet the criterion that $f(\theta) \cap [-\epsilon, \epsilon] \neq 0$. This means that for any value of θ, there is at least one possible design for some combination of real values within the Q and α intervals. As explained in the previous chapter this definition ensures that no feasible designs are excluded by the root finding procedure. Hence, the resulting cost interval bounds all possible real values for the cost of the column. In order to yield a feasible *nominal* design, the *nominal* value of the root is calculated from *nominal* values of q and α. This value will always lie within the interval solution due to the inclusion properties of interval arithmetic.

4.2.3 Process streams

Process streams are discretised in terms of component flow rates and pressure. The base component flow rate is set to 10% of the component flowrates in the feed stream. As a result, the semi-sharp column acts as a sharp separator as the small amounts of key

components disappear when stream discretisation is applied. Product tanks accept streams that are over 90% pure in any of the components.

The stream pressure is allowed to take one of the user specified number of discrete intervals. In the case study, presented in the next section, the level of stream pressure discretisation was set to be the same as the distillation pressure discretisation. This means that the pressure level of streams leaving the column is not re-discretised before further processing. This is useful for two reasons:

1. Further discretisation would lead to a widening of bounds, hence a finer discretisation scheme would be needed for the same confidence in the solutions.

2. If the pressure interval of the stream is not altered upon exiting a unit before feeding to the next unit, that means the *nominal* value around which the pressure interval is constructed corresponds to an attainable value. This means that for a given process flowsheet structure the nominal cost can be used as upper bound on the minimum cost of that particular structure.

4.3 A hydrocarbon separation case study

4.3.1 Case study definition

The separation of a 5 component hydrocarbon mixture into pure components has been attempted. It is a synthesis problem posed by Rathore et al. (1974). Table 4.1 shows the composition of the feed stream. The aim is to find the optimal process structure for this task. This problem has previously been attempted using Jacaranda (Fraga, 1998). The

Table 4.1: Feed stream composition

Component	Flow rate (kmol/hr)
propane	45.36
i-butane	136.08
n-butane	226.8
i-pentane	181.44
n-pentane	317.52

results obtained gave no indication of the effect of the discretisation on answer quality. In addition there is no guarantee that a superior structure has not been missed. This is because the appropriate set of pressure conditions for the optimal structure in continuous space may not have been tested by the discrete search procedure. The smaller the number of discrete pressure levels used the more likely it is that the true optimal structure will be missed.

Runs were attempted varying the level of unit and stream pressure discretisation. Three optimisation criteria were specified, each based on the lower bound:

1. capital cost
2. operating cost
3. operating cost + (capital cost/2) (Capital cost amortised over two years)

4.3.2 Results and discussion

Figure 4.1 shows the capital cost of the three *best* solutions ranked according to the lower bound on the capital cost. The position of bars on each line correspond to lower, nominal and upper values of cost. The costs of the solutions are shown for various degrees of

discretisation. As the number of discrete points increases the bounds on the solution become tighter.

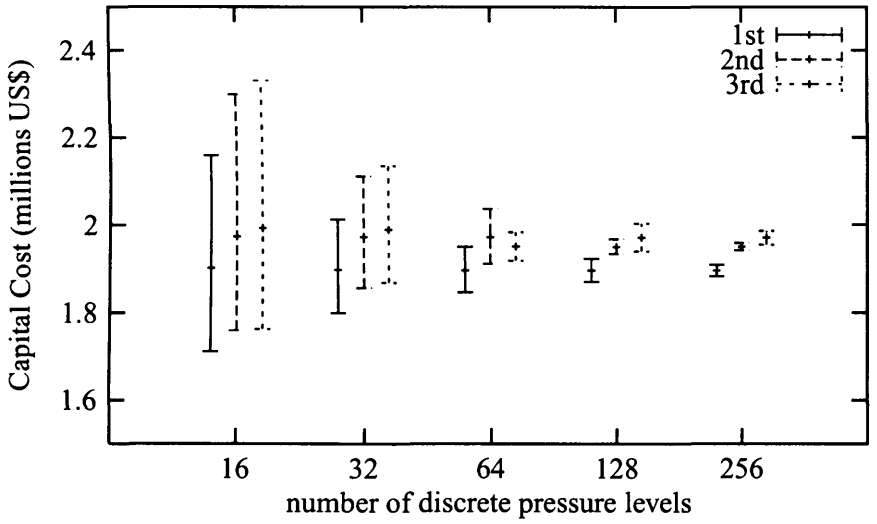

Figure 4.1: Solutions ranked according to the lower bound for capital cost

These bounds are not as tight as possible due to dependency. Nevertheless, the bounds strictly contain all the possible values of the objective function for a particular solution.

A particular solution represents two separate concepts related to the flowsheet being described.

- **Structure** A structure is defined by its constituent units and the way that the units are connected. Two structures can be said to be identical if they contain the same number of each type of unit and the units are linked to each other in the same configuration.

- **Operating conditions** In this case study the operating condition being considered is the pressure of the distillation columns. A particular solution not only describes a

flowsheet structure but the pressure interval within which each column is operated. This means that the cost interval of the solution does not bound all possible costs for the structure, but bounds the cost of operating at a certain set of pressure intervals.

As explained in section 2.5, the search constructs a list of *best* solutions to each problem stream encountered. Two solutions with the same structure are not allowed in the list together. In this situation the solutions are compared based on the objective function value and the one with the lower value is retained and the other solution discarded. For this reason, the criterion used for comparison was the lower cost bound of each solution. This ensures that no other solution with the same structure could possibly cost less. The lower bound of the *best* structure ranked in this way bounds the optimal cost for the case study. Comparing bounds of different structures can allow this optimal structure to be isolated. Note, this is only true if we make the following assumptions:

1. Other discretisations performed by Jacaranda have a negligible effect on the objective function value. The other source of discretisation in this case study is component flow rate. Interval analysis is applied to this aspect of the synthesis problem in chapter 5.

2. Potentially optimal solutions are not rejected due to part of an interval unit variable value being infeasible. For example, a certain distillation operating pressure range may yield a minimum reflux ratio interval that contains negative values. In this situation it is not clear whether or not this is caused by part of the pressure interval being infeasible or whether it is due to the bound widening of dependency. An appropriate discretisation profile can resolve this situation and the issue is addressed in chapter 6. In this case study, above 32 discrete pressure intervals, no designs were rejected for this reason. This suggests that many of the failures at coarser levels of discretisation were due to dependency.

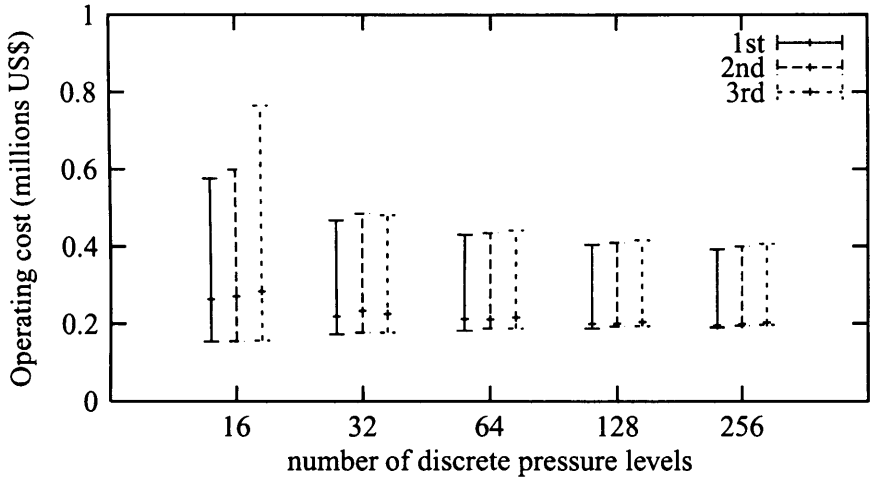

Figure 4.2: Solutions ranked according to the lower bound of the annual operating cost

Figure 4.2 shows the bounds on the annual operating cost for solutions ranked according to the lower bound on the operating cost. As with capital cost, increasing the level of discretisation sharpens the bounds. As a percentage of the nominal value, the bounds on operating cost are much wider than those of capital cost. In particular, the difference between the upper and nominal values is large compared to the difference between the lower and nominal values. This is due to the nature of equations used to determine heating and cooling requirements. Specifically, the large bounds are due to dependencies in the calculation of the heat balance around the column. The enthalpies of the streams are intervals as they are functions of the operating pressure. Furthermore, the feed enthalpy is a function of the stream pressure. The combination of these factors leads to bounds that are far from as tight as theoretically possible.

The original version of Jacaranda determines a solution in terms of discrete values. This gives no assurance that optimal solutions are not missed by overly coarse discretisation. Bounded results can provide this assurance: if the upper bound of the cost of the *best*

solution value is smaller than the lower bound of the second ranked solution, then the optimal structure is represented by the *best* solution. This is a useful result as it allows us to identify the discretisation level to use to be sure that the optimal solution has been generated. However, even more information can be gleaned from the nominal value (calculated from the stream and unit pressure mapped to real values).

As previously mentioned in section 4.2.3, the discrete pressures allowed in streams and columns were kept consistent. The pressure of a stream leaving a column would not change due to the mapping to discrete space as it would already be at one of the stream pressure levels allowed. The nominal value of the unit's operating pressure is never mapped to another value so the nominal value of an optimisation criterion is a feasible value. This is, of course, only true if there were no other discretisations, but as mentioned above, we have assumed that these other discretisations are negligible in comparison with the pressure discretisations.

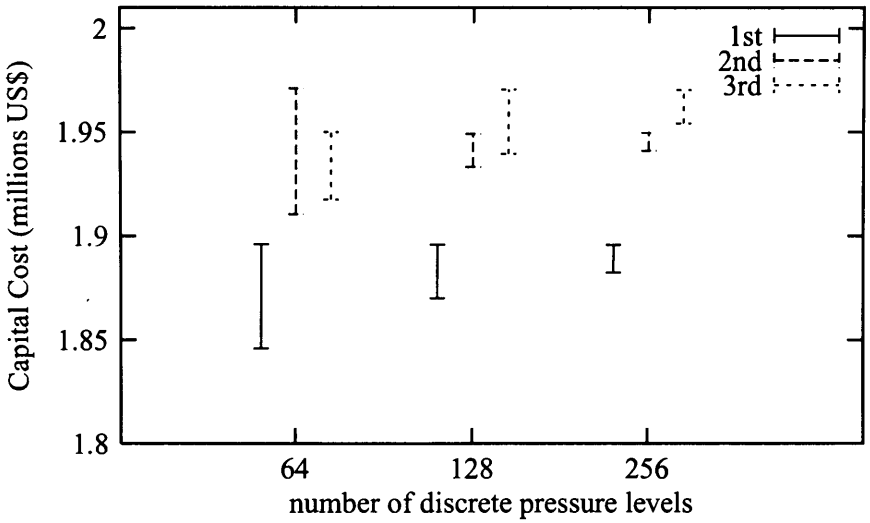

Figure 4.3: Solutions ranked according to the lower bound of the capital cost, using the nominal value as an upper bound.

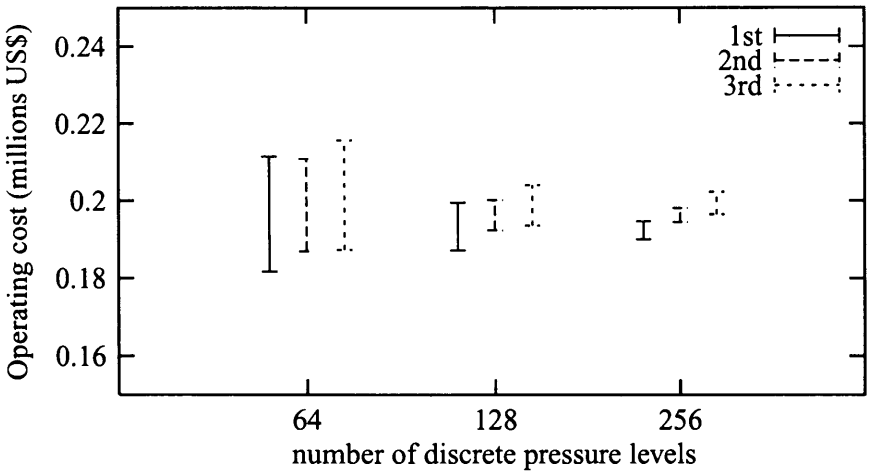

Figure 4.4: Solutions ranked according to the lower bound of the operating cost using the nominal value as an upper bound.

The argument used above when comparing the upper bound of the *best* solution with the lower bound of the second *best* solution can also be applied using the nominal value of the *best* solution. As this nominal value corresponds to an attainable set of real values, it is an upper bound on the global optimum. Therefore, if the nominal value of the *best* solution is smaller than the lower bound of the second ranked solution, the optimal value must be between the lower and nominal values of the *best* solution. We can ignore the range of values above the nominal value for all solutions. Figures 4.3 and 4.4 present the results for the three finest levels of discretisation as a result of this analysis.

Figure 4.5 shows the structure of the top ranked solution for annual operating cost. It cannot be claimed that this structure is optimal as its bounds coincide with those of the second *best* structure but it is certainly cheaper than the third *best* structure.

Figure 4.3 shows that, using 64 discrete pressure levels, the global minimum can be identified. With 256 discrete pressure levels, we can also distinguish between the 2*nd* and 3*rd*

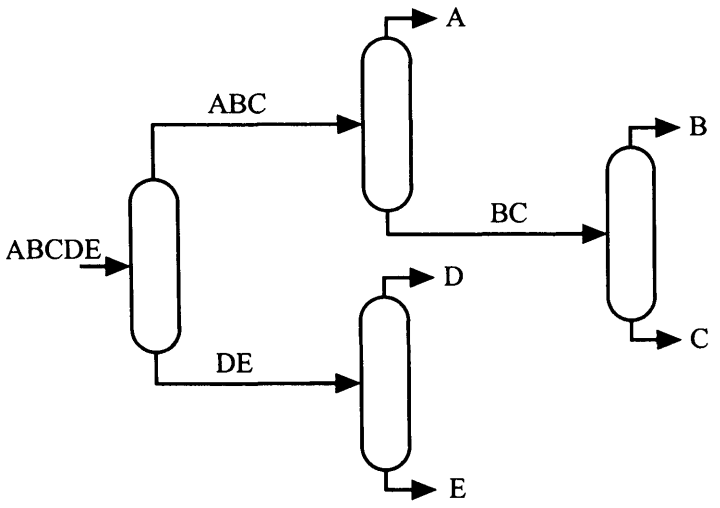

Figure 4.5: The optimal structure when considering annual operating cost

best solutions. Figure 4.6 shows the optimal structure in terms of capital cost.

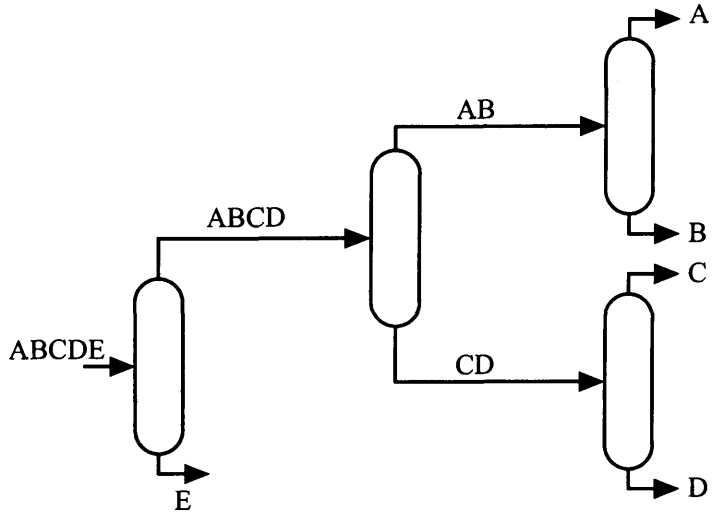

Figure 4.6: The optimal structure when considering capital cost

Figure 4.7 shows the results using an annualised cost criterion, the sum of the operating

cost with the capital cost amortised over two years. With 128 discrete pressure levels, the upper bound for the top ranked structure is smaller than the lower bound on the minimum of the second ranked structure. The top ranked structure in the list is therefore optimal. This structure is the same as for when considering capital cost only, that is shown in figure 4.6. Using 256 discrete levels, the second and third *best* solutions can be identified with certainty.

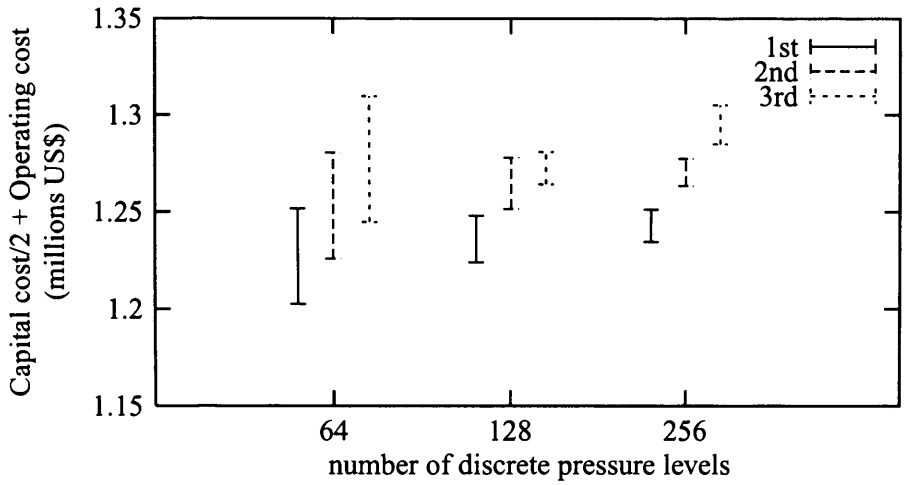

Figure 4.7: Solutions ranked according to the lower bound of the annualised cost, using the nominal value as an upper bound.

Runs were carried out without using the interval bounding method for 4, 8, 16 and 32 pressure levels. For capital cost the top three structures were the same for all these levels of discretisation. For operating cost, the top ranked structures for the runs with 4 and 8 discrete pressure levels were different from those yielded by 16 pressure levels and above. The list of top ranked structures for operating cost based on 16 discrete values was the same as that from the interval bounding procedure. This shows that the runs at 4 and 8 were missing better solutions between the discretised values. For example, the structure

shown in figure 4.5 has been shown to be in the top two structures based on operating cost. From the results of the runs at 4 and 8 pressure levels based on discrete values, this structure does not appear in the top three.

4.4 Summary

Interval arithmetic has been applied to the calculations, involving pressure for the design of a distillation unit. When combined with an implicit enumeration search for optimal separation sequences, the cost of each structure is bounded.

If a clear gap exists between the nominal value of one solution and the lower bound of a subsequent solution, then the former is sure to be a superior solution to the latter. This is, of course, only true with respect to the discretised variable that has been represented by intervals.

The bounds generated for the optimisation criteria are not tight in some cases. Nevertheless, they are valid bounds: the criteria values cannot be outside this range for a given structure and discretisation parameter values within the interval chosen. Furthermore, as the number of discrete levels increases, the bounds on the criteria values become sharper.

If the assumptions that the component discretisation has negligible effect on the solution and that no solutions are missed due to design failure are valid, then the results of the case study identify the optimal structure. This is the case when the objective function is minimised for the lower bound. In the case study, stream and unit discretisation were kept consistent. This means that the optimal structure can be isolated if the lower bound of the second *best* solution is greater than the nominal value calculated for the *best* solution. In effect, the nominal value is an upper bound on the minimum for that solution. Any

uncertainty, that discretisation of the variable analysed may have caused another optimal solution structure to be missed, is removed. If minimisation is carried out on the nominal value, the presence of a solution for this value of the objective function is assured. A lower bound, however, is not necessarily attainable due to dependency in the interval analysis. When developing unit models it is important to keep dependency to a minimum.

Examination of the runs using coarser pressure discretisation shows that the top ranked structures are not optimal. Only from successive runs using finer levels of discretisation can this be demonstrated. Without bounding information from the procedure described in this chapter, a user would have no idea of the quality of the solutions returned. An overly coarse level of discretisation may be used that yields poor sub-optimal solutions.

These results show that the use of Interval analysis has potential in the identification of optimal structures. There are fourteen possible structures that solve this synthesis problem and this method has identified the *best* structure with respect to capital cost and annualised capital cost. However, the approach described has some areas that can be significantly improved.

Using the current method it is necessary to carry out successive runs, each with a different level of discretisation before the optimal solution can be isolated. A superior approach would be able to change the level of discretisation during the search. The search could be made more efficient if variables were no longer discretised uniformly. The search could adapt to discretise more finely in certain areas when necessary. This issue is examined in chapter 6. However, before an adaptive approach is examined, the method must be enhanced to allow all variables to be used in interval arithmetic.

In this case study, only one variable, pressure, was included in the interval analysis procedures. The other discretised variables in this example were the component flow rates. All the other distillation model variables, such as reflux rate factor, were set as exact real

values. It was assumed that the effect of component discretisation would be negligible compared to that of pressure. This will certainly not always be the case. In order to proceed, it is necessary to address the effect of component flow rates on the solution. This effect is taken into account in the following chapter.

Chapter 5

Bounding the effects of flow discretisation

5.1 Introduction

This chapter describes the application of interval analysis in order to bound the effects of discretisation of component flow. Discretisation of the stream variables, pressure and component flow rate, allows efficient re-use of solutions. As discussed previously, good solutions may be missed between these discrete values. In terms of component discretisation, the basic Jacaranda system maps the continuous variables of component flow to the nearest multiple of the user defined *base* flow rate. If the flow rate of a component, on exit from a unit, is nearer to zero than one multiple of this *base* value then the component is removed.

The effect of component discretisation is different from that of pressure. The specification for product streams is often based on the mole or (mass) fraction of one or more

components. Thus, the discretisation that takes place can affect whether or not a stream is accepted as a product, and hence has an influence on the overall process structure. In addition, if a component flow is mapped to zero due to discretisation, it will no longer be considered in the alternatives for unit operations. For example, in distillation design it would not be considered as a key for separation. In order to bound the possible values, the components that disappear due to discretisation must be taken into account. A method of accounting for this and the results of a case study are presented in this chapter.

5.2 Discretisation and the re-use of solutions

5.2.1 Re-use

As explained in section 2.5, Jacaranda increases the efficiency of the search by allowing the re-use of solutions. The flowrate of each component in the stream is mapped to multiples of that component's *base* flowrate. The stream is then encoded based on the multiples of each component present. This yields a string that describes the flows of the discretised stream. When a stream is solved, the solution and stream information is stored in a hash table and referenced by this string encoding. Every new stream that is encountered is discretised and encoded. The hash table is then checked to see if this stream has been encountered previously. If so, the solution is retrieved and no further work is required for this stream. The coarser the discretisation of component flows, the greater the re-use of solutions. Without it, tiny differences in component flows would mean that a two streams were considered different. However, the primary aim of this work is to isolate the optimal structure and re-use is not as important an issue. In implementing the interval bounding method, discretisation of streams could be dispensed with entirely. It is not necessary to

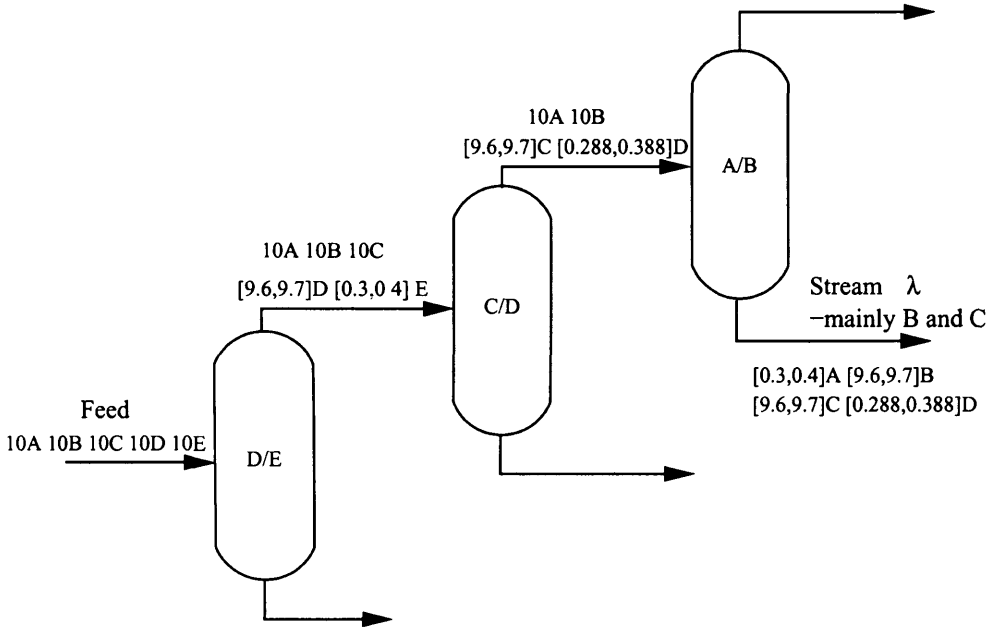

Figure 5.1: Part of a solution structure that requires passage through three columns to produce a stream containing mainly B and C

the feed flowrate of E is 10 kmol/h. Therefore, the flowrate of E in the top stream is $(1 - [0.96, 0.97]) \times 10 = [0.3, 0.4]$ kmol/h. Components A, B and C pass into the top stream in their feed amounts as the column is performing semi-sharp splits. The next column is performing a split with the light/heavy key split between components C and D, and the next column a split between A and B. In both cases a key recovery of between 96 and 97% is used. This results in a bottom stream, λ from the final column containing mainly B and C in the amounts shown in the figure. A very similar stream, μ containing mainly B and C can be generated from the two column structure shown in figure 5.2. It can be seen that the two streams only differ very slightly by the range of D present.

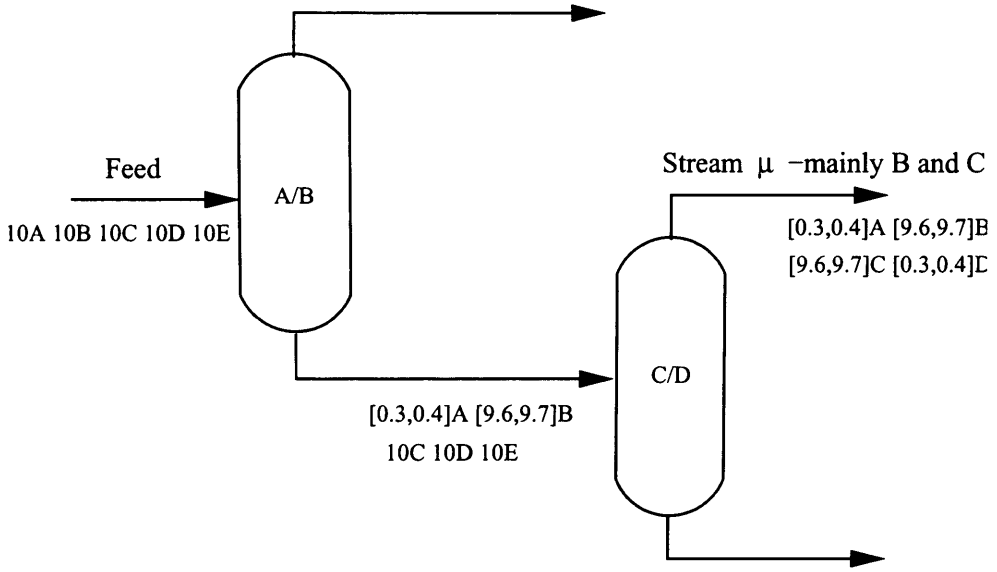

Figure 5.2: Part of a solution structure that requires passage through two columns in order to produce a stream containing mainly B and C

If these streams were to be discretised then a scheme similar to that used for pressure discretisation in chapter 4 could be applied. If the *base* flowrate for each component was 1 kmol/h, then interval ranges could be constructed around the discrete values of $1, 2, 3$

etc. The resulting intervals would be $[0, 0.5]$, $[0.5, 1.5]$, $[1.5, 2.5]$ etc. The flows of each component would be mapped to the interval within which they fell. The result of this mapping on streams λ and μ is shown in table 5.1. The mapped versions of the streams would be seen by the search procedure as identical. If stream λ had occurred first in the search then when stream μ arose the solution to λ could be retrieved and no further work on stream μ would be required.

Table 5.1: The discretisation of two similar streams using standard practice

Component	λ (kmol/h)	μ (kmol/h)	λ mapped (kmol/h)	μ mapped (kmol/h)
A	[0.3,0.4]	[0.3,0.4]	[0,0.5]	[0,0.5]
B	[9.6,9.7]	[9.6,9.7]	[9.5,10.5]	[9.5,10.5]
C	[9.6,9.7]	[9.6,9.7]	[9.5,10.5]	[9.5,10.5]
D	[0.288,0.388]	[0.3,0.4]	[0,0.5]	[0,0.5]

However, as mentioned earlier this discretisation has caused widening of the bounds on component flowrates. This ultimately leads to wider bounds on the cost of solutions and makes the isolation of the optimal structure less likely for a given level of discretisation. Another issue is what should happen when an interval flowrate to be mapped is present in two different discretised intervals. For example component B could have been present in the interval $[9.3, 9.7]$ kmol/h which would leave the decision of whether to map the flow to $[8.5, 9.5]$ or $[9.5, 10.5]$ kmol/h. These factors provide a persuasive argument for not discretising component flows. Removing this discretisation entirely would lead to longer search times as problem stream re-use would fall.

It can be noted that many of the differences between streams occur in the flowrates of components present in small amounts. This is the the case in the above example where the two streams only differ in the flowrate of component D, yet they arise from different routes through a flowsheet. This is one of the consequences of using semi-sharp separa-

tors based on heavy and light keys. Taking this, and the desire to reduce the amount of component discretisation, leads to the idea that discretisation should only be carried out for small flowrates. Using this approach, many of the difficulties described previously can be avoided. For these reasons the concept of the *trace* level is introduced.

If a component is present in amounts below this threshold then the flow of the component is mapped to a *trace* interval value. The *trace* interval has a lower bound of zero and an upper bound at the value of the *trace* flowrate for the particular component. The result of applying this scheme is that all possible values of component flow are bounded. This ensures that resulting flowsheet cost intervals strictly bound the cost of processing the feed stream. Table 5.2 shows what happens if this scheme is applied to the streams λ and μ from the above example. The streams λ^* and μ^* are the results of this operation. With the particular *trace* threshold values chosen, the resulting streams are identical. When one of these has been solved, no further work would be required if the other arose later in the search.

Table 5.2: The discretisation of two similar streams using the trace concept

Component	λ (kmol/h)	μ (kmol/h)	trace threshold (kmol/h)	λ^* (kmol/h)	μ^* (kmol/h)
A	[0.3,0.4]	[0.3,0.4]	0.2	[0.3,0.4]	[0.3,0.4]
B	[9.6,9.7]	[9.6,9.7]	0.5	[9.6,9.7]	[9.6,9.7]
C	[9.6,9.7]	[9.6,9.7]	0.5	[9.6,9.7]	[9.6,9.7]
D	[0.288,0.388]	[0.3,0.4]	0.5	[0,0.5](trace)	[0,0.5](trace)

The value of the *trace* threshold must be chosen carefully and on a component by component basis. If it is too large then cost bounds will be wide and the optimal solution may not be isolated. If it is too small then the amount of solution re-use will be affected.

There will be some loss of re-use as a result of using the *trace* method rather than standard

discretisation. Some streams may be almost identical apart from a difference between the flowrates of a component present in large amounts. This loss of efficiency is necessary in order to keep the bounds on the solutions as tight as possible.

5.2.3 Unit variable discretisation

In terms of the distillation unit model, the key recovery fraction is defined within a range of values. A finite number of real values in this range must be selected as the basis for the unit alternatives. In order to incorporate bounding information, each discrete value is enclosed by upper and lower bounds. These intervals span the whole range of possible values of key recovery. The discrete values themselves correspond to a *nominal* value. When the keys are divided between top and bottom streams, the interval split fractions cause component flow rates in the output streams to be intervals. This mechanism is demonstrated by figure 5.3. The figure represents an overview of the design of a particular alternative for the separation of a four component stream. The design procedure yields bounds on the height and diameter of the column and the size of the condenser and reboiler. These lead to upper and lower bounds on the operating and capital costs of the particular alternative.

Chapter 4 explained how the *nominal* values returned in that case study are attainable, as no discretisation takes place between units. In this chapter, the idea of mapping mass flows, below a certain threshold, to a *trace* interval value has been introduced. When *trace* mapping occurs, the *nominal* flow of a component is mapped to the upper bound of the *trace* interval. This is carried out so that a *trace* interval flows of a component are identical in all cases. Increasing the *nominal* flow of a component will only lead to increased *nominal* costs of any units that it passes through. This is acceptable as the

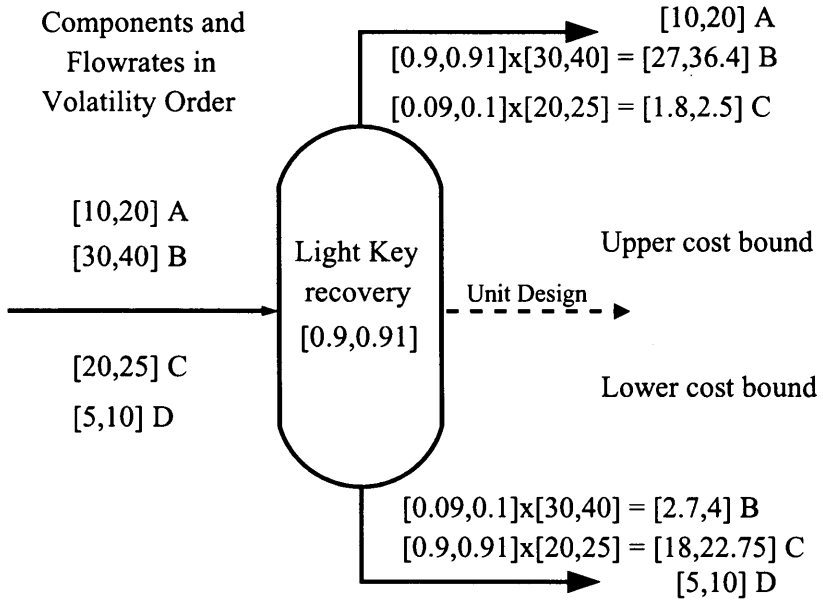

Figure 5.3: Illustration of the interval distillation column design procedure

nominal value still bounds the minimum cost of the structure. The *nominal* value is used as an upper bound in this way when describing the results of the following case study.

5.3 Benzene recycle separation case study

The procedure was tested on a case study involving the separation of benzene. A stream, defined in Table 5.3, must be purified to achieve 98% purity for benzene. This stream is the feed for the separation section of a chlorobenzene process and the benzene is to be recycled back to the reactor. The other components are to be removed as waste, with the requirement that any waste stream contains less than 10 mol% benzene and less than 10 mol% chlorobenzene. Any output stream which consists of > 90% chlorobenzene will also be accepted as a valid product stream. The flowsheet structure with the lowest capital

cost for this separation is required. The components are listed in order of volatility and a key letter is assigned to each component for the purposes of stream encoding.

Table 5.3: Feed for the benzene recycle case study

Component	Flowrate (kmol/s)	Component key
Benzene	0.97	A
Chlorobenzene	0.01	B
Di-Chlorobenzene	0.01	C
Tri-Chlorobenzene	0.01	D
Pressure	1 atm	
Temperature	313 K	

The optimisation criterion is the lower bound of capital cost. This means that the *best* flowsheet ranked on this basis, bounds the cost of the optimal flowsheet. This is true, provided that better flowsheets are not rejected due to partial infeasibility of an interval, as discussed previously in chapter 4.

5.3.1 Results

Distillation units were used with two key recovery intervals together spanning the range 97% to 99.9%. The *trace* level for benzene was set as 0.1% of its flow in the feed stream; all other components had *trace* values of 10% of initial flow rates. The discrepancy is due to the large difference in feed flowrate of benzene compared to the other components. The values of the *trace* threshold levels must be chosen to be consistent with the goals of the synthesis problem. If excessively large values are chosen, the required purity may not be attained, as components that fall below this level are mapped to their *trace* interval values. If the values are too small, the efficiency of the procedure suffers as solutions are not re-used due to small differences in flow rates.

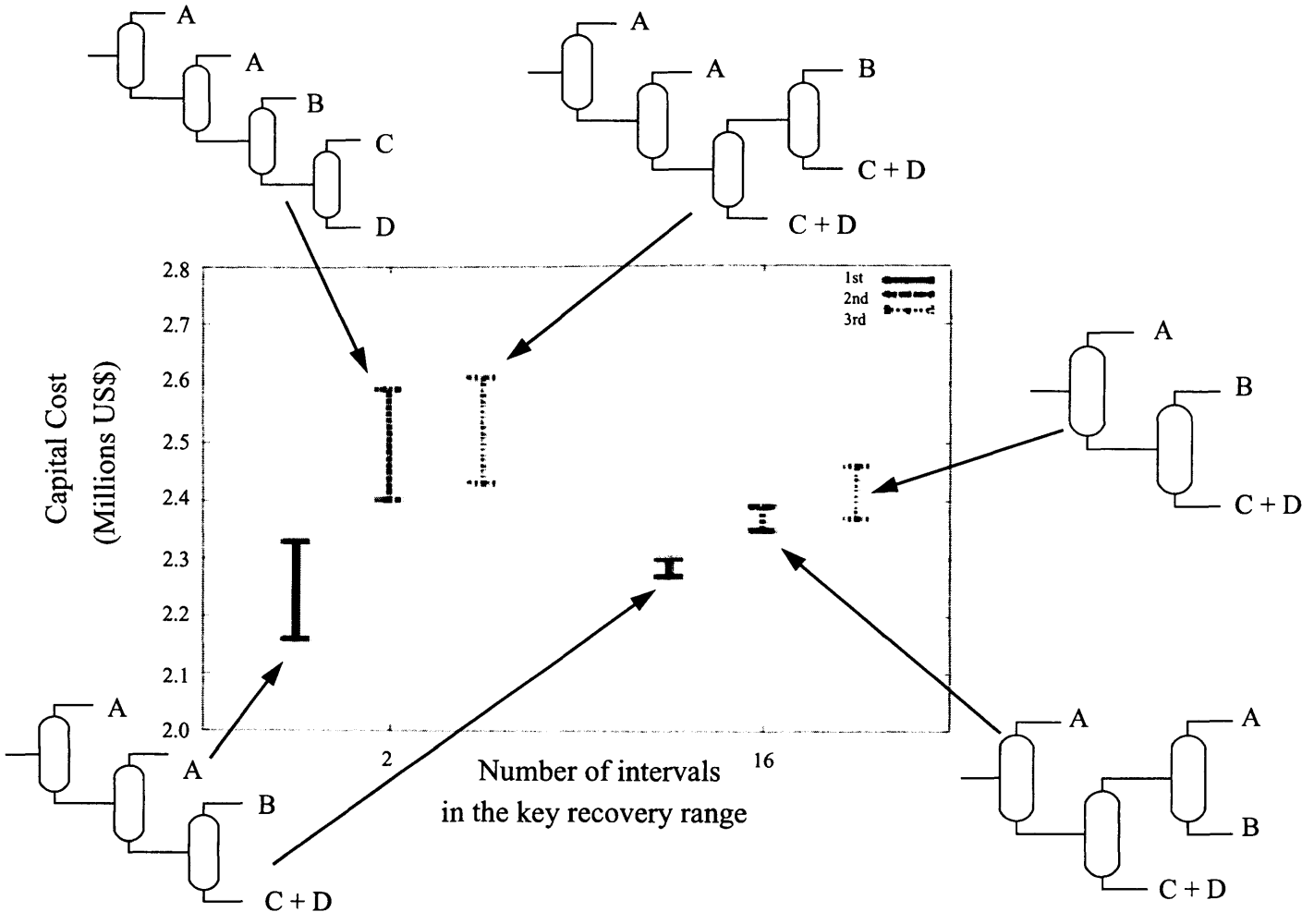

Figure 5.4: Cost bounds and process structures for initial and final runs of the benzene recycle case study

The results of the initial run of the procedure is a list of solution structures ranked according to the lower bound of capital cost. The ranking procedure ensures that only one design with a given structure is included in the list of best solutions. This is achieved by using a text string is used to describe structural information. When a solution is to be added to the list, it is compared to the solutions already present on the basis of this string of text. If the structure of a solution to be added, matches one that is currently present, the two solutions are compared on the basis of objective function value. The solution with the lowest objective function value is retained and the other discarded. The procedure acts in this way since the goal is to differentiate between possible structures, rather than decide upon operating conditions.

Increasing the number of key recovery intervals reduces the size of each interval, but has the effect of increasing the size of the search graph and, hence, the computational effort required to solve the synthesis problem. After an initial run split key recovery into two intervals, several further runs were performed. Each time, the number of key recovery intervals per column was doubled . Figure 5.4 presents the cost bounds and structures of the tops three solutions identified during two different runs of the procedure. The main components in the stream are represented in the figure by the key (A,B,C or D) from table 5.3. The right of figure 5.4 shows the capital cost bounds and the structures identified for the best 3 solutions using 16 intervals over the same range. For both the coarse and fine discretisation runs, the same three unit solution can be identified as the best structure. The isolation of this structure supports the application of an implicit enumeration approach to the problem. This is because in the best structure, the same light/heavy key split takes place in two separate columns. If a superstructure approach had been used, it is likely that this split would only be allowed once in the superstructure. If this was the case the solution would not have been considered.

Statistics for the computational runs discussed are presented in Table 5.4. Solution re-

use is defined as the number of times a previously solved problem stream is encountered divided by the total number of streams encountered. The statistics show that extensive re-use of solutions is maintained under the new scheme. Runs were carried out on a Compaq 850 MHz Pentium III PC, running Linux using Sun Microsystems Java 1.3.

Table 5.4: Computational and search statistics

	Number of intervals	
Statistics	2	16
Problem streams	210	218838
Re-use(%)	60	92
Elapsed time(s)	6	3151

5.3.2 Discussion of results

As mentioned in chapter 4, before the component intervals were introduced, it had to be assumed that flow rate discretisation had no effect on the structures obtained. This work removes the need for this assumption as all types of discretisation that occur can now be bounded using intervals. However, some issues remain outstanding.

Rejection of designs

Figure 5.5 highlights a difficulty with the current method that prevents the optimal structure from being assured. For the sake of discussion, the second best solution from the *fine* run using 16 key recovery intervals, is labelled as structure Z. The 2nd and 3rd best solutions from the *coarse* run, using 2 key recovery intervals, are labelled as structures X and Y respectively. From figure 5.5, it can be noted that structure Z does not appear in the

top 3 for the run using two key recovery intervals. Furthermore, the whole cost interval for structure Z is lower than the cost intervals of the 2nd and 3rd best structures from the coarse run. The bounding information is correct, as it results from interval arithmetic. This means that the run using 2 key recovery intervals per column is missing solutions. If solution Z had not been missed in the *coarse* run, then it should definitely have been ranked higher than solutions X and Y. At that level of discretisation, the cost bounds of Z would inevitably be wider, but the lower bound would be smaller than those of structures X and Y. In fact, structure X would never actually be considered by a designer, as the final column separates di-chloro from tri-chloro benzene, both of which are waste products.

Structures were missed due to the failure of an assumption mentioned in 4. If part of an interval, generated in a design calculation, corresponds to infeasible or impossible values then this alternative is rejected. This may occur due to the bound widening effect of dependency, or it could occur due to part of the unit variable interval being infeasible. During the *coarse* run, part or parts of structure Z must have been rejected during a column design. As the number of intervals in the key recovery range is increased, the likelihood increases that a set of key recovery intervals will lead to a valid structure. As a result of this, it cannot be stated that the structure identified by the *fine* run, with 16 key recovery intervals, is optimal. To make this statement, it would have to be assumed that the uniform discretisation is fine enough to prevent any feasible areas of the search space from being rejected.

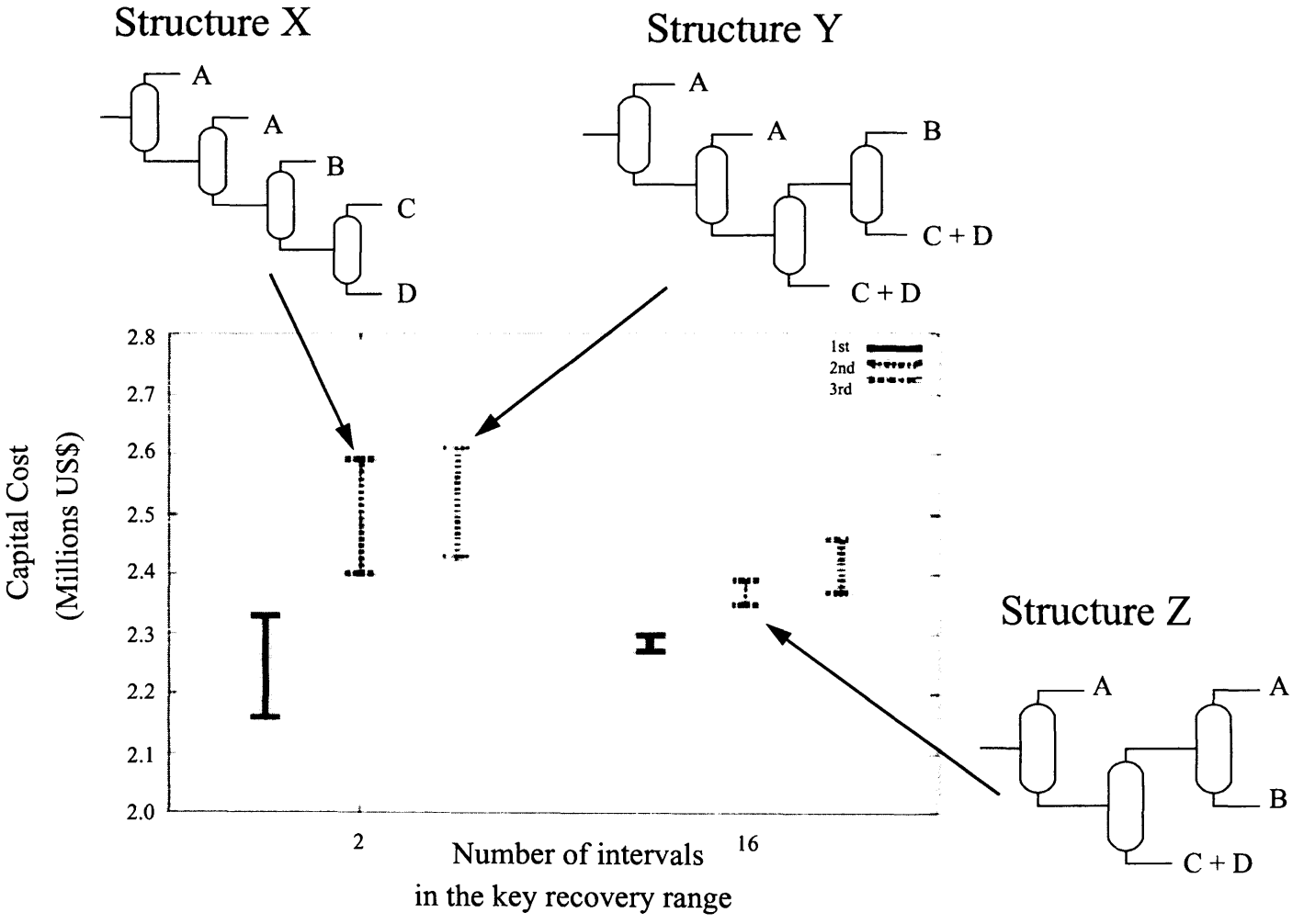

Figure 5.5: Highlighting the discrepancy in the results from two runs on the benzene recycle case study

For a given run of the procedure, each solution obtained is strictly bounded. The user may have confidence in the range of costs returned for a structure, but cannot be positive that a superior structure has not been missed. Designs may be rejected even though part of a design variable interval could produce a valid design. The finer the discretisation used, the more confident the user can be that a better solution has not been missed. At the finest level of discretisation used in this case study, one may be confident that a better solution has not been missed, but one cannot be sure of this. A strategy to handle design failures must be developed, in order to assure a globally optimal solution. In chapter 6, a procedure that takes into account the reason for design failure is introduced. The discretisation profile is selectively made finer in certain areas of the search until it can be proved that no feasible areas have been omitted from the search.

Crossing the *trace* threshold

Another possible difficulty with the approach described in this chapter is that the interval flow of a component exiting a unit may cross the *trace* flow threshold. This occurs when the lower bound of the flowrate is lower than the *trace* flow for the component, and the upper bound is greater than the *trace* flow. Whether the flowrate of a component is above or below the *trace* threshold is important in terms of how it is handled by the distillation unit model. If the whole flowrate interval is above the *trace* flow of the component, then it is considered as a key component. If it is below the threshold, it is considered not to be present in significant enough amounts to be a key component. If it crosses the threshold, the user is informed and the program fails to produce any solutions. With the development of the algorithm discussed in the next chapter, such occurrences can be avoided. An interval design variable can be split repeatedly until crossing of the threshold does not occur.

5.4 Summary

This chapter has shown that component flow discretisation during an implicit enumeration search can be bounded using interval techniques. Introduction of the *trace* interval to describe small flowrates has led to tighter bounds on solutions as discretisation in streams need only occur at the lower flows. At the same time, this allows many sub-problem solutions to be re-used via dynamic programming.

The quality of the solution obtained from the procedure still depends on how finely unit variables are discretised. In addition, the number of intervals used to span the possible range of unit variable values is the same for all designs in each run. Runs with successively finer levels of discretisation are necessary in order to increase the likelihood of yielding the optimal solution. The optimal structure is only isolated if feasible areas are not rejected, either due to dependency or because infeasible and feasible values are part of the same interval.

The following chapter describes the development of an algorithm that can change the level of discretisation at each node of the search graph based on results from downstream. It addresses some limitations of the current method, allowing the optimal solution to be assured.

Chapter 6

An adaptive algorithm

6.1 Introduction

Work described in previous chapters has used intervals to bound the effects of uniform discretisation of continuous unit variables. This allowed the cost of the most promising process structures to be bounded. These bounds have been tight enough to discount certain structure, showing the potential of applying interval analysis to an implicit enumeration search. The intention was to ensure that the optimal solution had been isolated. This did not prove possible using the earlier methods for one main reason. While the previous work bounded the cost of the structures produced, it could not assure that a superior feasible structure had not been missed. A list of the *n-best* structures and their cost bounds is returned. This is based on the uniform discretisation profile specified. In addition, there is no flexibility to deal with cases where a particular alternative fails due to part of a unit variable being classed as infeasible. This could mean that not all areas of the search space are exhaustively examined causing the optimal solution structure to be missed.

The aim of the work presented in this chapter, is to isolate the globally optimal structure by changing the discretisation profile during the search. This would mean that the optimal structure could be found by applying the procedure once rather than making a series of runs with increasingly fine discretisation. This can be accomplished by no longer using uniformly sized intervals. Ideally the search should be concentrated in areas where it is necessary to use finer discretisation in order to find a solution. In addition, the method should ensure that all feasible variable values are searched exhaustively. If a design fails due to dependency or an interval being partially infeasible, the interval variable should be split until the problem is resolved.

The development of such an algorithm leads to using the concept of boxes (Moore, 1966) to describe variable values. An algorithm can then be developed based on a box splitting global optimisation algorithm. The generalised box splitting algorithm is adapted to work within the framework of the intervalised implicit enumeration approach described previously.

6.2 Boxes

A *box* used in interval techniques is defined by one or more variables each with a lower and upper bound value. The analogy with a real box allows splitting to be visualised more easily. When this idea is applied to a unit design in the synthesis problem, not all the dimensions of the *box* are continuous variables. For example, a distillation column may have three degrees of freedom. Two sides of a box describing a column design could represent the continuous variables of pressure and key recovery fraction. The third could be the discrete choice of light key component. Figure 6.1 illustrates two possible boxes for such a distillation design. The first box represents a unit design where component A

is the light key, the design pressure is an interval between 1 and 2 bar and the recovery fraction of A is an interval between 0.95 and 0.96. Box 2 represents a unit design under the same conditions apart from component B now being the light key component.

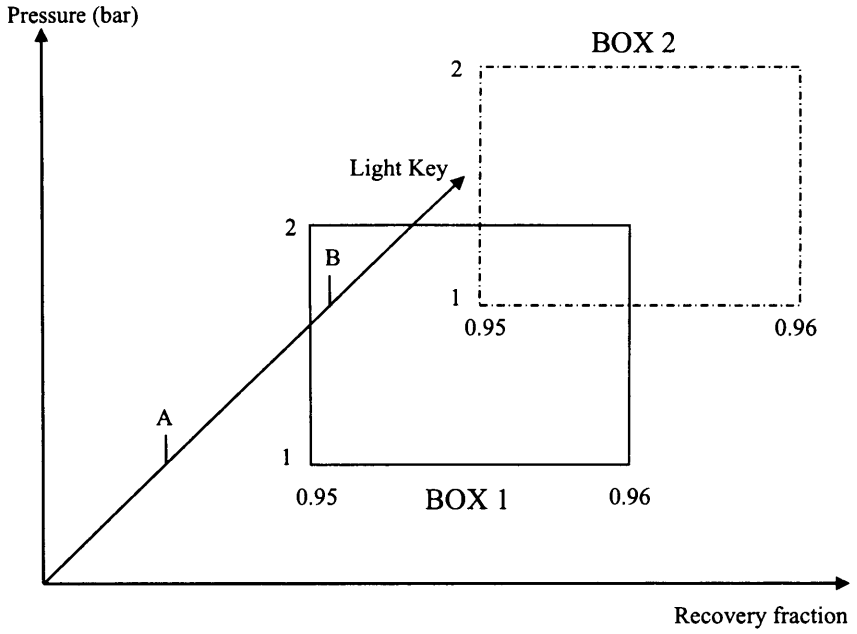

Figure 6.1: An example of two boxes

6.2.1 Splitting

A box may undergo a splitting operation on one of its continuous variables. In this work, the split occurs at the midpoint of the interval value of the variable, but in principal could be applied anywhere. Figures 6.2 and 6.3 show the two different splits that could be applied to a box representing interval values of two continuous variables. The original box represents variable Y with a lower bound a and an upper bound of b and variable x with a lower bound of c and and upper bound of d. In figure 6.2, the box is split on

variable x resulting in two new boxes with the bounds on x as shown. In figure 6.3 the box is split on variable y.

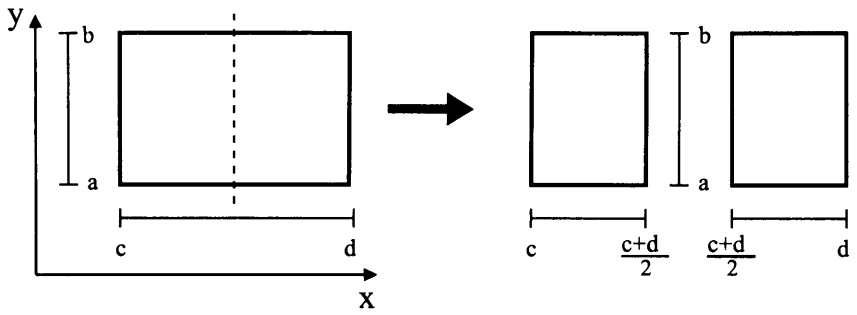

Figure 6.2: Box splitting procedure on the x variable

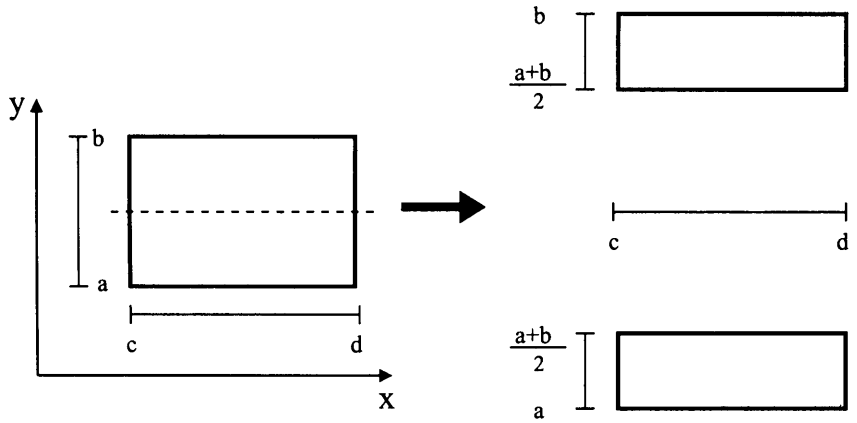

Figure 6.3: Box splitting procedure on the y variable

6.2.2 Other information associated with a box

As well as the information on the unit design variables, it is necessary to store information that allows boxes to be compared after the unit design has taken place. The following is a description of the additional data that is stored with each box.

Output streams

Each box that is produced by the splitting procedure holds bounds on the design variables for a particular unit. When the unit is designed based on these values, it may produce one or more output streams. Only a box that describes a product tank will not produce an output stream. The composition and state of these streams is stored along with the information that describes the box.

Output stream solutions

Once the output streams from a particular design have been identified, each is treated as a new problem stream and is solved. When the solution to a particular stream is isolated, information on the downstream structure and conditions is stored with the other box information. Once the solutions to all output streams from a box have been found then the structure that results from the box is known. This information can be used to compare two boxes during the search procedure.

Costs

If optimisation is to be carried out, it is necessary to attain the objective function value of each flowsheet. The objective function could contain capital cost, operating cost or another factor such as environmental cost. A box groups these interval costs into two types. The design costs are the costs of the particular unit design associated with a box. The solution costs are the costs of the unit design for this box plus the costs of solving the output streams from this unit design. An interval representing the objective function for a box and its downstream structure can then be calculated from these costs. Boxes can be pruned from the search based on the relative values of their objective function bounds.

Figure 6.4 illustrates the cycle of this information generation and storage. The design is based on the data stored by box 1, shown in figure 6.1, for a given feed stream. Firstly, the design of a distillation column is carried out based on the feed stream and design variables values of the box. This procedure gives the design costs of the box; in this case capital and operating costs. The unit design produces two output streams that are solved. This example shows that output 1 is solved by using another distillation column that yields two product streams. Output 2 is a product and requires no further processing. The costs and downstream structure associated with sub-problems 1 and 2 are retrieved and stored along with the data describing the box. To give the solution costs for this box, each of the design costs are added to the corresponding costs for sub-problem 1 and 2. From these total values, the objective function value range for this box is calculated. This may be simply the capital cost, the operating cost or some combination of the two.

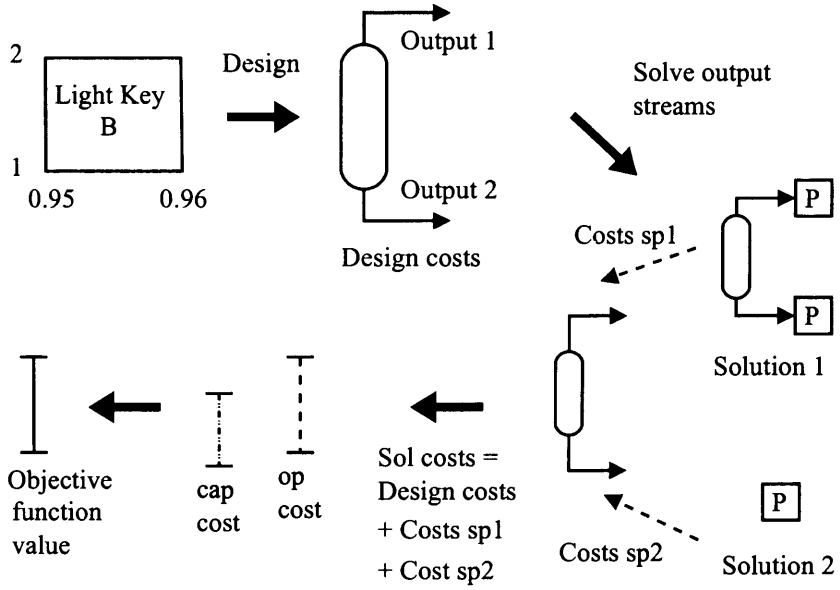

Figure 6.4: Building box costs and structure by unit design followed by solution of sub-problems

f

6.3 Changes required to standard box splitting algorithms

Various algorithms based on splitting boxes and generating bounds based on inclusion function have been suggested by Skelboe (1974), Ratschek and Rokne (1988), Hansen (1992), Kearfott (1996). These have many features in common and can be generalised by the algorithm shown in figure 6.5 (Csendes, 2001). It is based on finding the global minimum value, f^*, of a function, $f(x)$, on a search region covered by the box, X. If an inclusion function $F(X)$ can be found for $f(x)$ over the search range then the following general algorithm is valid.

---

1. Let L be an empty list of pairs of values. A is an initial box that covers the search region, X. Iteration counter $k = 1$. Set the upper bound on the global minimum, $\bar{f}$ as the upper bound of the inclusion function over the initial search region, $\bar{F}(X)$.
2. Divide A into s subsets $A_i, (i = 1, ...s)$. Evaluate the inclusion function, $F(X)$ for each of the new subintervals. Update $\bar{f}$ based on these function evaluations.
3. Let $L = L \cup \{(A_i, F(A_i))\}$
4. Remove members of L that cannot contain the global minimum point.
5. Choose a new $A \in L$ and remove it and its related function evaluation, $F(A)$, from L.
6. While termination criteria do not hold, $k = k + 1$. Go to step 2.

Figure 6.5: A generalised box splitting algorithm for global optimisation (Csendes, 2001)

The stopping criteria are likely to be based on a minimum width of the dimensions of A and $F(A)$. There are several schemes for the choice of the next box in step 5. The

most common method is to select the box that has the lowest value of lower bound for $F(X)$. The aim of this step being to select the box that is most likely to contain the global minimum point. Step 4 is where acceleration techniques are applied that help to speed up the convergence of the algorithm. For example, in an unconstrained problem where a box X is feasible throughout and the gradient, $g(X)$ is calculated. If $0 \notin g(X)$ then the box cannot contain a minima and it may be removed from consideration.

For this type of algorithm to be applicable for integration with the interval techniques described previously, alterations must be made to take into account the special properties of the synthesis problem. For example, the above algorithm assumes that the objective function is explicitly available for evaluation so that an inclusion function may then be obtained. This is not the case, as in implicit enumeration a graph is generated and searched at the same time, so that when a particular box is applied to a stream the downstream structure is unknown.

6.3.1 Bounding the global minimum

Box splitting algorithms require there to be an inclusion on the objective function. An inclusion function F on a real function, f is such that the application of $F(X)$ yields an interval that contains all possible values of $f(x)$ for $x \in X$. For this type of recursive search, the inclusion function would incorporate the cost of designing the unit described by the box plus the cost bounds of processing the resulting sub-streams for all possible unit variable values. The cost would include all possible downstream process structures. This would be inefficient as splitting would only occur on boxes associated with the initial feed stream. After every application of a box to this feed stream, the full range of variables for each unit would need to be applied to each of the subsequent streams in the process structure.

This work is intended for application to the early stages of process design. At this early stage, the main goal is to isolate the most suitable process structure from what is often a large number of possibilities. As a result, the aim of the algorithm is to isolate the optimal process structure. Unit design parameters are variables in the search, but a stream is said to be *solved* when the search has been narrowed down to one structural alternative for its processing. The values of the unit variables will have an effect on the downstream structure, but the goal is not to find their optimal values for each unit. Under these circumstances, the user can be sure that there is no set of variable values for another process structure that lead to a lower objective function value.

Given that the procedure is a search for structure then it would be preferable to bound the global minimum value rather than all possible values. A strategy that would allow splitting to occur at all levels of the search would be to recursively apply a box splitting algorithm for all streams encountered. For each stream encountered the procedure would isolate the optimal processing structure before passing control to the previous level of recursion. The optimal structure is assured at each level as the flowsheet is built from the downstream prodcuts upwards. The bounds that are calculated for a given box are made up of the cost of the associated unit design plus the cost of processing the output streams. The bounds on a box bound the minimum cost of processing the current stream given the operating unit and variable values represented by the box. For a given stream, the lowest value of the objective function and the lowest feasible set of variable values would be lower and upper bounds respectively on the global minimum for processing this stream.

Figure 6.6 shows a representation of this procedure. Starting in the top left hand corner, box X is to be applied to a process stream. This yields a unit design with cost D and two output streams, A and B. These and any subsequent streams are solved with a global optimisation algorithm. In the figure there are four possible downstream structures for stream A. Application of the box splitting algorithm results in structure A3 being accepted. The

same procedure selects downstream structure B2 for the processing of output B. The costs of optimal downstream processing are added to the cost of the unit design to give the total cost for this box. The bounds of this cost are bounds on the globally optimal value for this particular box. If the processing costs returned were instead the union of the cost of all the downstream processing possibilities then the bounds on the box would be an inclusion of all possible values. But, since the search is for the optimal processing structure then this is not necessary. As long as the optimal structure is isolated for each stream encountered the optimal overall flowsheet can be isolated.

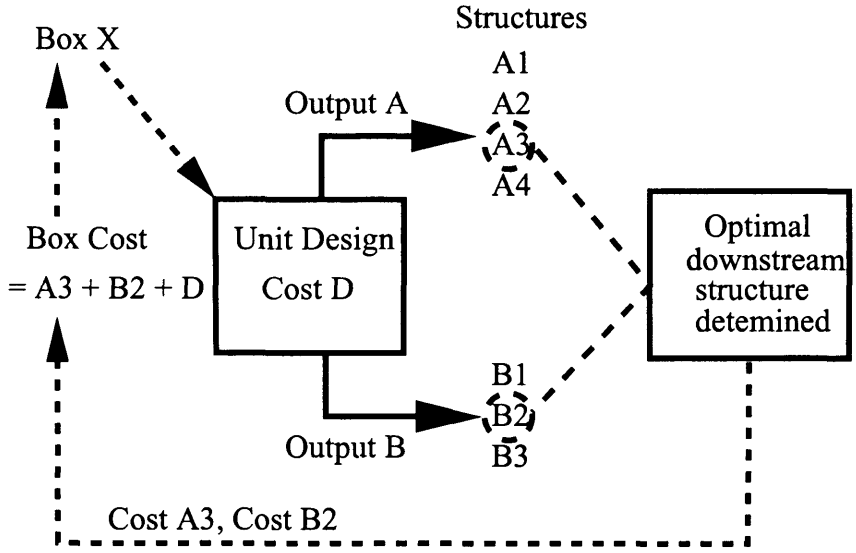

Figure 6.6: Constructing an inclusion function by successive optimisation of sub-streams

6.3.2 Updating the upper bound on the minimum

Many commonly used acceleration methods of step 4 cannot be applied to the synthesis algorithm as they require explicit knowledge of a term for the objective function. Other

methods require that the whole box will lead to feasible solutions which is not necessarily the case. One common acceleration method is the midpoint test where the real version of the function, $f(x)$ is evaluated at the centre of a box. If this point is found to be feasible then the value of $f(x)$ can be used to update the upper bound on the global minimum, $\bar{f}$. In the synthesis procedure this is the equivalent to evaluation at the *nominal* value of the box, a concept that has has been explained previously. After a box is applied to a stream, the algorithm is applied to the output streams of the resulting unit design. In addition to the bounds on the cost of processing the stream with this box, a cost of design and processing at the *nominal* value is also returned. If the box and downstream boxes have led to successful unit designs and products then this *nominal* value corresponds to a real and attainable cost for the flowsheet from this stream on-wards. As a result, a *nominal* cost of a box can be used to update $\bar{f}$ if it is less than the current value of $\bar{f}$. If for a box, A_i, $\underline{F}(A_i) > \bar{f}$ then this box can be eliminated.

6.3.3 Stopping criteria

For the generalised algorithm, the aim is to isolate the set of real variable values that result in the smallest possible value of the objective function. In practice this means that the box representing the variables should have dimensions smaller than a certain tolerance. At this level the user would essentially think of this box as a real point. For a box, A, the stopping criteria would therefore be $w(A) < \epsilon_1$. The other requirement is that the box represents points that are sufficiently close to the global minimum. For the box with the smallest value of $\underline{F}(X)$, if $\bar{F}(X) - \underline{F}(X) < \epsilon_2$ then all the points represented by the box are within ϵ_2 of the global minimum.

However, the search in the synthesis problem is not for a tight optimal box which closely approximates a point, but for a box with the dimensions necessary to ensure that the

optimal structure has been identified. This means that the stopping criteria do not need to be as tight. Therefore, it is possible to stop searching on a stream when all the remaining boxes represent the same process structure.

Another stopping criterion may be used if the boxes are stored in order of lower bound of the objective function and the box with the lowest lower bound is ranked first. If the lower bound of the second ranked box is greater than the upper bound of the top ranked box then the structure represented by the top ranked box is assured to be the optimal method of processing this stream. In practice it would be unexpected for this criterion to be used as it is likely that boxes that represent the same downstream structure will have more similar cost bounds than those with different structures.

6.3.4 Design Failure

In the methods described in chapters 4 and 5 the failure of a unit design resulted in that particular alternative or box being discounted in the search procedure. If the design is definitely infeasible then this action is appropriate. For example, a distillation column design that specified the heaviest component in the feed as the light key should be rejected. If the whole of the calculated reflux ratio is less than zero then the box should be discounted from the search procedure. However, this would not be desirable if only part of the reflux ratio is lower than zero. This result may suggest that part of the intervals that led to the design represent feasible designs and part do not. Alternatively it may be that the effect of dependency has caused bounds to widen and a design parameter interval to contain unrealistic values. In fact it is likely that both effects will occur and design failures of this type are a combination of the two factors. Whatever the reason, a mechanism is required that does not remove boxes that lead to design failures of this type. This is dealt with by assigning such a box with a cost interval with bounds ranging between $-\infty$ and $+\infty$.

These boxes will be ranked at the top of the storage list and will be selected for division first. In this way the area of the search space will be divided until a clear distinction between infeasible and feasible boxes is established.

6.4 An adaptive algorithm

The aim of the algorithm described in the following sections is to isolate the optimal process structure during the course of one run by changing the variable discretisation profile for each box that arises in the search. Previous work requires multiple runs to isolate a structure on the basis of cost bounds and these methods cannot ensure that a superior feasible structure has not been omitted. The box splitting algorithm described achieves this by searching in a depth first manner. Each problem stream encountered is solved to optimality before returning to the previous level of recursion in the search. This ensures that no areas of the search are omitted and that if a structure is returned, it is optimal.

6.4.1 Initial enumeration stage

Figure 6.7 shows the implicit enumeration algorithm of Jacaranda adapted to include an interval box splitting algorithm. For each new stream encountered in the search, boxes are stored in a newly initialised binary tree. It is necessary to store boxes in order of lowest lower bound and cycle through these in order and a binary tree carries out these operations efficiently. The algorithm is presented with a problem, p, that is a process stream with associated interval properties such as pressure or component flows. The binary tree is initialised and a boolean value, done, is set to false. As in the original

Jacaranda procedure, the algorithm checks whether or not this problem stream has been encountered previously. If so, no further action is required on this problem.

```
GIVEN: a list of available units, a list of product specifications, a range of values for each
unit variable, maxSplits = the maximum number of splits per box.
function solve(problem p)
  boolean done = false
  initialise empty binary tree, t, to store boxes
  upper bound on minimum = infinity
  if p already processed then
    done = true
  else
    for each available unit do
      for each discrete alternative do
        Create a new box, b
        process box(b,t)
      end for
    end for
    splitCount = 0
    while done = false and splitCount < maxSplits do
      if canStop(binary tree t) then
        store solution globally
        done = true
      else
        retrieve box with the lowest lower bound from t
        split along the longest side
        process resulting boxes
        splitCount++
      end if
    end while
  end if
end function solve
```

Figure 6.7: The main adaptive separations synthesis algorithm

---

If this is a new sub-problem, then the algorithm cycles through an initial set of alternative boxes. This is the same procedure employed in chapters 4 and 5. Each of the boxes produced by this initial enumeration is then processed in the procedure outlined in figure 6.8. A unit design is carried out based on the variable values of the box and the feed stream. If this design is successful then the cost of the unit design alone is compared with the current upper bound for each of the boxes stored. The cost of the stored boxes includes the cost of any downstream units. If the lower bound of the unit design is greater than any of these upper bounds, the box can be discarded. If the box is not discarded by this test, the output streams created are processed recursively by the solve procedure. The bounds on the box are then checked against the current upper bound before storage.

If the design fails, what happens depends on the type of design failure. If the design is definitely infeasible based on the criteria discussed in section 6.3.4, this box is removed from the search. If part of the box could lead to feasible solutions, this box is stored with cost bounds of $\pm\infty$. This will cause this box to be ranked top, or near there if other boxes have the same bounds, leading to it being selected preferentially for splitting. Hence in the splitting stage of the procedure, boxes like this will be split repeatedly until different areas are found to be infeasible or to lead to unit designs.

6.4.2 Splitting stage

After the initial enumeration of alternatives, there is a list of boxes each with bounds on the value of the objective function. Some of these bounds may be infinite due to design failure at the current level or further downstream. Now the box splitting stage of the solve

```
function process box(box b, binary tree t)
  design unit based on b
  if design successful and goodEnough then
    for each product do
      create a new problem p
      solve(p)
      retrieve solution
      update b with cost and structure
    end for
    if b is good enough then
      update upper bound on minimum
      store b in t
    end if
  else if dependency or partially infeasible failure then
    set infinite costs for b
    store b in t
  end if
end function process box(box b, binary tree t)
```

goodEnough compares the lower bound of the current unit design with the upper bound boxes stored in the binary tree, t. If the lower bound is greater than any of these upper bounds then the box can be discarded.

Figure 6.8: Box processing algorithm

procedure begins. Firstly, the criteria for stopping the solve procedure for this stream are tested. The stopping criteria checking procedure is shown in figure 6.9. If the box leads to a valid product tank design, then true is returned as a leaf node has been reached. Otherwise, the upper cost bound of the top ranked box is compared with the lower bound of the second ranked box. If the former is smaller than the latter then the procedure can stop as the *best* box has been found and hence the optimal downstream structure has been isolated.

```
function canStop(binary tree t)
  if any box stored within t yields a product tank then
    return true
  else
    u = upper cost bound of best box in t
    l = lower cost bound of 2nd best box in t
    if u<l then
      return true
    else
      while t has next box, b do
        if lower cost bound of b > lowest upper bound of all boxes in t then
          remove b from t
        else if structure of b is different from best box then
          return false
        end if
      end while
    end if
  end if
  return true
end function canStop(binary tree t)
```

Figure 6.9: Stopping criteria check

If it is not possible to stop based on the bounds of the top two boxes then the list of boxes is enumerated. If possible, a box is pruned based on its lower cost bound. If the box cannot be removed then its associated downstream structure is compared with that of the best box. The **canStop** function returns false if there is any diversity in the downstream structures of the boxes in the list and true if no diversity is found.

If it is not possible to stop then the box is divided along its longest side and the two

resulting boxes are processed as described earlier. Calculation of the longest side is based on comparing the current width of the interval that describes the side, divided by the inital value for this variable. This box splitting and processing continues until a stopping criterion is met or the maximum number of splits is reached. At this point, control returns to the level of recursion above.

6.4.3 Product requirements

A separations synthesis problem has product quality constraints. When dealing with real number flows of components, it is relatively easy to check if the product constraints have been met. With interval flows, the product purity requirement may be fulfilled by part of the flow.

In this situation, the combination of upstream structure and some parts of the variable value intervals that created this stream yield a product stream. However, some values within the intervals used upstream may not yield a product at this point without further processing. This issue is dealt with by realising that the search is for the optimal structure and not for the accompanying operating conditions. The structure to be isolated should give a lower value of the objective function than any other possible structure. That is, all other possible structures cannot attain such a low objective function value whatever combination of unit variable values is used.

As explained earlier, the nominal values of component flows within the stream are real values that represent feasible flows since they have not been mapped at any point. If the set of nominal flows within the stream meet a product constraint then the interval stream can be accepted as a product. This is because it is assured that part of the component flow intervals of the stream represent a product, therefore the upstream structure is capable of

generating a product without further processing. This means that the bounds on the cost of the upstream structure can be used to compare with other structures. It is possible that for some combinations of real unit variable values used within this structure, there will be a need for further processing in order to yield a product. Nevertheless, it is assured that at least one combination of real variable values in the upstream structure lead to a product without the need for further processing. This can be achieved within the cost bounds of the structure up to this point. For these reasons it is unnecessary to continue searching on this stream. A superior solution cannot be found by further processing as this will simply add to the cost of structure.

If the nominal value does not meet a product specification while part of the stream component flow intervals do meet this specification, the stream cannot be accepted as a product. However, this stream should not be processed further as it is possible that the upstream structure could produce a product at this point and the optimal structure could be missed. In this situation, control is returned to the previous level of recursion where the box that led to this stream is split along a variable that determines the stream composition. The boxes that result from this split are less likely to yield streams that cause the same problem.

6.4.4 Limiting the number of splits

It is necessary to limit the number of splits that are made in the attempt to solve a given process stream. The reason for this is that the width of variable values upstream in the process structure affect the width of the intervals, such as component flowrates, that describe subsequent streams. If the variable intervals are too wide upstream, the result may be that it is not possible to solve the downstream sub-problems even if degenerate intervals are used. This makes it necessary to have a mechanism in place that allows upstream

splitting to prevent the procedure becoming stuck in areas where a solution cannot be found. When the maximum number of splits is reached, control passes to the problem at the previous level. The box that resulted in the stream that was not solved has its bounds set to $\pm\infty$ which causes it to be selected for further splitting as boxes are selected for this on the basis of the lowest lower bound. This could also occur as a result that a product requirement is partially met by a stream as explained in section 6.4.3. In either case the stream contains intervals that are too wide to allow a downstream solution to be isolated.

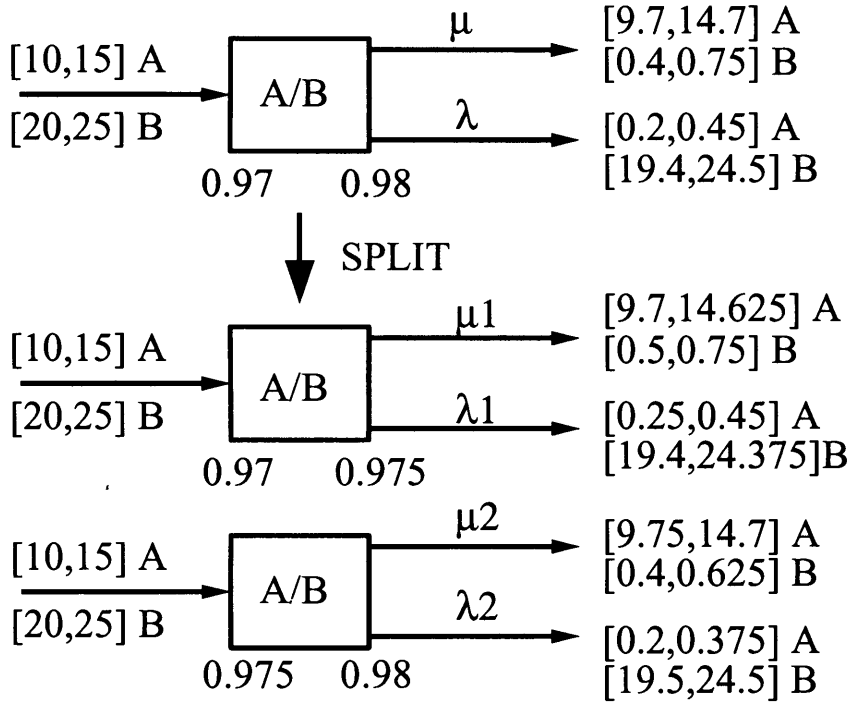

Figure 6.10: The effects of splitting on output streams

Figure 6.10 shows the effect on the flows as a result of splitting the key recovery variable in a box that represents a distillation column. This split would be necessary if either stream μ or λ was not solved within the maximum box splits or if one of them partially

met a product requirement. The split results in two boxes and the designs based on these boxes produce the streams $\mu 1, \lambda 1, \mu 2$ and $\lambda 2$. It can seen that the union of $\mu 1$ and $\mu 2$ is equal to the stream μ. In the same way $\lambda 1 \cup \lambda 2 = \lambda$. These new streams are more likely to be solved within the maximum iterations and are less likely to cross a product quality requirement. No stream possibilities are discounted from the search until it can be shown on the basis of bounding information that they lead to a non optimal structure.

6.5 Summary

An algorithm has been developed that is able to isolate the optimal process structure for a separation sequence. It is based on a generalised box splitting algorithm that has been adapted to fit into an implicit enumeration algorithm for process synthesis. The box concept used in these algorithms normally simply represents a set of lower and upper bounds on variable values. This has been enhanced to allow a box to describe the unit design for a process stream and store the resulting cost and downstream structural information.

The procedure applies the box splitting algorithm recursively on each stream encountered. In this way, the optimal structure is built from the leaf nodes upwards ensuring that the global minimum is bounded by the resulting structure. The effective inclusion function is around the minimum value of the objective function for the optimal downstream structure rather than including all possible downstream structures.

Boxes are pruned by continually checking the lower bound of each box against the upper bound of the global minimum. This upper bound is determined from the real nominal values of unit variables and stream flows. The nominal cost for a unit design and downstream structure corresponds to an attainable value as real values have passed continuously from

unit to unit. Consequently, it can be used to update the upper bound on the minimum cost for a particular stream.

The stopping criteria are determined by the fact that it is the optimal process structure that is required. For each stream encountered the procedure stops when the boxes in the list all represent the same downstream process structure.

The algorithm exhaustively searches the design space so that boxes with interval unit design parameters, which partially fail feasibility constraints, are not rejected. The box in question is instead stored for further splitting until the areas of feasibility can be isolated. No part of the search space is discounted except for on the basis of bounds or infeasibility.

The algorithm is not designed for a specific type of separation equipment. The next chapter demonstrates its performance and flexibility by the application of two different types of separation synthesis case studies.

Chapter 7

Case studies using the adaptive algorithm

7.1 Introduction

This chapter shows the applicability of the adaptive algorithm, described in the previous chapter, to two different types of separation synthesis case studies. The first is the application of the algorithm to a case study that was previously attempted using uniform discretisation in chapter 5. The second applies the algorithm to a bio-processing case study that has been previously attempted using the implicit enumeration approach of Jacaranda (Steffens et al., 2000).

7.2 Return to the benzene recycle case study

The benzene recycle case study described by table 5.3 and section 5.3, page 82, was tackled using the adaptive algorithm. The same distillation column design procedures are used as in the earlier case study. The following sections describe the results obtained and the manner in which the search was conducted.

7.2.1 Results and discussion

Figure 7.1 shows the optimal structure returned by the adaptive algorithm. The letters A, B, C and D refer to the components benzene, chloro-benzene, di-chloro-benzene and tri-chloro-benzene respectively. Below each column the light and heavy key components are shown.

The algorithm has demonstrated, through comparing cost bounds of solutions, that using the three column structure will incur the least capital cost. Distillation, with benzene and chloro-benzene as the keys, is carried out in two columns. These two separations are coarser than the very fine split required from one column. This structure matches that identified by the uniform discretisation procedure in chapter 5. The separation of benzene in two stages may not have been immediately obvious to a designer and shows the potential of the procedure for generating novel process structure.

In an enumerative search of this type, the order in which the search is conducted depends on the order in which the different chemical components are added to the feed in the original feed stream definition. This is because the first component added to the feed is the first component that is tried as the light key in a distillation column design. The search then solves all the sub-streams of this design before trying the next component as a light

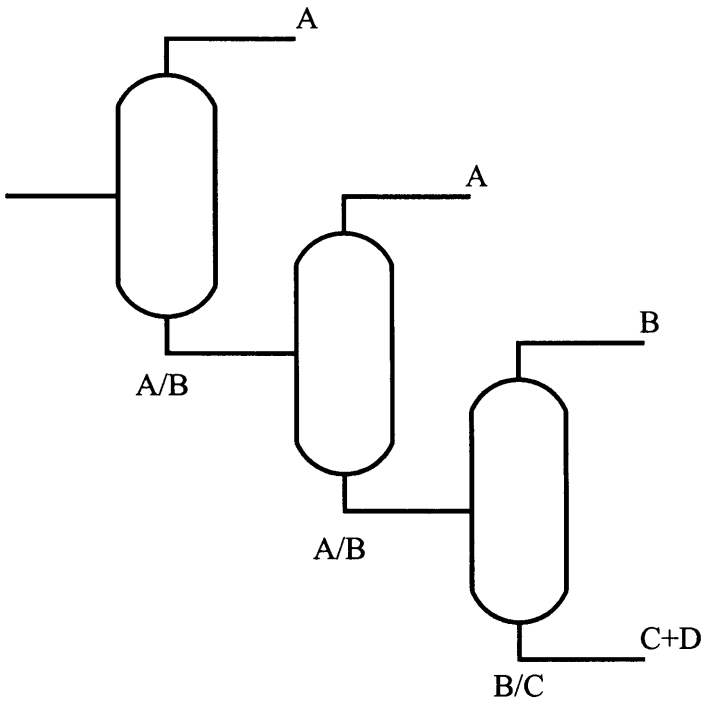

Figure 7.1: The optimal structure isolated by the algorithm for the benzene case study

key. Normally, the components are added in order of volatility for easier interpretation of the results by the user. Different orders of components were tried for this case study to change the path taken by the algorithm in the exploration of the search space. As expected, the structure returned is the same regardless of the way that the search is conducted. The algorithm guarantees that structures are not eliminated unless they are shown not to be optimal. It thoroughly investigates the entire search space, therefore if a structure is returned, it will be the same optimal structure whatever the search path taken. The path taken by the search can also be influenced by the way that the number of splits are limited for each sub-problem.

The following sections describe the manner in which the algorithm searches for the so-

lution to this case study. All processing times quoted were obtained using a Pentium III 850MHz computer running Java version 1.4.

Number of splits

As explained previously, it is necessary to limit the number of splits for each problem stream encountered. This prevents the algorithm becoming stuck on a stream that cannot be solved however many splits are carried out. The most straightforward way to do this is to have a fixed limit for every stream sub-problem encountered. Under these circumstances, control returns to the previous level of recursion where further splitting takes place. Naturally, the best value for the maximum allowed splits per stream will depend on the problem that is being tackled. For the benzene separation problem it was found that the lowest value that this limit could take was 4 splits. Below this number the algorithm is unable to return a solution because the low number of iterations does not allow sufficient resolution. The solution was obtained in 78 seconds with 2140 unique streams attempted and 25% re-use of solutions. If the order the components are listed in the input file is altered then the search proceeds in a different way. This is because the order of component chosen to be the light key changes. By reversing the order of components and using a maximum of four split per stream, the same solution was isolated. However, due to the different route taken, only 2018 streams were encountered and the procedure took 65 seconds.

Table 7.1 shows how changing the number of maximum splits allowed affects the way the search is performed. The larger the value for the maximum number of splits, the more problems are encountered and the longer the search takes. However, the rate of increase in time taken slows as the maximum number of splits allowed increases. This can be explained by examining the profile of where the splitting procedures take place.

Table 7.1: The effect of maximum split value on the search

Max no. of splits	No. of unique problems	Time (seconds)	%Reuse
4	2140	78	25
5	2240	82	26
6	2841	95	24
7	3194	100	24
8	3827	118	24
9	4531	139	24
10	5142	167	26
15	8088	288	39
20	9425	365	45
30	9498	378	48
40	9509	394	58
50	9572	438	63

During each run, the maximum number of splits was always reached for certain problems. When the maximum is reached, on some occasions a solution is found and on others it is not. Occasions where the maximum number of splits is reached without finding a solution support the idea that some streams cannot be solved however narrow the unit variable intervals become through splitting. In such cases, control returns to the previous level of the search and further splitting occurs at that level. Every time the solution of a stream is attempted, the number of splits required to accomplish this is recorded. Each time the maximum number of splits is reached without the stream being solved is also recorded.

Table 7.2 shows the distribution of numbers of splits required to solve the case study for various runs with different maximum split settings. For example, for a run where a maximum of 5 splits per sub-problem is allowed, 18 problems were solved requiring 2 box splits. 22 sub-problems were solved by making the maximum of 5 splits and 48 sub-problems were encountered where 5 splits took place but did not yield a solution.

In any run attempted, the largest number of splits where a solution was obtained was 7 splits per sub-problem encountered. For example, from the data shown in table 7.2, one column shows the maximum number of splits is set at 50. From this column it can be noted that 2 problems require 7 splits to find a solution. However, 72 problems are encountered where after 7 splits a solution is not found. Under these circumstances, it seems that solution cannot be found for these problems however many splits are made. The algorithm continues to split boxes until the maximum number is reached, in this case 50 splits, but doesn't find a solution.

Table 7.2: The number of occurrences of each split for various runs

	Maximum split setting								
No. of splits required	50	20	10	9	8	7	6	5	4
1	3002	2919	1286	1116	956	812	726	588	534
2	9	9	9	9	9	9	9	18	31
3	1	1	3	17	1	1	2	1	1
4	17	17	17	17	17	17	21	57	49
5	39	39	27	25	23	21	40	22	-
6	35	35	23	21	19	34	33	-	-
7	2	2	2	2	4	2	-	-	-
max(not solved)	72	72	48	44	40	36	40	48	77
Time(seconds)	438	365	167	139	118	100	95	82	78

For the runs where the maximum split were above 20 the splitting profiles are very similar. Maximum splits of 50 and 20 only differ in the number of streams where one split was made. It appears that the search in each case followed a similar path becoming stuck on streams and reaching the maximum 72 times. This would explain the slowing in the rate of increase of solution time as the maximum number of splits is increased, shown by table 7.1. The majority of the extra work required is for the 72 occasions when the box is split the maximum number of times. The total number of problems increases steadily

with increasing maximum splits but the proportion that have been encountered before also increases. This is useful because for other problems the best value to set for the maximum number of splits would be unknown. The results show that this parameter may be assigned a wide range of values while still yielding a solution.

Naturally, some streams require no box splitting in order to be solved. These are streams that meet one of the product specifications and are not shown in table 7.2. The table also shows that the majority of the time only one split occurred. It is the default that every box that is encountered undergoes one split in each unit variable. This may account for the large numbers of streams where only one box split occurs and removal of this mandatory single split could improve efficiency. In any case, since a single split always occurs the statistics presented over the following sections do not include such occasions and instead focus on when 2 or more splits occur.

Split depth

The number of splits that occur each time a problem is encountered yields some information on the way that the search progressed. However, alone it does not yield information regarding how each box was split. For further insight, information is required on the discretisation profile that results from the box splitting algorithm. A uniform profile would indicate that the algorithm was no more effective than the previously attempted uniform discretisation. The more uneven the distribution of splits, the more useful the algorithm is for increasing solution efficiency. For each stream encountered, a number of boxes result from the splitting procedure. Each of these boxes is descended from an original box that represents the full range of allowed variable values for a particular unit. The search may be very specific where splitting repeatedly occurs in the same area or uniform where splitting occurs evenly over the whole space. Figures 7.2 and 7.3 illustrate two different

ways that the same original box may be split to reach the same highlighted sub-box. In figure 7.2 the box is reached after 8 splits and in figure 7.3 the same box is reached after 4 splits. The former represents a uniform search and the latter as each in a specific area.

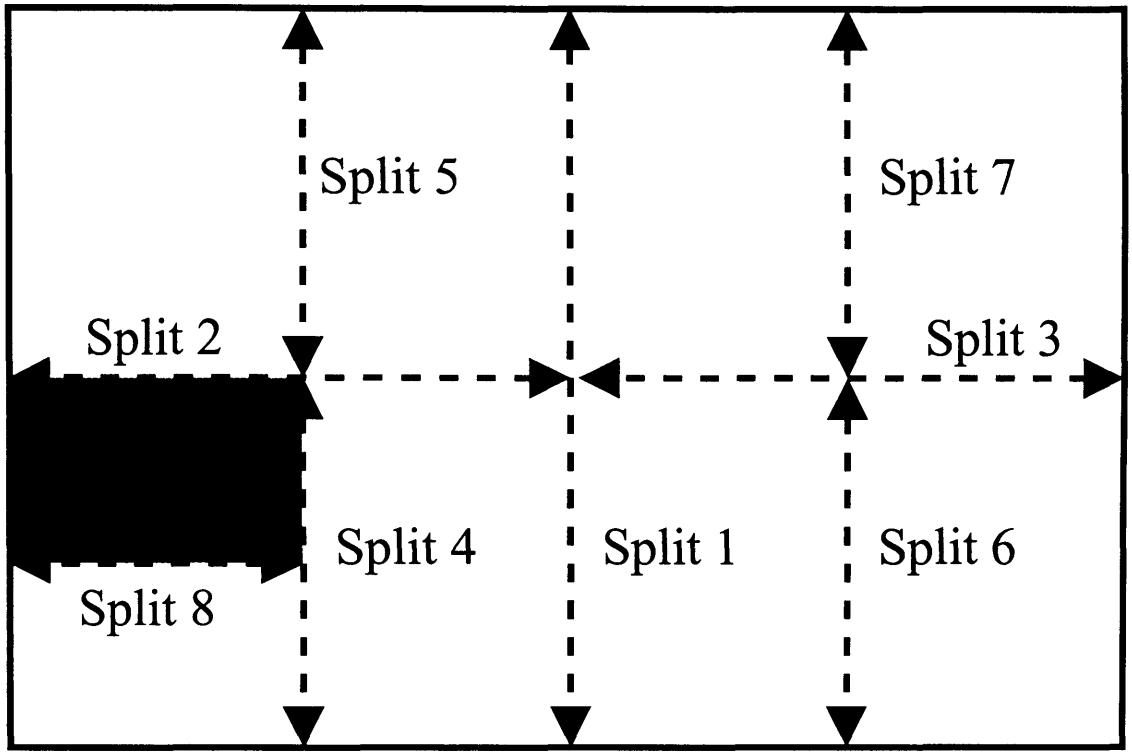

Figure 7.2: The maximum split depth in the figure is 4, the original box has been split 8 times

The term split depth will be used to describe how specific the search is for a particular problem stream. It is defined as the longest line of parent to child relationships in a given splitting scheme. For example in figure 7.3 the split depth is 4. This is because the shaded box results from a series of four splits where each split is carried out on a box resulting from the previous split. The split depth of the scheme shown in figure 7.2 is also 4. Even

though eight splits have taken place, the longest line of parent to child relationships occurs from split number 1, 2, 4 and 8. This is a total of 4 splits and hence the split depth is 4. For each stream encountered, a maximum split depth can be recorded. The larger the maximum split depth compared to the total number of splits the more concentrated the search to a particular area.

Figure 7.4 was generated from two runs, one with a maximum number of splits per stream of 4 and the other 20. Only data from problem streams that were solved are shown in the figure. The results show that the maximum split depth is often the same as the number

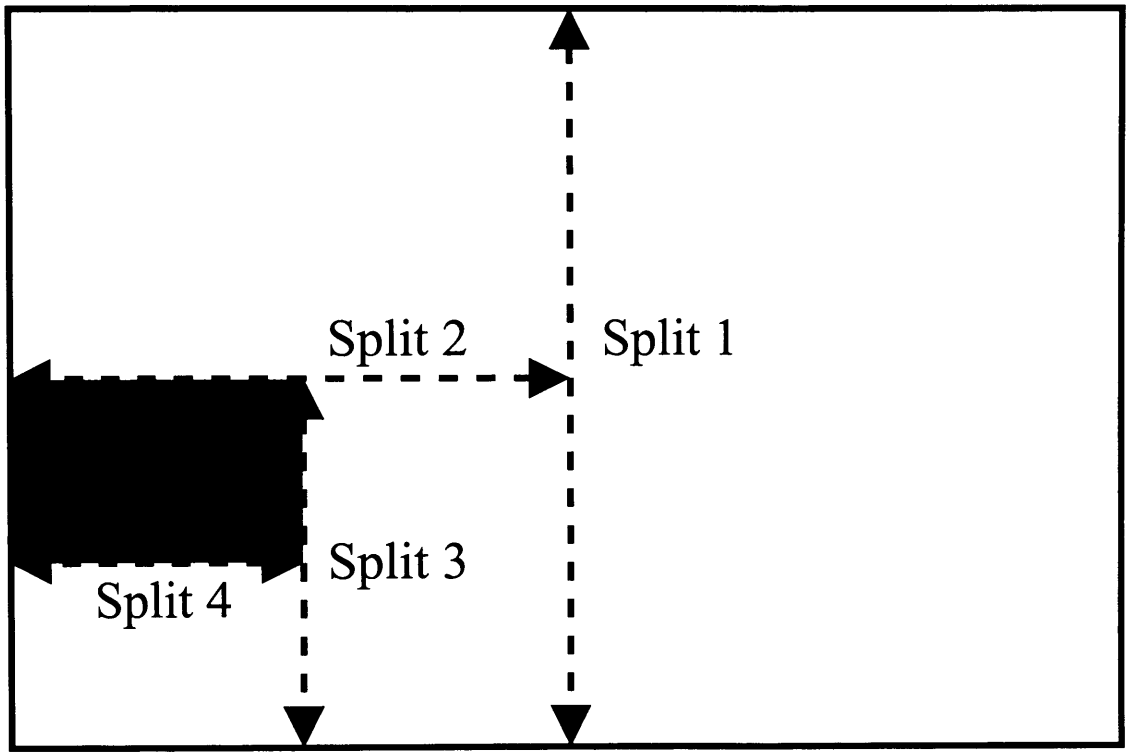

Figure 7.3: The maximum split depth in the figure is 4, the original box has been split 4 times

of splits or at most one less than it for both runs. This shows that specific areas of the search space are being concentrated upon. The algorithm adapts to search in the area where the solution is most likely to be found based on the bounding information from previous boxes. This leads to a more efficient search than a uniform discretisation. For the run where the maximum number of splits per stream was 4, the depth of the splitting is equivalent to 16 uniform splits. When the problem was attempted uniformly with 16 splits using the procedure described in chapter 5 it took 3151 seconds compared to 78 seconds when using the adaptive scheme.

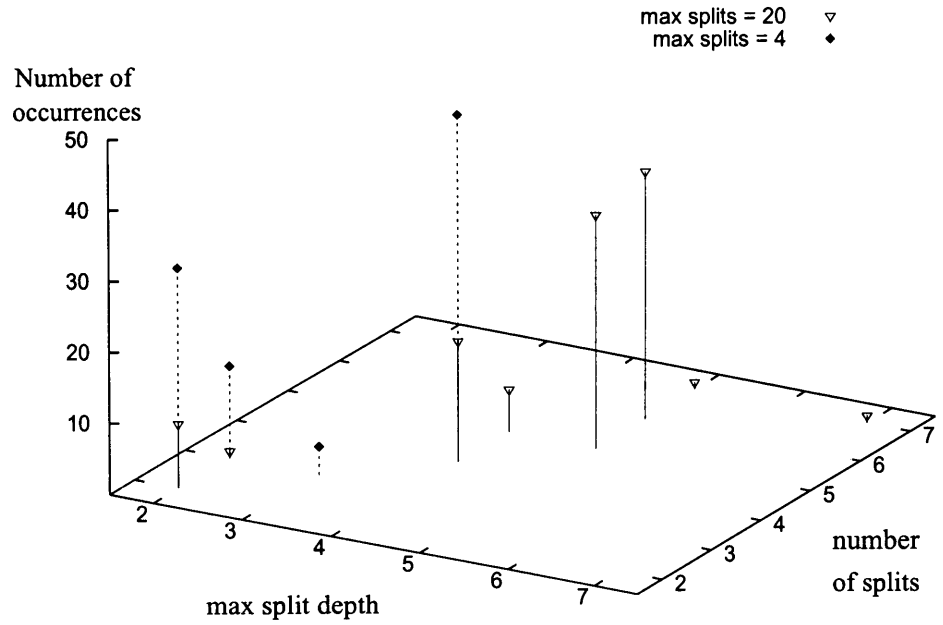

Figure 7.4: Comparing the distribution of split depth and number of splits for two splitting schemes

Stream characteristics and splitting

A number of characteristics of the streams encountered during the search were recorded. The following section discusses the effect, if any, of these characteristics on the number of splits required to solve the stream.

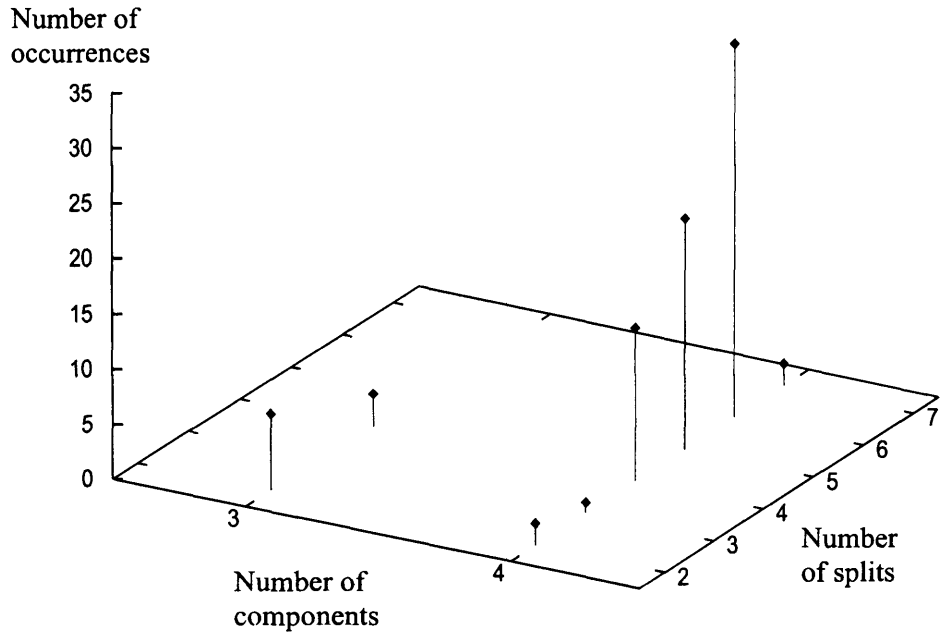

Figure 7.5: The number of splits carried out against the number of components present in the stream

Figure 7.5 shows the number of occurrences of each number of splits as a function of the number of components in the stream. The results shown are for a maximum of seven

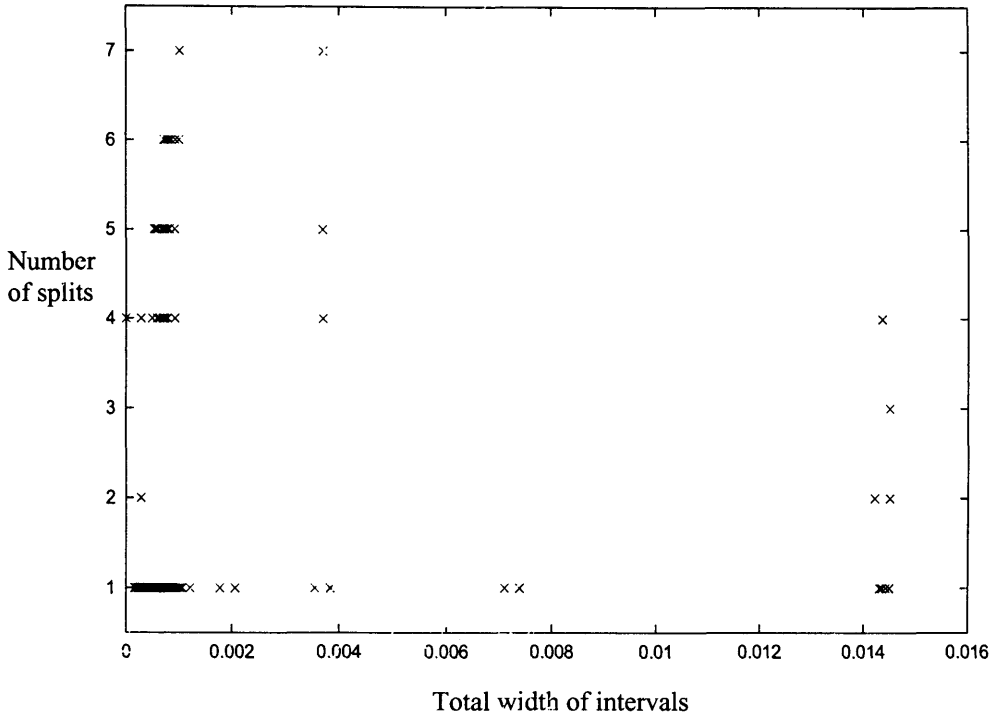

Figure 7.6: The number of splits carried out on streams against the width of interval flows

splits per stream. This is because for this case study, occasions where splitting occurs more than seven times never lead to a solution. The figure shows that streams containing four components are more likely to require larger numbers of splits than those containing three components.

Figure 7.6 shows the number of splits required plotted against the total width of the stream flow intervals in each problem stream. The data do not seem to show any trends based on this indicator. The final measure of stream properties was the total nominal flow of the stream. The number of splits is plotted against nominal flow in figure 7.7. For the purposes of resolution, five data points relating to streams with nominal flows between 0.98 and 1 kmol/s, were not included in the figure. Most of the streams that occur during

the search have much smaller nominal flow rates. Again, no trends in the data are evident.

These results suggest that there is no inherent predictability of how the search will proceed. No pattern is visible that would allow one to predict the discretisation profile required to solve a particular stream. This confirms the applicability of such an adaptive algorithm to the problem. The distributed nature of the data points in the previous figures shows that the procedure changes the splitting profile as required. This can be based both on bounding information and on the failure of previous attempts at solving the stream using wider intervals.

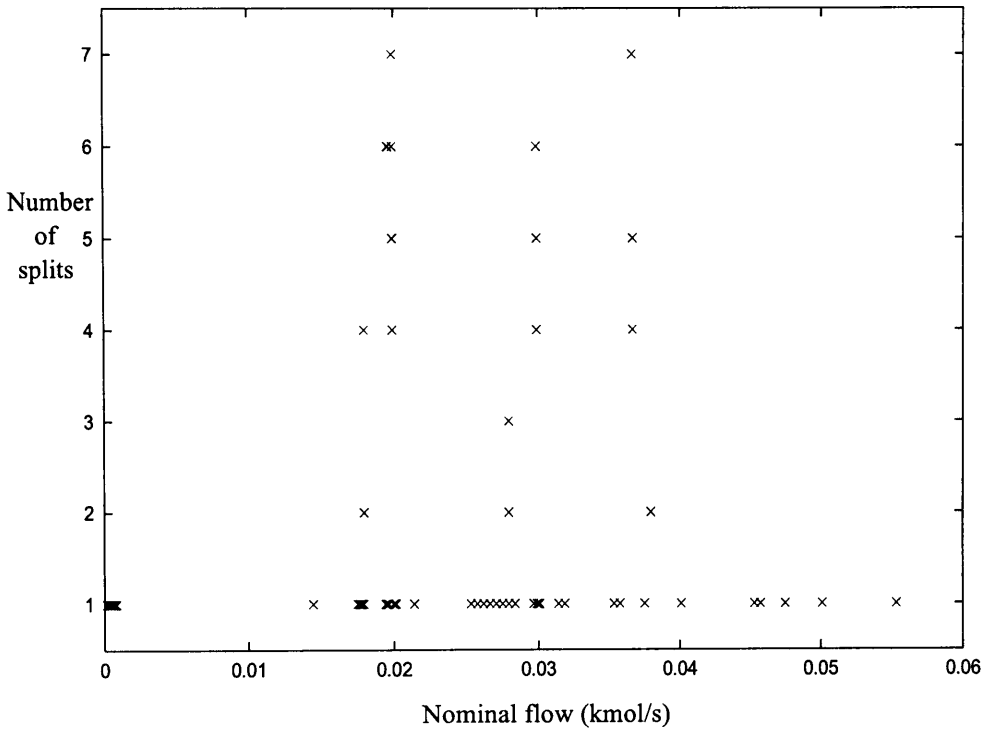

Figure 7.7: The number of splits carried out on streams against the nominal stream flow rate

7.2.2 Summary

This case study has demonstrated that the algorithm is able to isolate the optimal separation structure. As would be expected, the same structure is isolated whatever the route taken by the search procedure.

Investigation of the way that the box splitting occurs has shown that successive splits tend to occur in the same area of space. In effect, the intervals that span the search space are non-uniform and a fine level of uniform discretisation would be necessary for the same resolution. This demonstrates the effectiveness of applying an adaptive scheme to the problem. Plots of the number of splits against different stream characteristics show no pattern in the way that different types of stream are handled. This lack of predictability further supports the suitability of the algorithm to such problems.

7.3 Application to Bioprocess synthesis

Thus far, all the results of the application of interval analysis have been concerned with the synthesis of distillation sequences. The following sections describe the application of the new algorithm to bio-process separation synthesis. This is intended to demonstrate that the procedure is applicable to all types of separation synthesis problem.

The case study presented is based on previous work on bio-process synthesis described in Steffens et al. (2000). That work used Jacaranda with modified stream types and suitable biological separation units to synthesise a process for the purification of bovine somatotropin (BST). This is a hormone that increases milk production in cows. It is produced by the fermentation of a recombinant *E.coli*. The previous work discretised component flow rates and as expected the coarser this discretisation, the greater the reuse of solutions

and therefore the less time taken for the search. Using the adaptive algorithm it was not possible to attempt the full problem described in the paper as far less reuse of solutions occurs. This is partly due to the fact that discretisation was not used at all for the bio-process problem because of the necessity to account for very small flow rates of product and contaminants. In addition the problem is more combinatorial than those solved previously due to the many available units. This leads to a larger range of possible stream flowrate combinations than if just one type of processing unit is used. In order to compare the two approaches, a stream that occurs three unit operations downstream from the original fermenter product, in the optimal structure from the discretised run, is used as a feed stream for the adaptive procedure.

7.3.1 Bio-process streams

The constituents of streams encountered in biological processes tend to be very different from those found in chemical process streams. Due to the heat sensitive nature of proteins and other biological molecules, distillation is not considered as a separation technology. Instead the variation in physical properties, other than vapour pressure, is exploited in bio-separations. The data associated with bio-streams reflects this. The molecular weight, hydrophobicity, molecular size and density of each component is stored and the information may be recalled by the relevant units.

The BST from the fermentation is produced within inclusion bodies which themselves are present within the *E.coli* cells after the fermentation. At the stage of the flowsheet where synthesis is to be carried out in this case study, the cells have already been homogenised and the cell debris removed. The feed stream is shown in table 7.3. The physical properties of the components can be found in appendix table A.1. An inclusion body is a mass

of mis-folded proteins. Those present in the case study are made up of BST and a contaminating protein. The composition of the inclusion bodies is shown in appendix table A.2.

Table 7.3: Feed for the case study

Component	Flow rate (kg/hr)
Glucose	0.6
NH_4^+	0.2
SO_4^-	0.2
anti-foam	2
protein 1	5
protein 2	3.15
protein 3	2.05
protein 4	2.45
protein 5	2.15
protein 6	1.15
protein 7	3.4
protein 8	3
protein 9	3.3
protein 10	2.65
protein 11	1.75
protein 12	0.65
protein 13	0.35
inclusion body	12
water	400

The streams developed in the previous work have been adapted to integrate with the interval based procedure. Previously a real concentration for each component described the composition of the stream along with an overall volumetric flow rate. In the current work, each component has an interval mass flow. Due to the high level of purity required for the case study the *trace* interval has not been introduced to bio-streams. Instead an absolute component mass flow is used below which the component is assumed to no longer be

present. The value of $1 \times 10^{-8} kg/s$ was used as an effective zero flow as this was used as zero in previous case studies. The non-sharp nature of some of the bio-separation units means that unless such a limit is introduced components never *disappear* from streams and are seen to be present in ludicrously small amounts. Since such a limit is necessary, the value at which it is set would seem conservative.

7.3.2 Bio-processing units

Screening

The processing units use the same techniques for screening and unit design as in Steffens et al. (2000). Two types of initial screening are used in order to test whether or not a stream can be processed by a certain unit.

- Design constraints depend on the ability of a unit to process a stream. These may be heuristics, for example based on particle size, or the physical impossibility of a chromatography column processing solids. The constraints used are shown in appendix table A.3.
- The binary ratio (Jaksland et al., 1995) may be evaluated in order to assess the feasibility of using the unit for the required separation. Each unit takes advantage of a difference in a particular physical property between components in order to induce separation. The binary ratio for two components is the ratio of this physical property from one component to another. This is compared to the feasibility index, n, for the unit in question.

If the constraints and the binary ratio tests are passed then the unit design calculations are allowed to proceed. The unit models that are used in the case study can be broadly classified into two types.

Stream splitting units

These unit designs are based on the concept of key components similar to the distillation unit model described earlier. In the distillation model the feed is sorted in terms of volatility. In other stream splitting models the feed is sorted on the basis of whatever the difference in physical property that is being exploited for separation. Two adjacent components are then selected as *keys* and the design proceeds. Thus, for a particular stream and splitting unit a design occurs at most $ncomps - 1$ times where $ncomps$ is the number of components in the feed stream. The description of the procedure for the available stream splitting units is described below. In order to keep dependency to a minimum, the design calculations have been rearranged to their most simple form. This is compared to the models used in Steffens et al. (2000). If intervals had been substituted directly into the design, it would often lead to the same variable interacting with itself excessively. The following description of the stream splitting unit models reflects this. The parameters used in unit designs are shown in appendix table A.5 and the capital and operating cost data are shown in appendix table A.4.

Ultrafilter The components are ordered based on molecular size and large components are assumed to be completely impermeable and all pass into the reject stream. The concentration of all components in the reject stream, C_r is specified as a design parameter. The concentration of small components in the reject stream is the same as that in the feed so the concentration of large components in the reject stream can be calculated. Since the

mass of the large components is known the volumetric flow rate of the reject stream can be calculated. The known concentrations of the small components in the reject stream are then used to calculate the mass of each small component. The balance in the volumetric flow rate is made up of solvent, which is in this case water. A mass balance is then used to calculate the permeate composition. Membrane area is calculated by finding the limiting flux from a concentration-polarisation model (Ho and Sirkar, 1992). Gel concentration, c_g, and mass transfer coefficient, k, are assumed. Annual operating costs are composed of the energy cost and the cost of changing the membrane once each each year.

Microfilter Key components are selected in the same way as the ultrafilter. The design is similar to that of an ultrafilter, but with different ranges of values for screening and concentration of components in the reject stream. In order to calculate membrane area, cake resistance is assumed to be dominant and a concentration-polarisation model (Ho and Sirkar, 1992) is used to estimate the flux. Like the ultrafilter, operating cost consists of energy and membrane replacement.

Diafilter This is designed in the same way as the ultrafilter except that the dilution factor, D_F, is used as a design variable. This is the amount of solvent added in terms of number of multiples of the volume of the feed.

Rotary drum filter Again keys are selected on the basis of size. All the flow of large components passes into the reject stream. The mass fraction of the small components that pass through the filter cake is calculated from the wash rate, w . The wash rate is the volume of wash water per volume of slurry and is a design parameter (Kennedy and Cabral, 1993). The other design parameter is the concentration of solids in the filter cake.

The area is calculated using a constant cake resistance (Belter et al., 1988). Operating costs are composed of energy costs and the cost of the filter aid additive.

Other units

Chromatography units cannot be designed using key components. The columns use batch operations and take two separate fractions of the product of the column. A particular product is targeted, in this case it is the protein BST. The unit models for chromatography columns calculate the composition of each of the fractions based on an estimation of peak width and the difference in a particular physical property of the feed component (Leser et al., 1996). As with stream splitting units, design equations have been rearranged to minimise dependency. Gel-filtration chromatography exploits differences in Log_{10} of molecular weight and hydrophobic interaction chromatography differences in hydrophobicity as defined by Leser et al. (1996).

The Renaturing tank is used to first solubilise the inclusion body before refolding the product protein. It produces a single product stream. The residence time,T_R, and yield, y, are specified and the feed stream is diluted with water to reach a user defined protein concentration, c_p. Capital cost is based on those for a CSTR and the only operating cost is the cost of the chemical additive.

7.3.3 Results and discussion

As explained earlier the feed stream shown in table 7.3 has already been purified to some degree and is a stream generated by the discretised synthesis procedure described in Steffens et al. (2000). In the original work, most of the unit variables were set to one value,

except for the reject stream concentration in the microfilter unit model. The strength of the algorithm, described in this thesis, is the increased efficiency of a search that allows ranges of unit variable values. For this reason, three unit variable values are allowed a range of values in the case study.

- Reject concentration, c_r, in the ultrafilter: 50-400 g/l
- Reject concentration, c_r, in the microfilter: 400-1100 g/l
- Dilution factor, D_F, in the diafilter: 2-5 m^3/m^3

The results of runs using the adaptive algorithm can then be compared with the results of the discrete procedure that allows a number of discrete points within the range of each variable.

Two types of unit that terminate the search are used.

- A product tank accepts streams containing BST in a concentration of greater than 300 g/l. Impurities, apart from water, must be less than 0.1 % by mass.
- A waste tank accepts streams containing less than 0.1 kg/hr of BST. In the original work this parameter was set at 0.001 kg/hr, but the problem could not be solved using the new procedure using this value. The reason for this is that since component flows are no longer discretised, the flow of BST to a waste tank does not *disappear*. For the waste streams from certain unit operations, vital to the process, this constraint can never be met so no solution is found. Waste tanks have a set capital cost of $ 100000. A product value of 100 $/g is used in order to calculate a product penalty. This is the annual value of the product that goes to waste.

An objective function of annual operating cost plus capital cost amortised over 2.5 years plus product penalty is used in all runs. As discussed earlier, the adaptive method uses a minimum flow of $1 \times 10^{-8} kg/s$, below which a component is no longer considered to be present. If a similar level of discretisation is used for all components with a run of the discrete method, it is unable to reach a solution without running out of memory. For this reason, the finest level of discretisation used in the previous work of Steffens et al. (2000) will be used as a basis for comparison. The aim is to show the benefits of addressing the problem more realistically in terms of the resulting flowsheet structures.

Figure 7.8 shows the optimal structure that results from applying the adaptive algorithm and figure 7.9 shows the top ranked structure from the application of the discrete procedure. The number of discrete points for each variable in the discrete search was determined by the finest level reached during the adaptive search. The waste products from the units all contain some water, it is only specifically indicated when it is the only waste product. Search statistics are shown in table 7.4. The cost of the structure identified by the adaptive procedure is the nominal cost and therefore an upper bound on the cost of the structure. It does not include product penalty.

Table 7.4: Search statistics

	Discrete	Adaptive
Cost(US$)	1,592,405	1,333,231
Time(s)	82	5,770
Unique Problems	2,198	90,261
% reuse	48	14

The computation times quoted are from a 850MHz pentium III processor running Java 1.4. The statistics show that the adaptive algorithm encountered far more streams than the discrete method, which led to a solution time at the scale of hours as opposed to the scale of minutes. However, the extra rigour has reaped benefits in isolating a superior flowsheet

compared to the discrete method.

There is much less re-use of solutions by the adaptive compared to the discrete method. This is because without discretisation even a tiny difference between two streams will be enough for them to be treated as separate problems. In fact, the small proportion of occasions where problems occur more than once, might mean that it is better not to use dynamic programming for certain problems. The adaptive method was not able to deal with the size of the full problem because the dynamic programming table stores the solution of all problems encountered. When many problems are involved, the computer eventually runs out of memory. If the storage no longer occurred then the algorithm would be able to deal with far more problem streams.

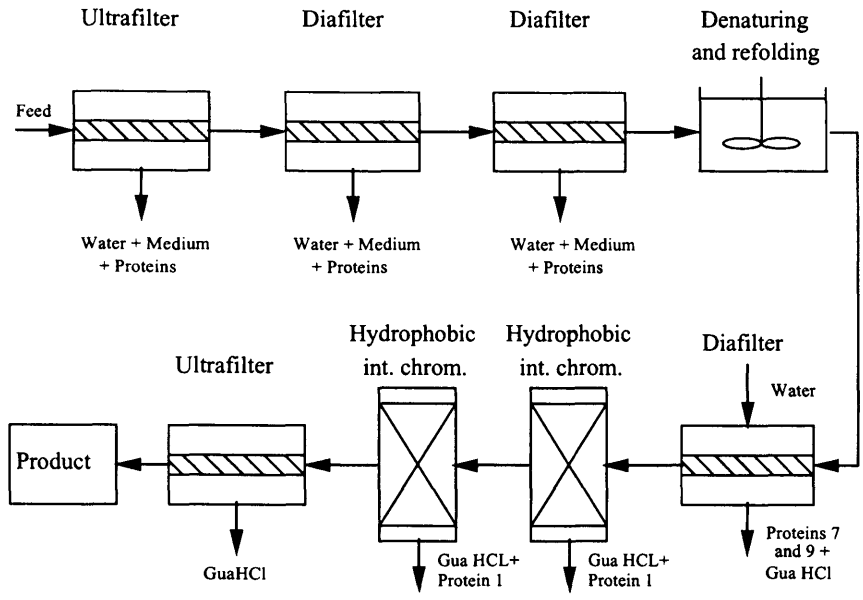

Figure 7.8: The optimal bio-separation structure as identified by the adaptive procedure with a cost of $1,333,231

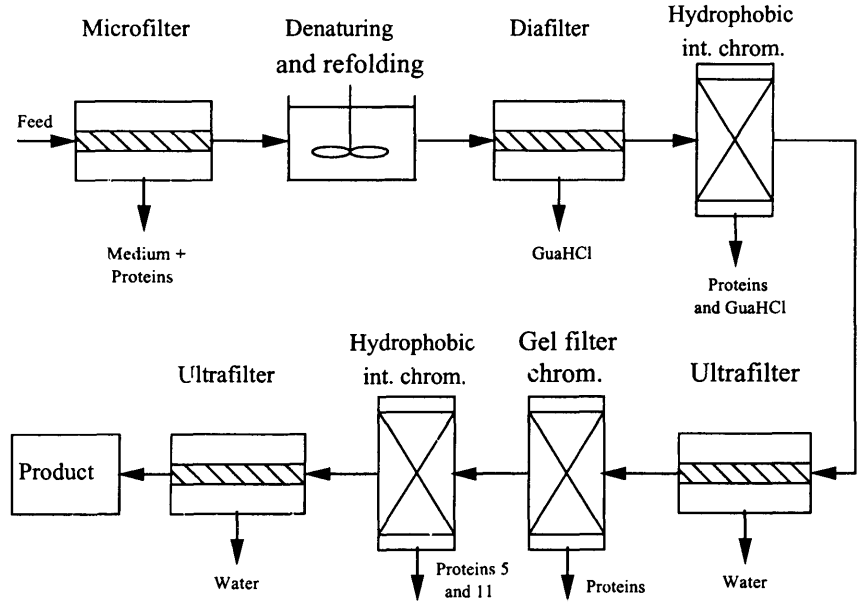

Figure 7.9: The top ranked bio-separation structure using the discrete method with a cost of $1,592,405

Product flow

The top ranked flowsheet from the discretised procedure does not invoke any product penalty meaning that there is no product whatsoever in any waste streams. This is due to the discretisation of the flow of BST. Analysis of the flows of BST through the two flowsheets reveals what is happening. Both have feed streams containing 6.14 kg/hr of BST. However, the flowsheet from the adaptive procedure has a flow of 4.72 kg/hr of BST into the product tank whereas the discretised flowsheet has a product flow of 4 kg/hr. The yield from the denaturing and refolding tank is 80% which accounts for 1.228 kg/hr of unrecovered product. This means that 0.91 kg/hr of BST are *lost* due to the discretisation procedure which is why no product penalties are incurred. The adaptive generated flowsheet seems to represent a more a realistic situation. To allow a fair comparision between the two methods, table 7.4 does not include the product penalty in the cost of the flowsheet generated by the adaptive algorithm.

Cost

Since component flows are not discretised away by the adaptive procedure, it might be expected that the streams would cost more to process. However, the adaptive procedure has isolated the structure shown in figure 7.8 as optimal. The upper bound on the cost of this structure is $1,333,231 which is approximately 15% cheaper than the top ranked structure from the discrete procedure. In this case discretisation has led to a superior structure being missed. This may be partly due to less BST being *lost* due to discretisation which leads to less processing required for the necessary product concentration.

The minimum cost of the optimal structure may be considerably less than the figure quoted above as this is merely the cheapest feasible set of units and variable values encountered during the search. The search algorithm has assured that this is the optimal structure but it is likely a different set of unit variable values will cost even less.

Box splitting

In terms of box splitting, the bio-processing case study does not require the same level of box splitting as the distillation case study. Table 7.5 shows the number of times each number of splits was necessary to solve a problem stream. Only occasions where the stream was not a product or waste stream are counted.

The vast majority of problems required only three splits for solution. This is because three units are available with a range of possible values for a unit variable. Each of these unit variables is always split at least once which means that the minimum number of splits for a non-terminating stream is three. Only just over 1% of streams require any further splitting compared with over 18% in the most efficient run of the distillation case study.

The finest fraction of the total range of a unit variable split by the search is 0.25 which is only a split depth (defined in section 7.2.1) of 2. In comparison, the largest split depth in the distillation case study was 7 which corresponds to a fraction of the total range of 7.8×10^{-3}. The reason for this difference is that are many more discrete choices in the bio-problem due to the additional choice various unit operations. The only discrete decisions in the distillation case study were the choices of key component in each column. Since there are so many available unit types in the bio-problem, there are far more possibilities for each stream encountered. The nature of these discrete decisions seems to have a greater effect on the objective function than the values of unit variables. Thus, the algorithm is able to reject many sub-optimal structures, as the structure has more of a significant effect than the unit variable values used.

Although the adaptive box splitting is not used as much in the bio-problem, it is still used to a certain extent. The advantage of employing the procedure can still be appreciated by considering the extra processing required to carry out the maximum number of splits shown in table 7.5. The maximum is required 13 times in the adaptive search. A relatively

Table 7.5: Split statistics for the bio case-study

Number of splits	Occurrences
3	40,306
4	401
5	84
6	62
7	13

huge amount more processor time would be required if the same number of splits were to be carried out on almost 41,000 streams. This would be necessary if the uniformly discretised interval procedure was used as in chapter 5.

7.3.4 Summary

This section has demonstrated that the new algorithm can be applied to a completely different type of process engineering problem showing its generality. The run for using the previous procedure presented in this chapter uses the same level of unit variable discretisation but the coarseness of the component discretisation makes it far less realistic. The results show that the increased rigour of bounding the solutions and removal of component flow discretisation is worthwhile as it leads to a significantly better solution.

The adaptive algorithm takes much longer to find a solution to the bio-problem than the discretised method but this is not surprising due to the differences in the two runs. Component discretisation is used to a large degree in the discrete run. Since many of the splitters in this problem are non-sharp, components do not *disappear* as readily when using the adaptive method compared to the discrete method. This causes there to be more unique streams in the adaptive procedure and leads to far less solution re-use. The statistics on splitting show that on relatively few occasions is there more than one split per unit variable. However, this is an indication of the efficiency of the algorithm. Specific areas

are targeted for finer discretisation while the vast majority of streams do not require this treatment.

The top-ranked solution from the discrete method does not show any BST in the waste tanks so incurs no product penalty. Analysis of the flowsheet shows that nearly 1kg/hr out of 6kg/hr is lost during discretisation. This does not occur using the adaptive method, which confirms a more realistic model of a real process. The flowsheet generated by the adaptive procedure yields a greater flow of BST as a product because of the lack of discretisation. This may be the reason for the lower objective function value.

Chapter 8

Conclusion

This thesis has addressed the problem of finding the optimal flowsheet structure for a separation process given a feedstock, product specifications and a set of units. The basis of the technique used is implicit enumeration with streams and units represented within an object oriented framework.

A drawback of the existing procedure was the need to discretise unit variables and stream characteristics such as component flows. The user had no idea of the effect of these discretisations on the quality of the solutions obtained. Indeed there was no guarantee that the top-ranked solution was in fact optimal. Interval analysis has been applied to stream and unit models in order to address these issues.

8.1 Intervals for cost bounding

Chapter 4 showed how interval techniques can be applied to the unit variable pressure in a distillation column. Intervals were used to span the spaces between discrete pressure levels. The resulting design of a column gave bounds on the capital and operating costs.

Ultimately this allowed the cost of an entire flowsheet to be bounded during the implicit enumeration search. This technique was applied to a hydrocarbon separation problem where a range of pressure was allowed in each column. The goal was to isolate the optimal distillation column sequence.

Several runs were carried out at varying levels of discretisation. The solutions were ranked on the basis of the lowest lower bound on the objective function. This ensured that there was no cheaper available solution to the separation problem. In the course of this investigation, it was realised that if the column pressure intervals were consistent with the stream pressure intervals, the nominal cost would represent a real solution. This allowed the nominal cost to be used as an upper bound on the global minimum cost for the problem. This result proved very useful when trying to resolve the superiority of one process structure over another. A structure is deemed as sub-optimal if its lower bound is greater than the best nominal value in the list of solutions. This allowed resolution between structures with a coarser level of discretisation than would have otherwise been possible.

Chapter 5 developed the application of interval analysis further by using intervals to describe component flows. This is a more challenging issue than that tackled in the previous chapter as the presence (or absence) of a component within a stream depends on its flow rate. Whether or not a component is present can in turn affect the units necessary for processing and therefore the resulting structure. The method proposed to deal with these issues, no longer discretises the majority of component flows as this leads to unnecessary widening of bounds. Discretisation only occurs for very small flows where the flow is mapped to the *trace* interval. This allows much solution re-use to be retained, as compared to no discretisation whatsoever, while providing a threshold where a component is no longer considered as a key for distillation.

8.2 The adaptive algorithm

Chapters 4 and 5 described a framework for bounding the cost effects of discretisation, giving some assurance on the quality of the solutions obtained. Successive runs provided progressively tighter bounds on these solutions. However, there were outstanding problems associated with using intervals such as when an interval parameter was part feasible and part infeasible. A scheme that could deal with these situations and change the level of unit variable discretisation depending upon necessity was described in chapter 6.

This algorithm signalled a departure from the straightforward implicit enumeration search. The new algorithm is based on previous box splitting algorithms but is novel in the combination with an implicit enumeration search. Each discrete alternative generated by the enumeration generates a box. Each stream encountered is then solved recursively, not to find the best box but to find the optimal structure for processing the stream. No area of the search space is discounted before it has been demonstrated that it is either sub-optimal or infeasible. This gives the assurance that the structure isolated is optimal.

The application of the new algorithm to the benzene case study in chapter 7 demonstrated its performance. The same case study was attempted in chapter 5 with uniformly distributed unit variable intervals. To reach the same resolution as the adaptive method the previous method required 40 times the processor time.

The number of splits required for different streams encountered was analysed. No obvious relationship was evident between the number of splits required to solve the problem and any of the stream characteristics. This unpredictability shows the need to use such an algorithm, as rules cannot be applied as to how a stream should be handled simply based on its constituents.

The ability of the algorithm to handle a different type of problem was demonstrated by its application to a case study involving the purification of a protein from a biological

process stream. In solving the problem the algorithm does not require the quantity or fineness of box splitting that was required by the distillation problem. However, this still indicates the efficiency of the procedure. It is better that further splitting only occurs on a small minority of occasions when necessary rather than to this fine level for every stream encountered.

The value of the interval bounding was demonstrated in the fact that the structure that was identified as optimal was different from the top ranked structure yielded by the previous discrete approach. The discretisations in unit variables and component flows had led to a superior solution being missed.

Overall, the methods developed have been successful in addressing the issues that motivated this project. Interval cost bounds give the user a feeling for range of likely costs for the process. The adaptive algorithm allows the optimal structure to be identified. This is achieved in a more efficient manner than if the whole search space were spanned by uniformly sized intervals at the necessary level of detail as used by the adaptive algorithm.

8.3 Future work

It has been demonstrated that interval analysis and box splitting can be applied to an implicit enumeration based search for optimal separation process structures. However efficient the algorithm, there are always a finite number of alternatives that must be explored in the search. The algorithm works most efficiently when there are fewer discrete options, such as choice of different units, and more unit variables with ranges of possible values. For a given unit design, the use of interval calculations takes longer and more memory is required to store solutions. For this reason, the new algorithm is more likely to fail for highly combinatorial problems than the previous discrete approach.

The issue of memory use could be addressed by removing the dynamic programming facility for certain problems. To tackle the bio case-study, discretisation, even at small flow rates, was not used. As a result there was little re-use of solutions. The savings in processor time are outweighed by the huge amount of memory required to save all the solutions. Without dynamic programming the search could continue for more than the timescale of hours and therefore would be capable of tackling larger problems.

In terms of processing time, it may be possible to increase efficiency by investigating alternative criteria for choosing the box and variable for the next split. For a given problem stream, this would only be useful when all the boxes in the list had been assigned a downstream structure and the aim was to eliminate sub-optimal structures.

It has been demonstrated that the interval based algorithm is applicable to different types of synthesis problem. Currently, using the algorithm for a new type of problem requires a lot of development time. This is particularly true for the conversion of unit models for interval calculations. In the future, it would be useful to develop a thorough strategy for minimising dependency in the perturbed interval equations encountered in unit models. This could be extended to the calculation of stream characteristics. Such a strategy would allow more rapid application to new problems and eventually to an interface for writing interval models.

Appendix A

Biological data

A.1 Physical properties

Tables A.1 shows the physical properties of the components of the feed stream to the case study presented in section 7.3.

Table A.1: Physical properties for the case study

Component	Density (g/l)	Size μm	Molecular Weight (g/mol)	Φ
glucose	1250	1.0×10^{-3}	180	0.1
NH_4^+	1050	5.0×10^{-4}	18	0.01
SO_4^-	1050	1.0×10^{-3}	96	0.01
antifoam	985	1×10^{-2}	500	1.0
BST (product)	1000	0.02	248200	0.9
protein 1	1000	0.01	18370	0.71
protein 2	1000	0.015	85570	0.48
protein 3	1000	0.013	53660	0.76
protein 4	1000	0.013	120000	1.5
protein 5	1000	0.013	203000	0.36
protein 6	1000	0.013	69380	0.36
protein 7	1000	0.013	48320	0.48
protein 8	1000	0.013	93380	0.93
protein 9	1000	0.013	69380	0.01
protein 10	1000	0.013	114450	0.63
protein 11	1000	0.013	198000	0.06
protein 12	1000	0.013	30400	1.0
protein 13	1000	0.018	94670	0.01
water	1000	4.0×10^{-4}	18	N/A

Table A.2: Inclusion body properties

Component	concentration (g/l)
BST	650
protein 1	620
Density	1270 g/l
Size	0.4 μm

A.2 Unit data

The following tables show the design constraints, parameters and costing information that were used by Steffens et al. (2000) and for the bio-synthesis case study presented in section 7.3.

Table A.3: Bio-process unit design constraints

Unit	Constraints
Ultrafilter	$1\mu m > x > 0.001\ \mu m$
	$c_f < 200$ g/l
	$c_r < 500$ g/l
Microfilter	$10\ \mu m > x > 0.1\ \mu m$
	$c_{rs} < 500$ g/l
Diafilter	$1 > x > 0.001\ \mu m$
	$c_f < 400$ g/l
Rotary drum filter	$200\ \mu m > x > 1\ \mu m$
	$c_{solidf} < 70\%$ w/w
Chromatography column	$c_f < 70$ g/l
	no solids in feed

Table A.4: Capital and operating costs

Unit	Capital cost	Operating cost
Ultrafilter	1000 \$/m^2	Membrane = 250 \$/m^2
		Energy = 5 kWh/m^3 permeate
		Energy costs = 0.04 \$/kWh
Microfilter	5000 \$/m^2	Membrane = 700 \$/m^2
		Energy = 0.13 \$/m^3 feed
Diafilter	1000 \$/m^2	Membrane = 250 \$/m^2
		Energy = 5 kWh/m^3 permeate
		Energy costs = 0.04 \$/kWh
Rotary-drum filter	9528 \$/m^2	Filter aid = 5kg/m^3 filtrate
	+ 22787 \$	Filter aid cost = 0.33 \$/kg
		Energy = 0.12 \$/m^3 feed
Gel filtration	273613 \$/m	Gel cost = 300 \$/l
	+ 82894 \$	
Hydrophobic-interaction chromatography	247490 \$/m diameter	Gel cost = 400 \$/l
	+ 88765 \$	
Solubilisation and renaturing tank	$1000V^{0.53402}e^{5.348}$ \$	2.15 \$/kg guanidine

The $\Longleftarrow$ symbol in table A.5 denotes variables that have been given a range of values for the purposes of the case study.

Table A.5: Unit design parameters

Unit	Parameter values	
Ultrafilter	c_g = 250 g/l	
	k = 1.0×10^{-6} $m^3/m^2/s$	
	c_r = 50-400 g/l	$\Longleftarrow$
Microfilter	c_k = 400 g/l	
	k = 1.5×10^{-5} $m^3/m^2/s$	
	c_r = 400 − 1100 g/l	$\Longleftarrow$
Diafilter	c_g = 250 g/l	
	k = 1.0×10^{-6} $m^3/m^2/s$	
	D_F = 2.0 − 5.0 m^3/m^3	$\Longleftarrow$
Rotary-drum filter	c_k = 350 g/l	
	w = 1.0 m^3 water/m^3 feed	
	Cake resistance = 5.0×10^{11} m/kg	
	Pressure drop = 68 Pa	
	Filtrate viscosity = 0.0011 kg/m/s	
	Cycle time = 180 s	
Gel-filtration	Maximum diameter = 1.0 m	
	B_{sam} = 5 %	
	Column length, l = 0.25 m	
	σ = 0.46	
	δ = 0.02	
	t_r = 1 h	
Hydrophobic interaction chromotography	Maximum diameter = 1.0 m	
	B_c = 20 mg/ml	
	Column length = 0.25 m	
	σ = 0.22	
	δ = 0.02	
	t_r = 1 h	
Solubilisation and renaturing tank	t_r = 44 h	
	y = 80 %	
	c_{prot} = 50 g/l	
	Solution of 3M guanidine HCl used	

A.3 Unit design procedures

A.3.1 Ultrafilter and microfilter

Stream composition

The following equaitions relate to the calculation of the composition of permeate and reject stream based on the design variable c_r (the total concentration of the reject stream).

The subscripts r,p and f relate to reject, permeate and feed streams respectively. The subscripts l and s refer to large and small components respectively.

$$c_{Lr} = c_r - c_{Sf}$$

The concentration factor is the ratio of the concentration of large components in the reject to the feed.

$$vcf = \frac{c_{Lr}}{c_{Lf}}$$

Small components have equal concentration in the product streams, this is the same concentration as in the feed. For small component i:

$$c_{if} = c_{ir} = c_{ip}$$

The total concentration of large components in the reject stream is then calculated.

$$c_{ir} = \frac{c_{if} c_{Lr}}{c_{Lf}}$$

The flowrate of the reject stream is then calculated from the concentration factor.

$$Q_r = \frac{Q_f}{vcf}$$

Volume factors for the small components in the feed and the large components in the reject are then calculated. s relates to the integer index of the components in the feed when ordered by particle size starting with the largest component having an index of 0. n is the number of components in the feed stream.

$$v_{fracSf} = \sum_{i=s}^{n} \frac{c_{if}}{\rho_i}$$

$$v_{fracLr} = \sum_{i=0}^{s-1} \frac{c_i r}{\rho_i}$$

The concentration of the solvent (water) in the reject and permeate streams can then be calculated from these volume factors.

$$c_{Wr} = \rho_W(1 - v_{fracLr} - v_{fracSf})$$

$$c_{wp} = \rho_W(1 - v_{fracSf})$$

Finally the volumetric flowrate fo the permeate is calculated.

$$Q_p = \frac{Q_f\rho_f - Q_r\rho_r}{\rho_p}$$

Costing of ultrafilter

The area, A, of the filter is calculated from the permeate flowrate, mass transfer coefficient, feed concetration and gel concentration.

$$A = \frac{Q_p}{k\ln\frac{c_g}{c_f}}$$

$$\text{capcost} = 1000\left(\frac{388}{325}\right)A \text{ (Ho and Sirkar, 1992)}$$

Energy costs, in USD/year, are then calculated from permeate volumetric flowrate and hours of operation per year.

$$E = 5Q_p \times 0.04 \times 3600\text{hpy}$$

Total operating cost is then calculated from the membrane cost, Cost_{mem} (USD/m^{2}yr). This was 250 \$/m^{2}yr in the case study. opercost = $\text{Cost}_{mem}A + E$

Costing of microfilter

The area, A, of the filter is calculated from the permeate flowrate, mass transfer coefficient, feed concetration and cake concentration.

$$A = \frac{Q_p}{k \ln \frac{c_k}{c_f}}$$

$$\text{capcost} = 5000A$$

$$\text{opercost} = \text{Cost}_{mem} A + 0.13 Q_f \times 3600 \times \text{hpy}$$

A.3.2 Diafilter

Stream composition

The same naming conventions are used for the dialfilter as the other two filters. It produces a reject stream and a permeate stream and the split is based on component size when those in the feed are sorted in order of size. For this unit, the volumetric flowrate of the reject stream is the same as that of the feed stream and the large compoent pass completely into the reject stream so $Q_f = Q_r$ and for a large component i, $c_{if} = c_{ir}$.

The design variable D_F is the ratio of volumetric flowrate of the permeate stream to the feed stream so $Q_p = D_F Q_F$. The concentration of a small component, i, in the reject is given by the equation $c_{ir} = c_{if} e^{-D_F}$.

The volume factors of large components in the feed and small components in the reject are then calculated.

$$v_{fracSr} = \sum_{i=s}^{n} \frac{c_{ir}}{\rho_i}$$

$$v_{fracLf} = \sum_{i=0}^{s-1} \frac{c_{if}}{\rho_i}$$

The concentration of water in the reject stream can then be calculated.

$$c_{Wr} = \rho_W(1 - v_{fracSr} - v_{fracLf})$$

The concentration of a small component, i, in the permeate stream is given by:

$$c_{ip} = \frac{c_{if}Q_f - c_{ir}Q_r}{Q_p}$$

Finally the concentration of water in the permeate is calculated.

$$c_{Wp} = \rho_w(1 - v_{fracSp})$$

Costing

The area of the diafilter is calculated using the same equation as the ultrafilter. The capital cost is also calculated in the same way. Operating is then calculated as follows:

$$\text{opercost} = \text{Cost}_{mem}A + 0.4\text{capcost}$$

A.3.3 Rotary drum filter

Stream composition

The density of small components in the feed are calculated from their concentration and volume fraction.

$$\rho_{Sf} = \frac{c_{Sf}}{v_{fracSf}}$$

The volume fraction of large components in the feed includes the contribution of the filter aid that is added.

$$v_{fracLf} = \sum_{i=0}^{s-1} \frac{c_{if}}{\rho_i} + \frac{c_{filterAidFeed}}{\rho_{filterAid}}$$

The concentration of filter aid in the reject stream is then calculated.

$$c_{filterAidReject} = \left(\frac{c_{filterAidFeed}}{c_{Lf}} \right) c_k$$

The fraction of the small components that are left in the filter cake, x_{Sk}, is a function of the design parameter wash rate.

$$x_{Sk} = 1.092\mathrm{e}^{-1.1945 x_{Sk}}$$

The concentration of water and small components in the reject is then calculated.

$$c_{Wr} = \left(\frac{1 - c_k/\rho_L}{1/\rho_W + \left(\frac{x_{Sk} c_{Sf}}{c_{Wf} \rho_{Sf}} \right)} \right)$$

$$c_{Sr} = c_{Wr} \left(\frac{x_{Sk} c_{Sf}}{c_{Wf}} \right)$$

Large components pass entirely into the reject stream in the same proportions as they are present in the feed. For a large component, i:

$$c_{ir} = \frac{c_{if} c_k}{c_{Lf}}$$

The properties of the permeate stream can then be calculated using a another mass balance. Small components pass into the reject stream in the same proportions as they appear in the feed.

Costing

Filter area is calculated from permeate volumetric flowrate, Q_p, filtrate viscosity, μ, cake resistance α, pressure drop ΔP, feed solids concentration, c_{solidf}, cycle time t_c and stream volumetric flowrates.

$$A = \frac{Q_p}{\sqrt{\frac{0.4\Delta P}{\mu \alpha c_{solidf} \frac{Q_f}{Q_p} t_c}}}$$

$$\text{capcost} = (9528A + 22787)\frac{388}{325}$$

The operating cost includes the cost of filter aid, $cost_{fa}$.

$$\text{opercost} = (3600 \times cost_{fa} \times \text{hpy}) + (0.12 Q_f \times 3600 \times \text{hpy})$$

A.3.4 gel filtration

Stream composition

In the following equations the subscript, P, refers to the product component and p refers to the product stream. The volumetric flowrates of the reject and product streams are equal.

$$c_{Pp} = c_{Pf} * (1 - \delta)$$

The concentration of component i in the output streams is calculated as follows. All logs are to base 10.

$$k_P = \log M_{wP}$$

$$k_i = \log M_{wi}$$

$$\Delta = k_P - k_i$$

If $\Delta \geq \frac{\sigma}{2}$ and $\Delta < \sigma$ then $x = (1 + \delta)\frac{(\sigma - \Delta)^2}{\sigma^2}$
If $\Delta \geq 0$ and $\Delta < \frac{\sigma}{2}$ then $x = (1 + \delta)\frac{(\sigma^2 - \Delta^2)}{\sigma^2}$

$$c_{ip} = x c_{if}$$

Costing

The volume and diameter are then calculated.

$$V = 100 Q_f t_r B_{sam}$$

$$d = \sqrt{\frac{4V}{\pi l}}$$

capcost = $273613.0d + 82894$

opercost = $3600\text{hpy}Q_f$

Water is added to the output streams in quantities that make the stream densities that of water.

Hydrophobic interaction columns are deigned in the same way as gel filtration columns except that Δ is calculated from the difference in hydrophobicity of the product component and component i. The volume of the column is given by

$$V = 100\frac{Q_f t_r}{B_c}$$

A.3.5 Solubilisation and renaturing tank

This unit is used to dissolve inclusion bodies and access the protein that is contained within.

Stream composition

The volumetric flowrate of the output stream,Q_p, is calculated from the design variable C_{protp}, the concentration of protein in the product stream, the concentration of protein in the feed stream, C_{protf}, the concentration of inclusion bodies in the feed, C_{ibf} and the volumetric flowrate of the feed, Q_f.

$$Q_p = (C_{protf} + C_{ibf})\frac{Q_f}{C_{protp}}$$

Components that are not within the inclusion bodies in the feed are then added to the product stream in their entirety. Proteins within the inclusion body are added to the product stream. The mass added is the mass in the feed multiplied by the yield variable, y.

The refolding chemical is then added to the product in the concentrations specified. In the case of the case study this was guadinine in a concentration of 300 g/l. Water is then added to make the volumetric flowrate of the stream up to the value of Q_f calculated earlier.

Costing

The volume of the tank is calculated from the volumetric flowrate of the feed multiplied by the residence time. The minimum volume is 0.4 m^3.

$$V = Q_f t_r$$

$$\text{capcost} = 1000 V^{0.53402} e^{5.348}$$

Operating cost is calculated from the cost of guadinine, $cost_{guad}$,the mass added and the number of operating days per year.

$$\text{opercost} = 3600\text{hpy} Q_f C_{guad} cost_{guad}$$

A.4 Biological nomenclature

B_c	binding capacity (mg/ml)
B_{sam}	sample volume (% column vol)
c_f	total feed conc (g/l)
c_{solidf}	feed solids conc (g/l)
c_g	gel conc (g/l)
c_k	cake solids conc (g/l)
c_p	total permeate stream conc (g/l)
c_{prot}	protein concentration (g/l)
c_r	total reject stream conc (g/l)
c_{rs}	reject solids conc (g/l)
D_F	dilution factor (m^3 water/m^3 feed)
hpy	hours operation per year (500 for the case study)
k	mass transfer coefficient ($m^3/m^2/s$)
M_{wi}	Molecular weight of component i
t_c	cycle time
t_r	residence time (h)
w	wash ratio (m^3 water/m^3 feed)
y	yield (g protein renatured/g total protein)
α	cake resistance (m/kg)
μ	filtrate viscosity (kg/m/s)
Φ	hydrophobicity defined by Leser et al. (1996)

Bibliography

A. Aggarwal and C. A. Floudas. Synthesis of general distillation sequences - nonsharp separations. *Computers chem. engng.*, 14:631–653, 1990.

E. Bek-Pedersen and R. Gani. Design and synthesis of distillation systems using driving force based approach. *Chem. Eng. Process.*, 43:251–262, 2004.

P. A. Belter, E. L. Cussler, and W.-S. Hu. *Bioseparations: Downstream processing for biotechnology*. John Wiley & Sons, New York, 1988.

R. J. Best, N. S. Dhalu, and W. R. Johns. Enumerative and AI methods for process design. *Chem. and Ind.*, 15:510–515, 1987.

R. P. Byrne and I. D. L. Bogle. Global optimization of modular flowsheets. *Ind. Eng. Chem. Res.*, 39:4296–4301, 2000.

O. Caprani and K. Madsen. Mean value forms in interval analysis. *Computing*, 25:147–154, 1980.

T. Csendes. New subinterval selection criteria for interval global optimization. *Journal of Global Optimization*, 19:307–327, 2001.

M. M. Daichendt and I. E. Grossmann. Integration of hierarchical decomposition and mathematical programming for the synthesis of process flowsheets. *Computers chem. engng.*, 22:147–175, 1998.

S. Djouad, P. Floquet, S. Domenech, and L. Pibouleau. *Fuzzy information engineering*, chapter 37, pages 619–631. John Wiley and sons, 1997.

J. M. Douglas. *Conceptual design of chemical processes*. McGraw-Hill Book Company, New York, 1988.

M. A. Duran and I. E. Grossmann. An outer approximation algorithm for a class of mixed integer nonlinear programs. *Math. Program.*, 36:307–339, 1986.

G. Y. Feng, L. T. Fan, P. A. Seib, B. Bertok, L. Kalotai, and F. Friedler. Graph-theoretic method for the algorithmic synthesis of azeotropic-distillation systems. *Ind. Eng. Chem. Res.*, 42:3602–3611, 2003.

M. R. Fenske. Fractionation of straight run pennsylvania gasoline. *Ind. Eng. Chem.*, 24: 482–485, 1932.

P. Floquet, L. Pibouleau, and S. Domenech. Separation sequence synthesis: how to use the simulated annealing procedure. *Computers chem. engng.*, 18:1141–1148, 1994.

C. A. Floudas, A. Aggarwal, and A. R. Ciric. Global optimum search for nonconvex NLP and MINLP problems. *Computers chem. engng.*, 13:1117–1132, 1989.

E. S. Fraga. The automated synthesis of complex reaction/separation processes using dynamic programming. *Trans IChemE*, 74a:249–259, March 1996.

E. S. Fraga. The generation and use of partial solutions in process synthesis. *Trans IChemE*, 76:45–53, January 1998.

E. S. Fraga and K. I. M. McKinnon. CHiPS: A process synthesis package. *Trans IChemE*, 72a:389–393, May 1994.

E. S. Fraga, M. A. Steffens, I. D. L. Bogle, and A. K. Hind. *Foundations of Computer-Aided Process Design*, volume 323, pages 446–449. AIChE Symposium Series, 2000.

F. Friedler, K. Tarjan, Y. W. Huang, and L. T. Fan. Graph-theoretic approach to process synthesis: polynomial algorithm for maximal structure generation. *Computers chem. engng.*, 9:929–942, 1993.

A. M. Geoffrion. Generalized benders decomposition. *J. Optim. Theory Appl.*, 10:237–260, 1972.

E. R. Gilliland. Multi-component rectification, estimation of the number of theoretical platesas a function of reflux ratio. *Ind. Eng. Chem.*, 32:1220–1223, 1940.

I. E. Grossmann. Mixed-integer programming approach for the synthesis of integrated process flowsheets. *Computers chem. engng.*, 9:463–482, 1985.

I. E. Grossmann. Mixed-integer optimization techniques for algorithmic process synthesis. *Advances in Chemical Engineering*, 23:171–246, 1996.

O. K. Gupta and V. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Manage. Sci*, 31:1533–1546, 1985.

E. Hansen. *Global optimization using interval analysis.* Marcel Dekker, New York, 1992.

W. Ho and K. Sirkar, editors. *The membrane handbook.* Van Nostrand Reinhold, New York, 1992.

C. A. Jaksland, R. Gani, and K. Lien. Separation process design and synthesis based on thermodynamic insights. *Chem. Eng. Sci.*, 50:511–530, 1995.

W. R. Johns. Process synthesis: Poised for a wider role. *Chem. Eng. Prog.*, 4:59–65, 2001.

W. R. Johns and D. Romero. The automated generation and evaluation of process flowsheets. *Computers chem. engng.*, 3:251–260, 1979.

R. B. Kearfott. *Rigorous global search: continuous problems.* Kluwer academic publishers, Dordrecht, 1996.

J. F. Kennedy and J. M. S. Cabral, editors. *Recovery processes for biological materials*, page 90. John Wiley & Sons, Chichester, 1993.

S. Kirkpatrick, C. D. G. Jr, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

R. L. Kirkwood, M. H. Locke, and J. M. Douglas. A prototype expert system for synthesizing chemical process flowsheets. *Computers chem. engng.*, 12:329–343, 1988.

Z. Kohavi. *Switching and finite automata theory.* McGraw-Hill book company, 2nd edition, 1978.

E. Kondili, C. C. Pantelides, and R. W. H. Sargent. A general algorithm for short term scheduling of batch operations.1.MILP formulation. *Computers chem. engng.*, 17:211–227, 1993.

Z. Kovacs, Z. Ercsey, F. Friedler, and L. T. Fan. Exact super-structure for the synthesis of separation-networks with multiple feed streams and sharp separators. *Computers chem. engng.*, 23:S1007–1010, 1999.

Z. Kovacs, Z. Ercsey, F. Friedler, and L. T. Fan. Separation-network synthesis: global optimum through rigorous super-structure. *Computers chem. engng.*, 24:1881–1900, 2000.

Z. Kovacs, F. Friedler, and L. T. Fan. Recycling in a separation process structure. *AIChE Journal*, 39:1087–1089, 1993.

D. M. Laing and E. S. Fraga. A case study of synthesis in preliminary design. *Computers chem. engng.*, 21:S53–S58, 1997.

E. W. Leser, M. E. Lienqueo, and J. A. Asenjo. An expert system for the selection and synthesis of multistep protein separation processes. *Ann NY Acad Sci*, 782:441–455, 1996.

P. Linke and A. Kokossis. Attainable reaction and separation processes from a superstructure-base method. *AIChE*, 49:1451–1469, 2003a.

P. Linke and A. Kokossis. On the robust application of stochastic optimisation technology for the synthesis of reaction/separation systems. *Computers chem. engng.*, 27:733–758, 2003b.

B. Linnhoff and E. Hindmarsh. The pinch design method of heat exchanger networks. *Chem. Eng. Sci.*, 38, 1983.

E. Marcoulaki, P. Linke, and A. Kokossis. Design of reaction-separation networks using stochastic optimization methods. *TransIChemE*, 79:25–32, 2001.

E. McCarthy. *Synthesis of separation systems for multicomponent product problems*. PhD thesis, University of Edinburgh, 2000.

E. McCarthy, E. S. Fraga, and J. W. Ponton. An automated procedure for multicomponent product separation synthesis. *Computers chem. engng.*, 22:S77–S84, 1998.

R. E. Moore. *Interval Analysis*. Prentice Hall, Englewood Cliffs, New Jersey, 1966.

R. N. S. Rathore, K. A. van Wormer, and G. J. Powers. Synthesis strategies for multicomponent separation systems with energy integration. *AIChE J.*, 20:491–502, 1974.

H. Ratschek and J. Rokne. *New computer methods for global optimization*. Ellis Horwood, Chichester, 1988.

R. W. H. Sargent. A functional approach to process synthesis and its application to distillation systems. *Computers chem. engng.*, 22:31–45, 1998.

G. Schembecker, K. H. Simmrock, and A. Wolff. Sysnthesis of chemical process flowsheets by means of cooperating knowledge integrating systems. In *Institution of Chemical Engineers symposium series*, 133, 1994.

C. A. Schnepper and M. A. Stadtherr. Robust process simulation using interval methods. *Chem. Eng. Prog.*, 44:603–614, 1996.

J. D. Seader and A. W. Westerberg. A combined heuristic and evolutionary strategy for synthesis of simple separations sequences. *AIChE J.*, 23:951–954, 1977.

S. Skelboe. Computation of rational interval functions. *BIT*, 14:87–95, 1974.

E. M. Smith. *On the optimal design of continuous processes*. PhD thesis, Imperial College of Science Technology and Medicine, 1996.

M. A. Steffens, E. S. Fraga, and I. D. L. Bogle. Synthesis of bioprocess using physical properties data. *Biotechnol Bioeng*, 68:218–230, 2000.

G. Stephanopoulos and A. W. Westerberg. Studies in process synthesis ii. evoutionary synthesis of optimal process flowsheets. *Chem. Eng. Sci.*, 31:195–204, 1976.

A. J. V. Underwood. Fractional distillation of a multicomponent mixture. *Chem. Eng. Prog.*, 44:603–614, 1948.

O. M. Wahnschafft, T. P. Jurain, and A. W. Westerberg. Split: A separation process designer. *Computers chem. engng.*, 15:565–581, 1991.

K. Wang, Y. Qian, Y. Yuan, and P. Yao. Synthesis and optimization of heat integrated distillation systems using an improved genetic algorithm. *Computers chem. engng.*, 23:125–136, 1998.

R. R. Wehe and A. W. Westerberg. An algorithmic procedure for the synthesis of distillation sequnces with bypass. *Computers chem. engng.*, 6:619–627, 1987.

T. Westerlund and F. Pettersson. An extended cutting plane method for solving convex MINLP problems. *Computers chem. engng.*, 19:S131–S136, 1995.

H. Yeomans and I. E. Grossmann. A systematic framework of superstructure optimization in process synthesis. *Computers chem. engng.*, 23:709–731, 1999.

H. Yeomans and I. E. Grossmann. Disjunctive programming models for the optimal design of distillation columns and separation sequences. *Ind. Eng. Chem. Res.*, 39:1637–1648, 2000.

Y. Zhu and T. Kuno. Global optimization of nonconvex MINLP by a hybrid branch and bound and revised general benders decomposition approach. *Ind. Eng. Chem. Res.*, 42: 528–539, 2003.