Carnegie Mellon University
Research Showcase @ CMU

Department of Chemical Engineering | Carnegie Institute of Technology

1988

Process engineering

Arthur W. Westerberg
Carnegie Mellon University

Carnegie Mellon University.Engineering Design Research Center.

Follow this and additional works at: http://repository.cmu.edu/cheme

Published In

.

This Technical Report is brought to you for free and open access by the Carnegie Institute of Technology at Research Showcase @ CMU. It has been accepted for inclusion in Department of Chemical Engineering by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Process Engineering

by

Arthur W. Westerberg

EDRC 06-54-89

PROCESS ENGINEERING

by

Arthur W. Westerberg

Engineering Design Research Center

Chemical Engineering Department

Carnegie Mellon University

Pittsburgh, PA 15213

July, 1988

Copyright © 1988 Arthur W. Westerberg

This research has been funded by NSF Grant CDR-8522616

University Libraries
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Table of Contents

1. ABSTRACT **1**
2. INTRODUCTION **1**
2.1. A CAVEAT ON COVERAGE 1
3. **THE IMPACT OF COMPUTER TECHNOLOGY** **2**
4. DESIGN IN PROCESS ENGINEERING 6
4.1. A MODEL OF THE DESIGN PROCESS 10
5. **SYNTHESIS** 11
5.1. Expert Systems 13
6. **ANALYSIS** 13
6.1. Steady State Modeling 14
6.1.1. Flowsheetlng Programs 14
6.1.2. Convergence 17
6.2. Dynamics 19
6.2.1. Stiff Equation Sets 19
6.2.2. The Index Problem 20
6.2.3. Architectures for Simulators 21
6.3. Partial Differential Equations 22
6.4. Optimization 23
6.5. In Conclusion 26

1. ABSTRACT

Process engineering is the application of systems concepts to the design and operation of chemical processes. This paper emphasizes design. We first examine important developments in computer technology that have and will impact design. We propose a classification scheme for design problems and a model for how design is performed. We discuss engineering synthesis - i.e., the automatic generation and selection of design alternatives. A final section traces important developments in analysis methodology - i.e., in the setting up, solving and optimization of complex models.

2. INTRODUCTION

It is an honor to be asked to review the area of *process engineering* by the Department of Chemical Engineering at MIT on its centennial celebration. As requested, this paper provides an overview of the history, current state and and future of process engineering.

We define *process engineering* as the application of systems concepts and methodologies to the design and operation of chemical processes. One issue is the understanding of the behavior of the individual component parts comprising a process; a second is the understanding of the behavior of a system of integrated parts.

In this article we generally assume we understand how the individual parts behave and look at the issues arising from combining them into a system. This paper will discuss only design.

2.1. A CAVEAT ON COVERAGE

This area is extremely rich with potential topics, each of which would require a separate paper to review in depth. The reader should be aware that stringent page limitations (slightly exceeded by this manuscript) have been placed on these reviews so this paper can cover only one of the author's favorite topics, analysis, in any depth.

To give perspective, this paper has to look at computer technology because of its enormous impact on process engineering; a narrative style has been chosen to allow brevity and personal interpretations to be freely expressed. The author is currently the director of our Engineering Design Research Center at Carnegie Mellon and, from this experience, this paper presents a classification scheme to show the diversity of design problems one can face and then one possible model of the design process itself to suggest how one might organize ideas in this area.

One of the two discussers will cover synthesis (Douglas), and this author has just written two papers, each of which is in part a review of process synthesis (Westerberg, 1987, Westerberg, 1988). Several articles have reviewed synthesis in the last decade, one over 40 journal pages long. Thus a two page summary is obviously not going to say much. This paper will only offer a few ideas not typically discussed but which are felt to be important in process synthesis when heat effects do not dominate the decision making. A more extensive section on analysis completes this work.

Another discusser (Morari) will present ideas on plant operation and control.

3. THE IMPACT OF COMPUTER TECHNOLOGY

It is impossible to examine the area of process engineering without accounting for the impact of computer technology on it.

The first computer on which this author worked had a fast memory of 1024 words in which both the data and program had to reside. The hardware filled an entire room and, when running, one suspects could easily have heated the entire building within which it was housed. It was vintage mid 1950s. We programmed it using assembler code only. It could be used for setting up and solving relatively simple models of single pieces of equipment, like a heat exchanger or a simple column. It had less power than one of today's small hand held calculators which costs under $100. In the early 1960s computers such as the IBM 7094 and the CDC 6600 came into being, the first really powerful computers. They were programmed largely in early versions of Fortran and Cobol. One remembers remarks that the world could use perhaps five CDC 6600 computers.

We look around today at a very different world of computing. Supercomputers abound. The computing center is being augmented and sometimes even pushed aside in many organizations to give way to a distributed work station environment where everyone has a powerful computer either on his/her desk or in the next room. Everything is networked or rapidly becoming networked; large files are readily transferred from New York to Munich to Trondheim to Tokyo. Students routinely log into supercomputers across the country to run large computations. Electronic mail makes world wide communications direct and easy with such messages often transferred and responded to in minutes. 300 plus megabyte disk storage devices are available for $3000.

New machine architectures are providing remarkable performance, provided the software can be written to use the architecture. *Hypercube* computers available commercially are multidimensional arrays of hundreds of computers, each with the power of a microvax.

Not all computers are traditional in the way they carry out computations. Neural networks are another form of computer which receive input signals and produce output signals that characterize what was input. Demonstrations of these computers show they can be taught to recognize complex patterns. For example, these computers can be shown a picture of a person and then recognize that another picture is of this same person even when viewed from a different perspective. They have been taught to recognize connected speech.

There are new concepts in software revolutionizing computing, too. The Macintosh computer and its easy to use icon based software brings computing to virtually anyone willing to sit at the computer and try. We hear of four year old kids playing with computers and doing things many of us could not have done ten years ago without considerable training.

The UNIX operating system is offering for the first time the possibility of an operating system that is not dependent on the hardware vendor. The battle over operating systems is far from over, however.

There is a revolution going on in computer languages, too. An anecdote of the mid 1960s tells of someone from IBM being asked what would be the most popular language in the year 2000, e.g., Algol, Cobol or Fortran. He responded "I do not know how it will look, but it will be called Fortran." Those of us who have seen Fortran evolve in major ways to pick up the characteristics of many of these other languages will find that answer very insightful. However, there are other languages out there now (which will force more changes on Fortran) that are really different.

Their value is often based on subjective but very appealing arguments. Languages to support object oriented programming such as Smalltalk, Flavors or CommonLoops are an example. These languages are not like Fortran, at least not at this time. Among other things, they enforce a style to programming which many suggest reduces the time to program complex systems by factors of something like five.

Two features make them very interesting. The first is their extreme degree of modularity where each piece of code to do anything is called an object. The user at the screen is an object; the printer

is an object. Communications among objects is by messages which the executive system passes. Thus one object can send a message to another object representing a vessel and ask for its volume. The method to compute volume differs depending on the type of object, but lots of objects can respond to that message. Thus one can often remove an object in such a programming environment and replace it with another as long as it responds to the same messages without destroying the integrity of the system. Their second feature is the power with which they can represent information. They have in them the concept of inheritance structures. An object can be a class called column. One can send a message to this object and ask it to create an instance of itself and call it "B-T splitter." The B-T splitter inherits computational methods from the class called column; it also inherits default values for variables. Asked for the number of stages, the B-T splitter may not yet have a value posted, and the response will be a default from the class that might be a nominal value of 40. The response will be the actual value after it has been computed and posted within the B-T splitter object.

Not all computations are numerical in nature. The early to mid 1970s produced a flurry of computer science and some engineering literature about using computers to solve problems automatically using complex human reasoning. In the mid 1970s Minsky (Minsky, 1975) published a paper which has had a significant impact on the area of artificial intelligence. He posed the problem of two people carrying out a conversation something like the following: (Person A) Tomorrow is Mary's birthday; (Person B) I wonder if Mary would like a kite. We all understand that person B is thinking of giving Mary a kite for her birthday. There is nothing in this conversation that would let a computer know this meaning. Minsky argued that the computer would have to have in it a number of scenarios about common experiences. One would be about birthdays and that people exchange gifts. Another scenario would be that a kite and things like that are typical gifts. When the above conversation is examined, the computer would search its memory and retrieve relevant information which could be used to interpret what was implied by Person B. This recalled information would be a frame. (Thus started a controversy that exists even today as to what a frame is; even Minsky did not define it precisely.) An important property of frames is the notion of default information. Minsky argued that when someone mentions a car to you, you are likely to think about your car and not about just any car. If you are then asked about the weight of a car, you would likely estimate the weight based on the specific car you are thinking about. Only when someone particularizes the car by saying it is Jim's car, will you move away from the default car. A lot of understanding can come by reasoning using defaults.

Another idea of the early late 1960s and early 1970 was to produce a general problem solving capability suitable for all complex problems. This idea did not succeed. Next came the idea to encode knowledge about a particular problem domain which could then be applied to solving problems automatically in that domain, the concept behind knowledge based or expert systems. This idea is being explored extensively by almost every company and university. There are demonstrations which show that it really works.

Interestingly there are many disbelievers. The argument is that every concept promulgated by expert systems is well known and has been "done" in Fortran programs for years. There just was no name given to the concepts. The opinion of this author is that there are real issues in this area worthy of research and development. These systems offer utility systems called *shells* which make it easier to encode and use qualitative information in solving problems. One is not only modeling physical artifacts, but now one is trying to model the *process* of complex problem solving.

Computer scientists are trying to understand the really difficult issue of automatic learning. Can computer programs be constructed which can learn concepts and in doing so improve their performance when called to solve similar problems a second time? Perhaps the most difficult issue here is how to guess the correct generalizations when one is presented examples of a concept so what is learned is more generally applicable than to just the training examples. Humans guess these generalizations rather well. (Not all reasoning is deduction and induction. Complex problem solving seems to rely heavily on abduction, too. Abduction is where one guesses the solution and then proves it using the guess to direct the proof.)

Other significant unsolved problems include the following. How can the typical engineer find and use the best tools for the problem at hand? It can take months to find the tool and months to learn to use it. And it may prove to be the wrong tool. Another major problem is how to deal with the massive amounts of information that is available and being generated within a company. Data logging a process produces more information than anyone cares to think about. How can it be usefully saved and accessed? Should it be saved? The answer is a resounding yes when a problem occurs in the process.

The future looks very bright in computing. Supercomputers are about to show up on desk tops. Markedly increased speed will come in two ways: new architectures which allow massive parallel

processing and new technologies such as optical computing where performance factor increases of 10^5 are possible. There will be substantially improved understanding of the impact of language and information structuring in problem definition - a pet topic for this author (Piela et al, 1988). Everyone will be attached to a world wide information network where just about any piece of information will be available electronically. Means to search this base will improve to the point that the system will intelligently aid the user to find his/her way through the maze. Automatic language translation will exist; one form will be to translate the spoken word so the two participants in a telephone conversation may use different languages. Graphics will improve to the point that enormous amounts of information will be suitably summarized and presented to the engineer, some in the form of holograms. Neural network computers coupled with symbolic computers will add new power to problem solving.

It is interesting to wonder if the future will be more limited by a shortage of people who can determine how to use this power rather than by the power that will be available.

4. DESIGN IN PROCESS ENGINEERING

We return now to the main theme of this paper - process engineering. As mentioned earlier, process engineering concerns itself with the design and operation of processes. We now look at design.

We define *design* as that step during the creation of a new artifact or the modification of an existing artifact when one gathers together and generates information on all aspects of it to plan its fabrication or modification. By artifact we mean anything from a chemical process to a building to a computer code.

Studies in several industries suggest that design activities consume about 10 to 15 percent of the funds needed to move from the original artifact concept to its final creation. The same studies conclude that one makes decisions in the design step that fix about 80 percent of the final costs involved. A really bad design decision can almost never be compensated for by those who must fabricate the artifact. Thus design, while a small part of the activity, is the most important. Any manufacturing company which intends to remain competitive in today's marketplace must continually improve its design capability both in terms of people and tools.

To understand the important issues for a design, it is helpful to develop a classification scheme to characterize a design. If the scheme is appropriate, then the methodologies for designing should be related for designs similarly classified. We are using abduction to conjecture the usefulness of this classification scheme. See Fig. 4-1.

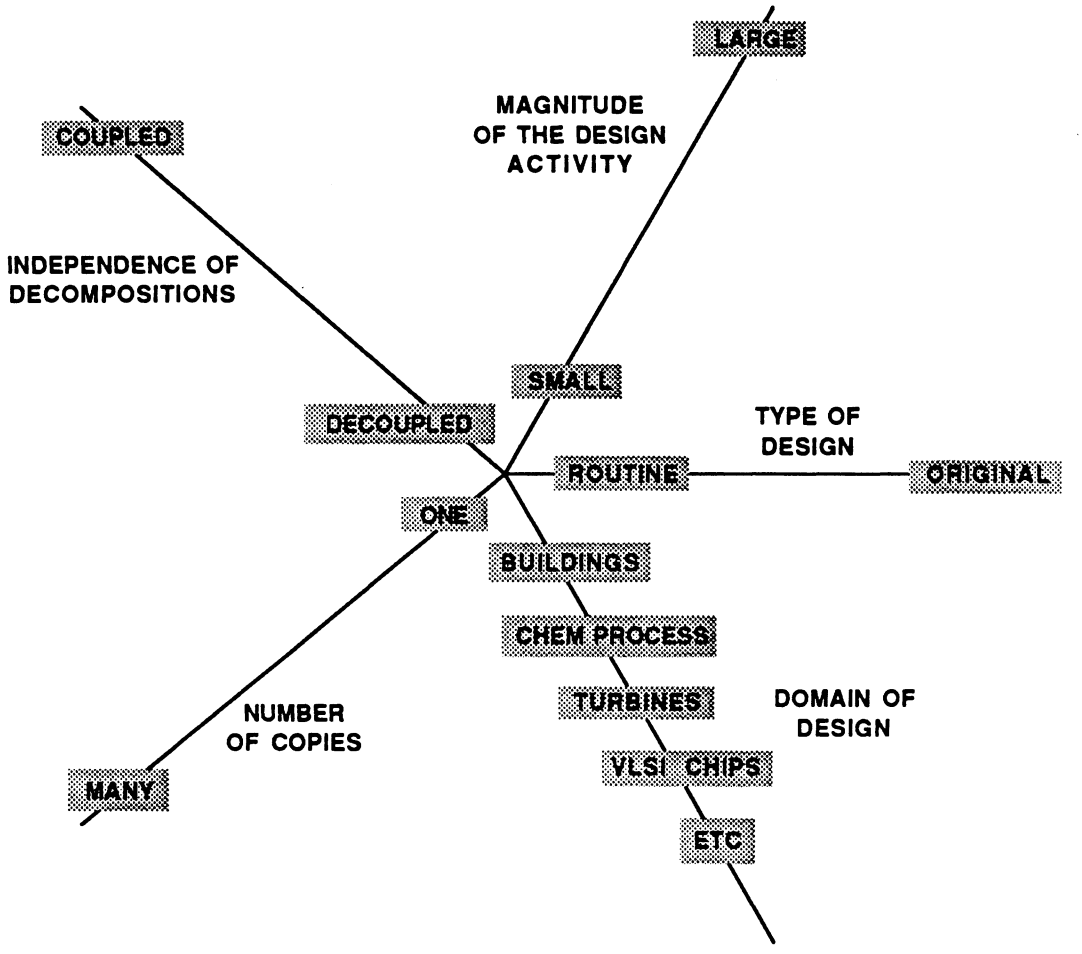

Figure 4-1: Space of Design Problems

Original versus Routine Design: A major argument typically ensues at any design conference on what is meant by design. This argument is frequently because one set of people is talking about *original* design, where the design team has never designed anything quite like the given artifact before, and the other group is talking about *routine* design, where the design concepts and methodologies are well understood by the design team. Many designs are between these two ex-

tremes. A design morphology must classify a design according to this measure as the important issues are very different for these two types of design problems.

For original design the activity is one of alternatingly brainstorming and gathering information, possibly by a geographically dispersed team. This activity is to generate the concepts on which the design and supporting design calculations will be based.

For routine design, the potential exists to *automate* the design activity completely. An example is the designing of a house in Japan. The customer looks into a catalogue for the style house desired - say, a "Frank Lloyd Wright" - and, for that style, enters the number of bedrooms and room size parameters which can be adjusted to customize the design into a form on a computer screen. The computer designs the house; it then generates instructions for the factory which will prefabricate the parts. Produced almost without human intervention in a factory that is creating parts for many different houses simultaneously, the parts for each house arrive at the loading dock just in time to be loaded together onto a truck. When dropped at the constructions site, the parts are in the order they are to be used. And they fit together. What contractor is going to survive in that market who refuses to use a similar technology?

Magnitude of Design Activity. There is a significant difference in the magnitude of the design activity needed to design a light switch versus a Boeing 747. For the former, the concept of *concurrent engineering* can be accomplished by putting a dozen people into a room for a few weeks, where each person represents a different concern such as the design, manufacture, use or sales of light switches. For the Boeing 747 one can imagine putting the entire team into a single room only at the early stages of design to create the overall design concept. Specialist teams will design the engine, the undercarriage, the wings and so forth, with each team requiring its own room. Here concurrent engineering requires a very different set of organizational controls. A question is, for example, how to propagate changes from those designing the engine to those designing the wing and vice versa.

One or Many Copies: One typically builds one copy of a chemical process or a major office building. At the other extreme, one will build millions of computer memory chips. The care taken during the design on getting all the details right varies dramatically with these two extreme cases. For a process, certain corrections can be taken during fabrication to overcome design flaws - one would like to avoid these but changes can be made. On the other hand, one could ill afford to design a memory

chip that had to be hand adjusted to make it function correctly. It has to be absolutely right when designed. The fabrication process for a one only production cycle is seldom highly automated - steps in it will be but the whole process will not; the fabrication process for chips is totally automated.

Integration Problems: When a design is accomplished by partitioning it, the parts must be integrated to form the solution. If the partitions are strongly coupled by important overall performance measures or constraints, integration is difficult. A typical grassroots chemical process design is first designed functionally. One then chooses the actual equipment. These decisions are not strongly coupled simplifying this aspect of the design activity dramatically. VLSI circuits have a similar strong decoupling allowing them to be designed functionally first. In contrast the aerodynamic performance of an airplane is determined by the totality of its shape and weight distribution. This characteristic, which is crucial to the performance of an airplane, cannot be assessed until the component parts are integrated. It strongly couples the design decision making for the subsystems.

In the retrofit design of a process, we must couple the decision making about function and the reuse of existing equipment. In an example involving the redesign of a distillation sequence to handle 30% more feed, this coupling caused an increase in the number of potential sequences from about 140 to over 4.5 million (Grossmann et al, 1987).

Improved decomposition of the design activity can strongly impact this dimension thus it is a measure of both the inherent characteristics of the artifact to be designed and the current effectiveness of the design methodology available.

This characteristic is often difficult to assess for a design. Even for a chemical process, the overall energy integration of the process can complicate the design decision making if one chooses to iterate on the design to find processes which heat integrate well. Safety can strongly couple the selection of equipment to the functional design step. At some later date we may have safety analyses that will allow for better decoupling.

Domain of the Design: A final axis is the domain of the design: chemical processes, computer programs, buildings, VLSI, electro-mechanical devices, etc. The knowledge required for reducing the size of design problems comes specifically from the domain of the design. However, the concepts and the form for the tools which use this knowledge to support design seem much more affected by

the other axes than by this one. (This conjecture is the principal hypothesis upon which the Engineering Design Research Center at Carnegie Mellon University is operating.)

4.1. A MODEL OF THE DESIGN PROCESS

Talukdar and Westerberg (Talukdar and Westerberg, 1988) propose that design is the moving among a number of different views or *aspects* for the artifact. At the highest level, the aspects are very abstract. At the lower levels more and more details are built up about the artifact.

Different aspects are needed to translate the design into a set of terms suitable for applying knowledge to the design to criticize it and/or to alter it. For example one may create a fault tree for a process to argue about its safety or a *cost diagram* (Douglas, 1988) to look at its capital and operating expenses. Temperature versus enthalpy diagrams help to think about the heat integration of processes.

One uses a synthesis step to move from an abstract representation to a more detailed one where *synthesis* is the activity of generating design alternatives and selecting the better ones based on incomplete information. The alternatives selected are often modeled in detail (perhaps by moving down through several more aspects) and *analyzed* to see if they indeed satisfy the requirements specified in the more abstract representation. Finally, if one is going to allow changes to be made to the more detailed representation, then one has to worry about mapping those changes back to the more abstract representation, an activity we can term *abstraction.* This latter activity is very difficult to accomplish in general. There seems to be no formal literature on how to do it (this statement is probably a controversial one of the type we were asked to make). Typically one has to guess the changes to make at the less abstract level which will permit one to have the properties desired at the more detailed level.

The complete design activity can be modeled as a network of aspects with operators among them that allow movement from one to another. Where the operators can be automated, one can create tools; for the others, the designer has to carry out the required step, often in an ad hoc manner.

With this model of the design process, a terminology becomes possible for describing different steps in it. One can then imagine formalizing approaches to carrying out design that are more general than the examples used to alert one to the concepts.

5. SYNTHESIS

To move from an abstract representation to a more refined one is accomplished by a synthesis step. A definition of the synthesis activity is as follows:

> *the automatic generation of design alternatives and the selection of the better ones based on incomplete information.*

Synthesis appears repeatedly throughout a design because of its recursive nature. It is the activity of *generating* alternatives to increase the net present worth of the company; it is the activity of generating a process flow diagram. In this latter case, this activity may itself have embedded synthesis activities such as the synthesis of separation subsystems or the synthesis of heat exchanger networks. It is also the act of *selecting* the better alternatives based on assessing the value of the design before one can prove it is one of the better ones.

There have been three issues listed as being significant in synthesis: representation, evaluation and search. The representation issue is related to the notion of aspects discussed above. The correct representation can often allow one to "see" the solution or how to generate just those alternative solutions which will be competitive to problems that otherwise look very difficult to solve. Representation is often based on developing the appropriate insights into the problem. For example, noting that a distillation column degrades heat to accomplish separation work motivates a "cascaded" heat flow representation for a column (Andrecovich and Westerberg, 1984) that is useful if one is trying to find how to heat integrate columns with the rest of the process.

An important synthesis problem is that of generating alternative chemical reaction paths by which a desired target product might be made. Thousands of reaction paths can be generated even for small molecules. How can one evaluate each of them? The correct evaluation is to establish the economics for the alternative processes which would be based on each of them. One simply cannot do this for thousands of alternatives so simpler evaluation functions are required which will allow one to screen out the poorer alternatives with little computational effort. An example is to look at the thermodynamics for the reactions and reject any for which equilibrium severely limits the production of products from the reactants.

The search space for synthesis is typically enormous. One must do it as effectively as is possible. Branch and bound methods, hierarchical structuring, constraint propagation, and heuristics to rule out whole sets of possible designs are among the search concepts which are useful.

Perhaps one of the most effective is the hierarchical structuring as one then makes high level decisions which will rule out very large numbers of lower level alternatives. The following illustrate the potency of this approach. Suppose one has a problem with $m_1 + m_2$ decisions, and one proposes to search by forming a grid of n points in each direction. If m_1 of the decisions can be made first using approximate reasoning and the remaining m_2 are then made, the search space reduces from one of size *rfi*^ to one of size H"7/*//^. Letting n, m_1 and m_2 all equal five, the sizes are 5^{10} (about 10^7) versus $2*(5^5)=6250$.

Separation system design offers an example. At the first level of decision making, we could classify the species and the phases involved to decide the type of likely separation methods which might be used. King (King, 1980) lists 54 different separation methods. Some are for gas/gas separation, others for gas/liquid, still others for gas/liquid/solid and so forth. An approach is to make high level decisions on whether gas/gas, gas/liquid etc type separation is needed for each of the separation steps required (Douglas, 1988). Then based on these decisions, lower level decisions are needed to select exactly which method to use.

Constraint propagation also reduces the search space size. It is to look at the problem from every view one can think of and in each of these views develop and propagate constraints on the solution space. In a separation system synthesis problem, one could for example look at the mixture and decide that the species D and E should be separated alone and by using distillation. This constraint dramatically reduces the search space size. Decisions not leading to D and E being isolated into a stream which will then be separated by distillation need not be considered in any enumeration scheme.

In chemical engineering the design of energy efficient processes has dominated the synthesis literature. The classical heat exchanger network synthesis problem is the most well developed. It is to find the structure of a heat exchanger network to exchange heat among a number of hot and cold streams within a process. This problem has been significantly aided by the discovery of very useful representations based on plots of temperature versus heat availability and heat need for a process.

These plots allow one the establish minimum utility requirements (Hohmann, 1971). Network arguments allow one to argue how many exchangers are needed. Methods exist to estimate the cost of both the utilities and the equipment without inventing the actual design. Understanding this problem has led to synthesis strategies Tor creating processes that will heat integrate well, such as the design of multieffect evaporator systems. A recent paper (Gundersen and Naess, 1988) reviews over 200 articles on this topic alone.

5.1. Expert Systems

Future synthesis programs are going to combine both quantitative and qualitative reasoning about the artifacts being designed. A serious issue will be the control of these programs so they can solve the problems opportunistically using every device available to reduce the search space. Lien (Lien, 1988a, Lien, 1988b) provides one interesting approach to structuring such a system to be opportunistic. We should see commercially available programs in the future for routine design problems such as physical property experts or absorption experts, replacing many of the design handbooks of today.

6. ANALYSIS

The tools needed to carry out an analysis step are likely the most well developed as aids for design in virtually every discipline. In analysis one proposes a model that can be used to describe the behavior of whatever phenomena is of interest. Such models can range from worrying about how an oil droplet might sit on top of a pool of water to worrying about the production scheduling of an entire company. They can range in level of detail; the oil droplet could take the power of a supercomputer to solve while the company model might exist on a personal computer.

As pointed out in the Amundson report (Amundson, 1988), Chapter 7, the (super)computer of the future will have enough power to allow "ab initio" computations which we are learning how to pose to replace laboratory experiments. To do this *analysis* step, we need to be able to set up and solve really large scale computations. We might ask how well we can do this and what the problems have been and are.

At least these two issues arise: setting up models and solving them. To appreciate the problems, imagine having to set up a model comprising 10,000 equations in 11,000 variables. Try doing this without writing too many equations nor too few in terms of the variables involved, then try to figure out which 1000 variables to fix - not every set will leave the remaining 10,000 equations in 10,000 variables nonsingular, then try to give values to the 1000 variables you have fixed in such a manner the problem has a solution, and finally find that solution. Lately people have added the twist: find all the solutions that might exist.

Not all models are necessarily specified as equations to be solved. Some are specified procedural^, in the form of rules (as in production systems such as OPS-5 or OPS-83) and/or in qualitative terms. These models offer even more interesting problems for solving which can take us into the domain of expert system concepts. Learning to pose and solve such models effectively is still an area for research.

Since pure equation based models involving as many as a few hundreds of thousands of equations arise and need to be solved, we see the need for ways to assure this activity can occur with some degree of success. Perhaps the oldest of such approaches comes with flowsheeting programs.

6.1. Steady State Modeling

We look in this section at developments in the setting up and solving of large scale steady state models for complete chemical processes.

6.1.1. Flowsheeting Programs

Today engineers routinely compute heat and material balances and preliminary sizing and cost estimations for a traditional chemical process using commercially available flowsheeting programs. These programs permit one to set up a model for an arbitrarily configured process quickly and to solve that model using different degrees of rigor both for the equipment models and for the physical properties required. These models, with the physical property equations included, are often tens of thousands of equations in size.

These programs first came into being in the late 1950's (Piesler and Kessler, 1960). Flowsheeting systems tied individual unit models together so that entire processes could be modeled. They almost

always use the *sequential modular architecture.* In it each unit model is represented by a subroutine which is designed to compute the unit output streams given the unit input stream values and enough unit parameters to fix its performance. An example is to compute the vapor and liquid streams out of an isothermal flash unit given the feed stream enthalpy, pressure, composition and flowrate and the temperature and pressure of the flash unit.

Tying these computations together gives a model which has the same recycles in it that exist in the actual process being modeled. These recycles have to be guessed to start the computations. When values for them are computed as outputs of later units, the computed values have to be compared to those guessed. If essentially the same the computation terminates, otherwise new guesses are needed and the process interated.

Early research in flowsheeting involved discovering automatically the better streams to use as the ones to guess (tear streams) by selecting the order in which the unit subroutines should be called. Also methods were published to improve the guessing, which in its simplest form is to use the values just computed - what is termed "successive substitution." Often the designer wishes to specify values for intermediate streams parameters which in the sequential modular architecture have to be computed, not specified. Computational controllers compare the computed values for these streams to those specified and force the flowsheeting program to be run iteratively while the controller adjusts some input parameters until the desired intermediate stream behavior is met.

In early attempts at optimization, pattern search based methods put yet another loop around these programs. Fifteen hundred flowsheet simulations to find the optimum were not uncommon; the computation often failed.

By the late 1960's, the idea surfaced to gather together the equations for an entire flowsheet model and automatically derive a solution procedure to solve them. Almost no one in industry took this idea too seriously as whatever tests were run on the idea showed it failed too often. Also the problems got too large too fast, particularly if the physical property evaluation equations were included in the set.

The first efforts in academia to derive solution procedures were based on extending the automatic tearing ideas being used in solving flowsheets. In tearing a small subset of the variables are guessed and iterated to solve the entire set of equation. Tearing ideas floundered in the early to mid 1970's.

Variables had to be guessed for which no intuition existed, effective solution seemed to require symbolic manipulation of the equations which was way too slow, and finally some neat embedded tearing algorithms - loops within loops - were doomed to failure. This last idea looks really useful when the inner equations are linear. However, all too often the inner loops are singular even though the overall problem is not.

The Newton/sparse matrix methods being used by electrical engineers have become the solution method of choice. Hutchison and his students at Cambridge were among the first chemical engineers to publish this approach, in the early 1970's. They used a quasilinear model rather than a Newton one, but the ideas were really very similar. (It would appear that the COPE flowsheeting system of Exxon was Newton-based; it existed in the mid 1960's, but it slowly evolved into a sequential modular system. One must assume the Newton method failed to compete.)

By now several equation-based flowsheeting systems exist; perhaps the best known of these is SPEED-UP which has been and is continuing to be developed at Imperial College by Sargent and Perkins and their students. It is now commercially available; its strongest attraction in industry seems to be as a dynamic simulator. TISFLO at Dutch State Mines is also often mentioned in the literature. As a complete flowsheeting system, no equation based approach is yet very popular. One might wonder why.

Equation based systems still have two characteristics which users find unattractive. They are a half order of magnitude slower at solving simple flowsheeting problems; thus they lose when compared in timing studies. They fail to converge as often. This failure can almost certainly be ascribed to the initial guesses used. A large part of the code in the subroutines modeling units in sequential modular systems is used to get a close initial guess. None of the equation based systems as yet has a comparable capability for making an initial guess. When they do, the two approaches will likely have comparable convergence characteristics - the equation based approach may even turn out to converge more often.

Finally, the attractiveness of equation-based flowsheeting systems is that one can choose fairly arbitrarily which variables to fix and which to compute. Inputs can be computed in terms of outputs, etc. This attractiveness, however, gives the designer freedom that he/she is often unable to use correctly. Deciding which variables to fix is difficult, particularly when there are a thousand of them to

be selected. Since the sequential modular approach has preselected which are fixed, getting the degrees of freedom right for that approach is generally not a serious problem. The equation solving approach does not preselect (its strength and interestingly also its weakness). Two forms of aids are possible: default selecting to pick most of those variables to be fixed (e.g., the molecular weight for methane should be treated as being fixed), and a set of algorithms which can carry out structural and numerical analyses to aid in getting the degrees of freedom set correctly (Barnard et al, 1986). Generally these aids are not available.

6.1.2. Convergence

A major research issue has been the development of approaches to aid the convergence of difficult problems or to speed up the solving of easier ones. Making better initial guesses is certainly crucial; it may be the single most important issue in converging stubborn problems. Yet, strangely enough there is little in the literature about aiding this problem. One should look with a jaundiced eye at many reported convergence results. Seldom does a paper tell of the many false starts that the author made which failed to converge until the initial guess became good enough. Slight perturbations in the initial guesses can move one from a problem that refuses to converge to one that marches directly to the solution.

There are alternatives to getting initial guesses; examples include: (1) use default nominal values that reflect the type of the variable - 300 K for temperatures, 10,000 Pa for pressures, 0.3 for mole fractions, (2) use ad hoc coding to set initial guesses - linearly interpolate the temperatures in a column section, and (3) solve the equations for a computation which is a more natural one first and then switch the degrees of freedom to those desired - solve a column fixing the reflux first and then fix the top product purity and let the reflux be computed with the first solution being the initial guess to the second.

If one has made the best guess possible, then more rugged methods exist to get to an answer, for example *continuation methods* (Weyburn and Seader, 1987). In continuation, one picks a scalar parameter such as t - think of it as time - that one will move from zero to unity (some versions move to infinity) while the solution is to move in a continuous manner from one that is easily found to the one that is being difficult. Thus one *creeps* up on the solution.

The equation for such a method is as follows.

$$H(x,t) = (1-t)\, g(x) + f/(;c) = O \qquad (1)$$

As *t* moves from zero to one, H(x,t) moves from the solution of g(x)=0 to that of f(x)=O. g(x) are a set of functions for which the initial solution is trivial to find; f(x) are the equations one is having difficulty solving.

Continuation methods have an interesting degree of freedom: the direction in which *t* will move. Often to get to the solution *t* moves first in the positive direction, then for a time in the negative direction and then again in the positive direction. Thus these methods typically change the independent variable for moving from Mo s, the distance the continuation path has moved in the *[x,t]* space.

One continuation method reconstructs exactly the Newton method when *t* moves in the *positive* direction. Think of the surface which corresponds to summing the squares of the functions one wishes to drive to zero. If the Newton method flounders in a local hole in this surface where the bottom of the hole does not reach down to zero and thus where the equations do not have a solution, it would be very useful to climb out of the hole by going in the reverse of the Newton direction (i.e., by simply reversing the sign on *t*), hopefully over the top of a nearby ridge and down the other side into a hole where a solution does exist. A continuation method does just this.

Kuno and Seader (Kuno and Seader, 1988) show how to use continuation methods to find all the solutions for a set of nonlinear equations. One simply continues past the first instance when *t* equals unity to find all other instances where it again equals unity. They show how to select a starting point for the search to guarantee all solutions will be found. No proof exists for their method, but they have tested it extensively without a known failure.

Vickery and Taylor (Vickery and Taylor, 1986) suggests using *t* as the exponent for those parts of expressions which are giving convergence difficulties. Used in this fashion, *t* can be termed a *natural continuation* parameter. For example consider writing the following vapor/liquid equilibrium relationship with the parameter *t* included as shown.

$$y_i = \left(\frac{\gamma_i}{\phi_i}\right)^t \frac{f_i^o}{P} x_i \qquad (2)$$

When *t* is zero, the relationship expresses essentially ideal behavior, *t* moving to unity slowly introduces the nonideality expressed by liquid activity and the vapor fugacity coefficient. Taylor and others report solving some very difficult problems using this approach.

6.2. Dynamics

We discuss here the problem of solving models which characterize the dynamic behavior of processes. These models in general comprise a set of ordinary differential equations (ODEs), a set of algebraic equations and a set of initial conditions from which the solution can be started. One solves by converting the ODEs into a set of approximating algebraic equations which, together with the algebraic equations, can be stepped incrementally through time to trace out the trajectories of the variables. Each step is the solving of a set of algebraic equations for the variables at time step *k+1* in terms of those at time step *k.*

6.2.1. Stiff Equation Sets

How one forms the approximations for the ODEs is crucial to the performance of this approach. Gear (Gear, 1971) and many other since showed how *implicit* methods convert the ODEs so the solution method is stable and can therefore be used to solve stiff sets of equations. Implicit methods give algebraic equations that generally must be solved iteratively at each time step as they usually involve the variables at time step *k* nonlinearly.

Great debates have raged over what is meant by "stiff." A single equation can be stiff (Sincovec et al, 1981), e.g., consider the following one:

$$\frac{f}{t} = -10^6 (y - \sin(t)) + \cos(t); \; y(0) = 1 \qquad (3)$$

The solution is y = sin(t) + expf-IO^6^. However, any deviation from that solution is blown up by the first term on the right hand side in the ODE above and will cause numerical problems. Thus one will have to take very small steps in *t* to generate what appears to be a slowly moving smooth curve. Stiffness should reflect the size of the step that one has to take versus the size one thinks should be needed when looking at the solution.

6.2.2. The Index Problem

Sincovec et at (Sincovec et al, 1981) presented a very disquieting example of an initial value problem consisting of two ODEs. They showed that only one of the two state variables involved can be given an independent initial value. It was not hard to see why the problem occurs, but it was evident that such a problem could easily be hidden in a larger example. This work also proves that if an incorrect initial condition is specified and an implicit integration scheme is used, the solution will march directly to a solution which corresponds to one where a legitimate initial condition was used. One should note that the initial condition may not be one of interest, however.

Petzold (Petzold, 1982) and Gear and Petzold (Gear and Petzold, 1984) defined an *index* for mixed sets of ODEs and algebraic equations which indicates if the numerical scheme used to compute the solution might have unexpectedly poor error characteristics. Crudely speaking, the way the degrees of freedom have been specified for a problem can force one or more of the approximation equations to be used *backwards,* i.e., to differeniate rather than to integrate. The error estimate is "worse" roughly by h^2, where h is the integration step size; i.e., where one might think the method is second order in step size, the error could in fact be independent of the step size. If more than a single equation is used backwards, the error behavior is even worse. One would use the approximation equations backwards if one were to compute the control variable trajectory which will give a prescribed state variable behavior. They present a method to compute this index for linear or linearized problems. Gear (Gear, 1988) showed that the index problem could be eliminated by differentiating the equations.

Pantelides (Pantelides, 1986, Pantelides et al, 1987) in his work with Sargent defined the index in a manner that exposes its potential to cause problems in initialization as well as in the integration error. They too showed that the index problem can be eliminated by differentiation. Noting that only some of the equations need to be differentiated, they use a method based on the structural properties of the equations to discover these equations. They cite several examples in which the index problem is almost certain to occur in setting up and solving dynamic simulation models, e.g., flash calculation dynamics and problems where the trajectory of a state variable is specified.

6.2.3. Architectures for Simulators

Implementation of dynamic simulators have led to interesting research issues. For example, many have been implemented in a sequential modular format. To carry out the integration correctly from the point of view of correctly assessing integration errors, each unit model can receive as input to it a current estimate for the "state" variables (variables x), the unit input streams variables, and any independent input variables specified versus time (variables u), and it can then calculate the unit output variables and the right-hand-sides for the dynamic equations provided they are written in the following form.

$$\frac{dx}{dt} = f(x,z,u,t) \tag{4}$$

where z are the variables whose values are established by the algebraic equations in each of the unit models.

The executive can then solve the models in an appropriate sequence to converge the stream variables which are involved in a process recycle. Once these are converged, the executive can then use the RHSs to integrate simultaneously the differential equations for all the units in this recycle loop.

Brosilow presented an approach to coordinate the solving of unit models where each integrates it own ODEs internally, using whatever integration method it chooses and using its own step size control. The executive system coordinates this activity. This approach is very appealing, but care is required to assure the overall system integration errors are correctly assessed and maintained. The advantage is of course that a "quiet" or slowly moving unit will have little computational work to do.

Many current dynamic simulators are equation based. Thus they seem to require that all parts of the simulation move together in the integration. Kuru (Kuru, 1981, Kuru and Westerberg, 1985) present an approach to take advantage of the modularity of the flowsheet but in an equation solving environment. A partitioning scheme for the Newton equations which accounts for the modularity of the flowsheet allows one to converge all the equations but with differing numbers of iterations, e.g., with no iterations for the equations for units which are not moving. Thus *latency* is moved into the Newton scheme. There are no approximations being made so error handling is unaffected.

6.3. Partial Differential Equations

There is an enormous literature in the area of integrating of partial differential equations. This author has very limited experience in this area and is therefore unwilling to say much here. However, there has to be a strong overlap with the concepts covered above for solving both algebraic and mixed ODE/algebraic models. Typically PDEs are converted into very large structured sets of approximating algebraic equations in terms of variable values at a number of strategically located grid points, either using finite difference or finite element concepts. These equations relate values of neighboring grid points so they will approximately satisfy the PDE operator in some optimal sense. In other approaches they are discretized only in *n-1* of the *n* coordinate directions and thus are converted into a set of ODEs.

One of the interesting research issues currently receiving much attention is the placing of the grid points for the discretization. For a single distributed variable such at temperature, one can see how to place the points, i.e., put more points where the variable is changing more rapidly and fewer where it is not. How does one do it for many distributed but coupled variables when each is changing at quite different rates in different parts of the space? This problem sounds like slow and fast moving units in a dynamic simulation, only here there is no natural modularity that occurs within a flowsheet of interconnected units.

One idea for grid point placement is to guess at a placement, solve, and then based on the solution, add and delete points where it seems appropriate, solve, regrid, etc. Another is to solve for the variable values and the grid point locations simultaneously. This latter approach can give rise to a very much larger equation set to be solved and to sets of equations that have a high probability to be singular or nearly so. The grid placement is often done to minimize an error criterion that measures the error between the PDE operator and the equations used to discretize it. Often the grid placement has little impact on this error estimate. Then the equations for grid placement become singular. One scheme to correct this problem is to add penalty functions to the error criterion. A better one would seem to be to remove the degrees of freedom if they are not needed - i.e., remove the equations which place the points and substitute others which will space them out in a reasonable fashion if that of minimizing the error is not helpful. The issue is to leave just enough points whose location will impact the error and move the others relative to these. Handling nearly singular equation sets for flowsheet modeling is certainly related to this line of thinking.

Another issue that seems interesting to think about: how to handle discontinuities. Often this has been done by increasing the number of grid points. Why not simply remove the constraints that are forcing continuity at the break points and let the temperature or composition profile take on two values at that grid point? Mavaridis etal. (Mavaridis et al, 1987) has shown how useful this idea is. Again one has to discover where to make these changes as the solution evolves.

Finally it seems one should be worrying about the index problems for mixed sets of PDEs and algebraic equations. One cannot help but wonder how many problems have been solved where some "derivable" independent equations have been missed.

6.4. Optimization

This author reviewed optimization in an earlier article (Westerberg, 1981) in a manner consistent with this presentation. Therefore this section will only summarize some of the ideas and mention some results that have occurred since that review.

Optimization first became possible for large scale problems with the development of linear programming codes in the late 1950s. The oil companies very quickly adopted this methodology for the scheduling of refineries. With the introduction of mixed integer linear programming in the 1960s, these same companies started to use these codes to aid in making design decisions. For example, should a new refinery include a fluidized catalytic cracking unit or not? A binary variable, one which takes on a value of zero or one only, was used to indicate the existence (value of one) or non-existence (value of zero) of a part of the proposed design. A modest amount of nonlinear behavior could be added by approximating nonlinear behavior with straight line segments; the cost is the adding of a large number of binary variables.

Early attempts to add optimization to flowsheeting calculations were not well received. Here optimization requires the handling of nonlinearities. These attempts used pattern search approaches which proved to be extremely slow and not very reliable. These approaches require only the evaluation of the objective function and constraint violations; i.e., they do not use gradient information. One would regularly see reports of 1000 to 2000 function evaluations (i.e., flowsheet simulations) to carry out an optimization using pattern searches. These approaches often stalled completely on ridges.

Industry did not adopt this methodology for solving real problems.

An approach which has enjoyed industrial success is the application of sequential linear programming. Here the nonlinear problem is locally linearized, with bounds placed on the allowed moves for all the variables so the local linearization is still appropriate. An iteration is to linearize, solve the corresponding linear program, move to the solution of the LP and converge that solution to the solution of the nonlinear problem. This point is the start of the next iteration. Industry has used this approach fairly successfully for a number of years.

Gradient based algorithms such as the generalized reduced gradient and, in the last ten years, sequential quadratic programming, offer much better performance for optimizing. Experience with the latter suggests that one can often optimize a flowsheet in the equivalent time that it takes to solve the flowsheet only two to three times - compare that to the 1000 to 2000 times for pattern search approaches. Most of the commercially available flowsheeting packages have introduced optimization capabilities based on these concepts.

Attempts to use algorithms which require gradients were and often continue to be thwarted by the modeling techniques used within flowsheeting systems. Frequently the models contain IF statements which are used to switch from one type of model behavior to another depending on the value of the variables. These switches cause unsmooth or even discontinuous behavior, which can destroy gradient based algorithms. Examples are to switch from a laminar to a turbulent formula to estimate friction factor for flow in a pipe as the Reynolds number passes through 2100 and to switch from a two phase flash calculation to a one phase computation as the flash temperature passes below the bubble point or above the dew point. Physical properties packages are notorious for containing such discontinuous or unsmooth behavior, e.g. their complex decision making for the handling the roots of the (cubic) equation of state models. Even the subtle differences caused by using inner convergence loops that may iterate a different number of times each time can provide unsmooth behavior to the outer optimization algorithm.

These IF statements are really a form of discrete decision making embedded within the model. One possible approach to remove the difficulties caused by it is to move the discrete decisions to the outside of model and the continuous variable optimizer. For example, the friction factor equation can be selected to be the laminar one irrespective of the Reynolds number that is computed later. Con-

straints can be added to forbid movement outside the laminar region or to forbid movement too far outside the laminar region. If the solution to the well behaved continuous variable optimization problem (it is solved with few iterations) is on such a constraint boundary, tests can be made to see if crossing the constraint boundary can improve the objective function. If so the boundary is crossed - i.e., a new value is given to the discrete decision, etc.

Grossmann and his students (Duran and Grossmann, 1986a, Duran and Grossmann, 1986b, Duran and Grossmann, 1986c, Kocis, 1988)have produced some very exciting new developments in the solving of mixed integer *nonlinear* programs. As indicated above integer variables can be used to allow discrete changes in a model such as the addition or removal of a unit. It can thus be a tool for design where one first sets up a superstructure within which are embedded the various alternatives that one wants to consider for a design. Such a tool is very powerful when used for problems with highly coupled decisions.

The approach is as follows. First the designer creates the superstructure for the artifact to be designed, say a flowsheet. Then he/she selects a substructure in that superstructure which is thought to be a good candidate for the optimal one. This selection corresponds to picking values for all the discrete (binary) variables. This flowsheet alternative is optimized using a nonlinear programming code, e.g., one which uses reduced gradients or sequential quadratic approximation concepts. Next at the optimal solution to this alternative, the flowsheet modeling equations are linearized using an *outer approximation* to the nonlinear equations. An outer approximation is required to ensure that no part of the solution space for the continuous variables is cut off in the linearization. (One cannot always guarantee that such an approximation has been used - more in a moment on this problem.) A constraint known as a *cut* is added to ensure the previous solution for the discrete decisions is not found again, i.e. the sum of the binary variables set to unity for this solution is set to one less than the number of these variables. This linear approximation with the discrete decisions is solved as a mixed integer *linear* program to find a new set of discrete decisions. These will correspond to a different flowsheet alternative. If one has used a proper outer approximation, the MILP solution is a lower bound on the cost of the flowsheet it finds. This new flowsheet is optimized (a nonlinear program again). Again this flowsheet is linearized and these equations are *added* to the previous linear model; the combined linearization with another cut to eliminate the previous solution is again solved using an MILP code to discover yet another flowsheet alternative. The process terminates when the MILP

solution costs more than the best flowsheet alternative found so far. It is not allowed to reuse any previous solution so it is saying that the next best solution is too expensive.

In their work Grossmann and his coworkers have shown that this approach converges amazingly fast, often in two to five cycles (i.e., five NLP/MILP cycles). Only a few seconds to minutes of computer time is required for the problems solved, one of which was a complete flowsheet model for the hydrodealkylation of toluene process described in Douglas (Douglas, 1988). The model had several hundred equations in it only so it is a simplification of a rigorous model. Some of the very interesting contributions in the just completed work with Kocis have been a better approach to handle equality constraints, an approach to make improved linearizations for those units which are inactive in the initial solution, a linearization scheme for mixers and splitters that guarantees these will be "outer approximated" (their models are nonconvex), and a scheme (admittedly ad hoc but apparently often effective) to detect and make adjustments when nonconvexities in the problem have led to incorrect outer approximations.

6.5. In Conclusion

We see from the above discussion that there are many new results occurring in analysis. The Amundson Report (Amundson, 1988) indicates that speed of computers is doubling every year, partly from new hardware capabilities and partly from new numerical techniques. We see little that suggests either will slow for some time. Most exciting will be the use of the computer for mixed qualitative/quantitative problems where human like reasoning will aid in getting problems correctly stated and solved.

References

Gear, C. W. & Petzold, L. R. (1984). ODE Methods for the Solution of Differential/Algebraic Systems. *SIAMJ. Numer. Anal.,* 27(4), 716-728.

Amundson, N.R. (1988). *Frontiers in Chemical Engineering - Research Needs and Opportunities.* Washington D.C.: National Academy Press. National Research Council Report.

Andrecovich, M.J., and A.W. Westerberg. (1984). A Simple Synthesis Method Based on Utility Bounding for Heat-Integrated Distillation Sequences. *AIChE J, 31,* 363.

AIChE. (Mar 1986). *Three Issues in the Design of an Equation-Based Process Simulator.* AIChE 1986 Spring National Meeting.

Douglas, J.M. (1988). *Conceptual Design of Chemical Processes.* McGraw-Hill.

Duran, M.A., and I.E. Grossmann. (1986). An Outer Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs. *Math Prog, 36,* 307-39.

Duran, M.A., and I.E. Grossmann. (1986). A Mixed-Integer Nonlinear Programming Approach for Process Systems Synthesis. *AIChE J,* 32(4), 592-606.

Duran, M.A., and I.E. Grossmann. (1986). Simultaneous Optimization and Heat Integration of Chemical Processes. *AIChE J,* 32(1), 123-38.

Gear, C.W. (1971). *Numberical Initial Value Problems in Ordinary Differential Equations.* Englewood Cliffs, NJ: Prentice-Hall.

Gear, C. W. (1988). Differential_Algebraic Equation Index Transformations. *SIAM J. Sci. Stat Comput,* 9(1), 39-47.

Grossmann, I.E., A.W. Westerberg, and LT. Biegler. (1987). Retrofit Design of Processes. Reklaitis, G.V., and H.D. Spriggs (Eds.), *Foundations of Computer Aided Process Operations.* New York, NY: CACHE/Elsevier.

Gundersen, T., L. Naess. (1988). The Synthesis of Cost Optimal Heat Exchanger Networks. An Industrial Review of the State of the Art. to appear in Comput. Chem. Eng., Spring.

Hohmann, E.C. (1971). *Optimum Networks for Heat Exchange.* Doctoral dissertation, Chemical Engineering, Univ. of So. California,

King, C.J. (1980). *Separation Processes, 2nd Edition.* New York: McGraw Hill.

Kocis, G.R. (1988). *A Mixed-Integer Nonlinear Programming Approach to Structural Flowsheet Optimization.* Doctoral dissertation, Carnegie Mellon Univ.,

Kuno, M., and J.D. Seader. (1988). Computing **All** Real Solutions to Systems of Nonlinear Equations **with a Global Fixed-Point** Homotopy. *Ind. Eng. Chem. Res., 27,*1320-29.

Kuru, S. (1981). *Dynamic Simulation with an Equation Based Flowsheeting System.* Doctoral dissertation, Carnegie Mellon Univ.,

Kuru, S.f and A.W. Westerberg. (1985). A Newton-Raphson Based Strategy for Exploiting Latency in Dynamic Simulation. *Comp. Chem. Eng.,* 9(2), 175-82.

Lien, K.M. (1988). *Expert Systems Technology in Synthesis of Distillation Sequences.* Doctoral dissertation, University of Trondheim, Norwegian Institute of Technology, Laboratory of Chemical Engineering,

Lien, K.M. (June 13-15 1988). *Expert Systems in Design* (Tech. Rep.). European Federation of Chemical Engineering, Gotenborg, Sweden.

Mavaridis, H., A.N. Hrymak, and J. Vlachopoulos. (1987). Finite-Elememt Simulation of Stratified Multiphase Flows. *AlChEJ,* 33(3), 410-22.

Minsky, M. (1975). A Framework for Representing Knowledge. Winston, P. (Eds.), *The Psychology of Computer Vision.* New York, NY: McGraw-HilP.

Pantelides, C.C. (1986). *The Consistent Initialisation of Differential-Algebraic Systems* (Tech. Rep.). Imperial College, London, submitted for publication.

Pantelides, C.C, D. Gritsis, K.R. Morison, and R.W.H. Sargent. (Apr 26-30 1987). *The Mathematical Modelling of Transient Systems Using Differential-Algebraic Equations* (Tech. Rep.). XVlll Congress, Use of Computer in Chemical Engineering Conference, Glardini Naxos, Italy,

Petzold, L. R. (1982). Differential/Algebraic Equations Are Not ODE's. *SIAM J. Sci. Stat. Compute* 3(3), 367-384.

Piela, P.C. (1988). ASCEND - An Object Oriented Environment for Mathematical Modeling. EDRC Report in preparation.

Piesler, A.H., and M.M. Kessler. (1960). The Computer Approach to Optimizing Plant Design. *Refining Engineer, 32,* C2.

Sincovec, R.F., Erisman, A. M., Yip, E.L. & Epton, M. A. (1981). Analysis of Descriptor Systems Using Numerical Algorithms. *IEEE Trans. Automatic Control, AC-26^),* 139-147.

Talukdar, S., and A.W. Westerberg. (Mar 6-10 1988). *A View of Next Generation Tools for Design* (Tech. Rep.). AlChE, New Orleans, LA, U.S.A.,

Vickery, D.J., and R. Taylor. (1986). Path-Following Approaches to the Solution of Multicomponent, Multistage Separation Process Problems. *AIChE J, 32*, 547.

Westerberg, A.W. (1981). Optimization in Computer Aided Design. Mah, R.S.H., and W.D. Seider (Eds.), *Foundations of Computer-aided Chemical Process Design.* New York, NY: Engineering Foundation.

Westerberg, A.W. (1987). Process Synthesis: A Morphological View. In Liu, Y.A., H.A. McGee, and W.R. Epperly (Eds.), *Recent Developments in Chemical Process and Plant Design.* John Wiley and Sons, Inc.

Westerberg, A.W. (June 1988). *Synthesis in Engineering Design* (Tech. Rep.). Chemdata88 Conference, Gothenburg, Sweden,

Weyburn, T.L., and J.D. Seader. (1987). Homotopy Continuation Methods for Computer-Aided Process Design. *Comput. Chem. Eng., 11*, 7.