UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

NEW METHODS AND NEW PARADIGM TO DESIGN SENSOR NETWORKS FOR PROCESS PLANTS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

DUY QUANG NGUYEN THANH
Norman, Oklahoma
2009

NEW METHODS AND NEW PARADIGM TO DESIGN SENSOR NETWORKS FOR PROCESS PLANTS

A DISSERTATION APPROVED FOR THE
SCHOOL OF CHEMICAL, BIOLOGICAL AND MATERIALS ENGINEERING

BY

_____
Dr. Miguel J. Bagajewicz, Chair

_____
Dr. Mohammed Atiquzzaman

_____
Dr. Dimitrios V. Papavassiliou

_____
Dr. Lance L. Lobban

_____
Dr. Alberto Striolo

DEDICATION

I dedicate this dissertation to my parents, whose unconditional love and support carry me this far

ACKNOWLEDGEMENTS

I am grateful for the financial support from National Science Foundation for making my studying here possible. Travel financial support from University of Oklahoma is also gratefully acknowledged. I also thank OU Supercomputing Center for Education and Research (OSCER) staff for providing resources (OSCER super computers) and helpful suggestions for my research.

I would like to thank my research advisor Dr. Miguel J. Bagajewicz for his guidance throughout the course of this work.

I would like to thank my friends, my classmates and officemates in Chemical Engineering department, with whom I had useful discussions on lectures and research.

Finally, I would like to express my deepest gratitude to my parents who always stand by and encourage me. My academic achievement is dedicated to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Due to economic reason, not every process variable can be measured by a sensor. In the context that data treatment techniques like data reconciliation and gross errors detection are used, the location of measured points (i.e. location of sensors) has direct effect on the accuracy of estimators of variables of interest (key variables), which in turn effect process plant performance. The problem of optimum selection of sensor location is referred to as sensor network design problem (SNDP). More details on the problem and research works up to year 2000 can be found in Bagajewicz (2000).

Being a combinatorial optimization problem, the SNDP poses significant computational challenges for researchers, especially for large scale problems. The methods to solve the SNDP can be divided into two three classes: mathematical programming, graph-theoretic methods and stochastic methods (e.g. genetic algorithm)

The SNDP problem itself can be divided into two big classes: designing sensor network intended for process monitoring purpose (to obtain accurate process data) and designing sensor network for process fault diagnosis and resolution. The former can be solved by many methods while the latter is usually solved by graph-theoretic methods

Although extensive researches have been done on this problem, efficient methods to design sensor networks for large scale nonlinear problems have not yet been found. Moreover, all the published models are developed from technical point of view, which requires knowledge / expertise of the users to use appropriate constraints / specifications in the model. A model that bases solely on an economic viewpoint has not yet been proposed.

Addressing the mentioned drawbacks is the objective of this work. More specifically, in this work:

i) Efficient computational methods to solve SNDP for large scale nonlinear problems are proposed.

ii) A value-optimal SNDP is proposed and solved by using appropriate methods.

1. INTRODUCTION

The first chapter aims to give a general overview of different approaches and methods used to solve the sensor network design problem (SNDP). Additionally, the objectives of this work are presented.

Improved process monitoring via data reconciliation and appropriate gross error detection is achieved and/or improved by proper systematic location of sensors in a process plant. In all cases, the notion was to take advantage of the process relations, presented in the form of a mathematical model, to obtain accurate estimate for certain variables (called key variables) using available measurements. Considering the fact that, in reality, only a fraction of process variables are measured by sensors, the precision of the estimators depends on the location of the sensors and the precision of the sensors themselves. Hence the problem of systematically locating sensors that meets pre-specified criteria arises naturally, formally known as the sensor network design/retrofit problem.

This chapter is organized as follows: firstly some background on data treatment technique and overview of SNDP are introduced, then a brief summary of different approaches and different methods to solve SNDP is presented, followed by a literature review on SNDP. Finally an overview of the main content of this dissertation is given.

1.1. Background

1.1.1. Data reconciliation

In a modern chemical plant, process measurements are used in a variety of activities such as planning, process control, optimization, and performance evaluation. The presence of random and nonrandom errors (gross errors) in "raw" measurement data, can easily lead to deterioration in plant performance. The problem of improving the accuracy of process data so that they are consistent with material and energy balances of the system is known as data reconciliation. Process data after being treated by data reconciliation technique is called reconciled data or the estimators. Thus, data reconciliation is the technique to improve the accuracy of process data by making use of process constraints (typically material and energy balances).

The essence of data reconciliation is that given the process measurements $\boldsymbol{y}$ from the plant, we want to estimate the process state $\boldsymbol{x}$, which satisfies the process constraints. We denote these reconciled values of process data as $\hat{\boldsymbol{x}}$. $\hat{\boldsymbol{x}}$ is obtained by solving the following problem:

$$\begin{aligned} &\underset{x}{\text{Min}}(\boldsymbol{y}-\boldsymbol{x})^{T}\boldsymbol{S}^{-1}(\boldsymbol{y}-\boldsymbol{x}) \\ &\text{st.} \qquad \boldsymbol{A}\boldsymbol{x}=0 \end{aligned} \tag{1-1}$$

where $\boldsymbol{S}$ is the variance-covariance matrix of measurements (and usually is a diagonal matrix) and $\boldsymbol{A}$ is incidence matrix (i.e. $\boldsymbol{Ax = 0}$ is the process balance equations like material balances)

Assuming all process variables are measured, the variance of the estimators $\hat{\boldsymbol{x}}$ is given by (Bagajewicz, 2000):

$$\hat{\boldsymbol{S}} = \boldsymbol{S} - \boldsymbol{S}\boldsymbol{A}^{\boldsymbol{T}}(\boldsymbol{A}\boldsymbol{S}\boldsymbol{A}^{\boldsymbol{T}})^{-1}\boldsymbol{A}\boldsymbol{S} \tag{1-2}$$

The variance $\hat{S}$ is used in the SNDP as precision of the estimators, which needs to be compared against the threshold values.

If not all variables are measured (as is usually the case), it has been shown that an unmeasured variable can be modeled in data reconciliation formulation using a fake sensor with very high variance (Chmielewski et al., 2002). This approach greatly facilitates the SNDP because it eliminates the need to solve the data reconciliation problem for partly measured systems.

1.1.2. Gross error detection

Gross errors are systematic errors that can exist in measurements (measurement biases) and process model (process leaks). Measurement bias relates to malfunction of instruments and is the more prevalent form of gross error. Even when only one gross error exists, it deteriorates accuracy of all measurements in the process system through "*smearing effect*" of data reconciliation. The reason is that a large deviation from true value in one measurement (i.e. gross error) will cause a series of small "corrections" made to other measurements through data reconciliation treatment. Thus it is crucial that gross errors are detected, identified and eliminated.

The maximum power measurement test (MPMT) is probably the most popular technique to detect biases. The measurement test (MT) is based on the vector of measurement adjustments $\boldsymbol{m}$:

$$\boldsymbol{m} = \boldsymbol{y} - \hat{\boldsymbol{x}} = \boldsymbol{SA^T}(\boldsymbol{ASA^T})^{-1}\boldsymbol{Ay} \tag{1-3}$$

The maximum power measurement test (MPMT) proposed by Mah and Tamhane (1982) is based on vector ***d***, which is obtained by premultiplying ***m*** by $\boldsymbol{S}^{-1}$ ($\boldsymbol{d = S^{-1}m}$). The test statistics, given by Eq. (1-4), have been shown to possess maximum power if ***S*** is a nondiagonal matrix (Mah and Tamhane, 1982):

$$Z_{d,j}^{MP} = \frac{|d_j|}{\sqrt{\boldsymbol{W}_{jj}}} \tag{1-4}$$

where $Z_{d,j}^{MP}$ is the maximum power measurement test statistic for measurement j; d_j and W_{jj} are elements of vector ***d*** and matrix ***W***, ***W*** is given by $\boldsymbol{W = SA^T(ASA^T)^{-1}AS}$. If the test statistic $Z_{d,j}^{MP}$ is larger than the threshold values Z_{crit} (equal to 1.96 at level of confidence of 95%), then measurement j is declared to contain gross error.

The expected value of $Z_{d,j}^{MP}$ is given by Eq. (1-5) (Bagajewicz, 2005):

$$E[Z_{d,j}^{MP}] = \frac{\left|\sum_i W_{ji}\delta_i\right|}{\sqrt{W_{jj}}} \tag{1-5}$$

where δ_i is actual bias in measurement i

1.1.3. Redundancy and Observability

Observability and redundancy are defined as follows (Narasimhan and Jordache, 2000):

Observability: a variable is said to be observable if it can be estimated by using the measurements and steady-state process constraints.

Redundancy: a measured variable is said to be redundant if it is observable even when its measurement is removed.

From the above definition of observability, it is obvious that a measured variable is observable, since its measurement provides an estimate of the variable. However, an unmeasured variable is observable if it can be indirectly estimated by exploiting process constraint relationships and measurements in other variables. Measured variables are redundant if they can also be estimated indirectly through other measurements and constraints even when their measured values are eliminated.

1.2. Sensor network design problem

1.2.1. Model formulation

The popular cost optimal sensor network for process monitoring is formulated as follows (Bagajewicz, 1997)

$$\begin{aligned} & Min \sum_{\forall i} c_i q_i \\ & s.t. \\ & \sigma_i(q) \le \sigma_i^* \qquad \forall i \in M_S \\ & q_i = 0,1 \qquad \forall i \end{aligned} \tag{1-6}$$

where q_i is the vector of binary variable indicating that a sensor is located in variable i, c_i is the cost of such a sensor and M_s represents the set of variables where a certain specification is required (desired level of precision / residual precision or error delectability, etc.), $\sigma_i(q)$ is the value of the property under consideration (e.g. precision). A brief description of network properties other than precision is given next

1.2.2. Residual precision

Residual precision is the ability of the network to guarantee a certain level of precision in key variables where the measurements are eliminated because the sensors either fail or are found to contain biases (Bagajewicz, 1997). Formally, a variable has a residual precision of order k, when the specified value of residual precision is maintained even after k measurements, regardless of their position in the network, are eliminated (Bagajewicz, 1997).

1.2.3. Error detectability

The ability of the network to detect k gross errors of a certain dimensionless size κ^D or larger is called error detectability of order k (Bagajewicz 1997). More specifically, when measurements follow a normal distribution, the objective function of data reconciliation follows a central chi-square distribution with m degree of freedom. Using these concepts (which was developed by Madron, 1985; 1992), Bagajewicz (1997) provided an inequality that relates κ^D to the noncentrality parameter and the variances of the measurements and the estimator, respectively.

$$\kappa_i^D \geq \omega \frac{\sigma_{i,m}}{\left(\sigma_{i,m}^2 - \sigma_i^2\right)^{1/2}} \tag{1.7}$$

This inequality needs to hold for gross error detectability of order k=1. No inequalities were developed for higher order.

1.2.4. Resilience

If a gross error of a certain magnitude occurs in some variable and is not detected, a certain corruption of data will take place when data reconciliation is performed. The ability of the network to limit the smearing effect of k undetected gross errors of a certain adimensional size or lower is called gross error resiliency of order k (Bagajewicz, 1997).

1.2.5. Illustrated SNDP example

Example 1.1. Consider the following process example, which is shown in Figure 1.1

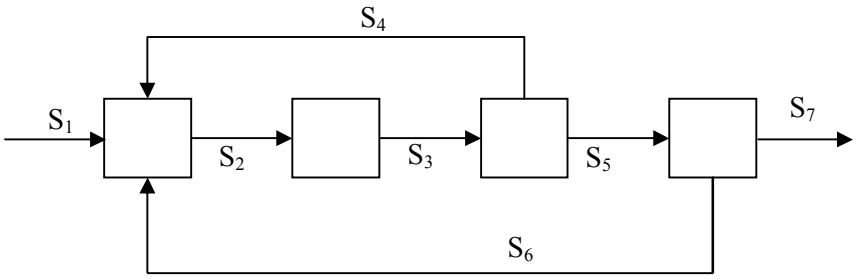


Figure 1.1 - Example process

Flowrate, precision and cost of sensors for example 1.1 are shown in Table 1.1

Table 1.1- Data for example 1.1

Stream	Flow rates	Sensor precision (%)	Sensor cost
S_1	100	2	55
S_2	140	2	40
S_3	140	2	60
S_4	20	2	50
S_5	120	2	45
S_6	20	2	55
S_7	100	2	60

The design specification and the obtained optimal solutions for four design cases are shown in the first six rows of table 1.2. The last two rows show the optimal solution (optimal measurements location and optimal cost)

Table 1.2 - Results for example 1.1

Case Study	**1.1a**	**1.1b**	**1.1c**	**1.1d**
Key variables	S_1 & S_5	S_1 & S_5	S_1 & S_5	S_1 & S_5
Requirement	Observability	Redundancy	Redundancy & error detectability	Redundancy & error detectability & resilience
Precision thresholds	1.5%	1.5%	1.5%	1.5%
Residual precision thresholds		4%	4%	4%
Error dectectability thresholds			4	4
Error resilience threshold				3
Measured variables	S_1, S_6, S_7	S_1, S_5, S_6, S_7	S_1, S_5, S_6, S_7	S_1, S_2, S_4, S_5, S_6
Sensors cost	170	215	215	245

Notes

- For design case **1.1c**: in addition to redundancy requirement, it is required that biases whose magnitude is greater than four times the standard deviation be detected
- For design case **1.1d**: in addition to the specifications used in design **1.1c**, it is required that the induced biases in key variables causes by any biases in the systems be less than three times the standard deviation

A few observations can be withdrawn from the above results:

- For observability problems (key variables are required to be observable only), the solution of directly measuring all the key variables is usually not the optimal one. Moreover, if the required precision thresholds are smaller than the standard deviation of sensors, then the number of measurements is usually larger than (or at least equal to) the number of key variables (because more measurements are needed to improve the precision of the estimators). These are evidenced in design case **1.1a** (directly measuring S_1 & S_5 makes S_1 & S_5 observable but the precision of the estimators of S_1 & S_5 are above the threshold values).
- As one increases the level of specifications (e.g. more requirements to be satisfied), the obtained (feasible) solution would contain more sensors.

1.3. An overview of different approaches and methods to solve SNDP

The SNDP can be divided into two classes:

- Class one: designing sensor network for process monitoring purpose. More specifically, the sensor networks are designed to provide accurate estimators (measured or estimated value) for the process variables of interest (key variables). The most popular model formulation is to find cost-optimal sensor network satisfying a certain number of pre-specified requirements (e.g. observability & redundancy of key variables). The problems of data reconciliation and gross error detection of partly measured systems are inherent parts of the process monitoring-focused SNDP.
- Class two: designing sensor network for process fault detection and isolation purpose. This problem is based on the principle that a process fault (malfunction / failure in an instrument in a process) at one point in the system will propagate to other locations in the system, which would eventually be detected by the sensors-based monitoring system. The sensor network will be able to detect the fault if it can detect the symptoms of the fault. The two different faults can be differentiated from each other if their symptoms (as shown up in the monitoring system) are different. The problem of detecting and identifying faults (using various well-established techniques) is an inherent part of the problem.

Computational methods to solve SNDP can be divided into three classes:

- Mathematical programming: the problems are transformed into well-established optimization models such as mixed integer linear programming MILP (usually solved by using GAMS, a commercial software for solving optimization problem) or solved as an integer programming problem (usually solved by using branch and bound method). These methods guarantee optimality but they suffer from scaling problem, that is, the computational time for large scale problems are usually unacceptably long.
- Graph-theoretic methods: the methods are based on the many principles and operators of graph theory. The SNDP for fault diagnosis purpose is usually solved by using graph-theoretic methods.
- Stochastic methods: genetic algorithms are usually used to solve multi-objective SNDP

1.4. Literature review

After the seminal work of Vaclavek and Loucka (1976) several papers were published: Kretsovalis and Mah (1987) minimized a weighted sum of estimation error and measurement cost using a combinatorial search algorithm. Madron and Veverka (1992) used multiple Gauss Jordan elimination to achieve observability of all key variables at minimum sensor cost. Meyer et al. (1994) used graph oriented approach for cost-optimal sensor network design with requirement on observability of key variables. Luong et al. (1994) considered several requirements in the design of sensor network: observability of variables required for process control, required degrees of redundancy

for some variables, reliability of the measurement system and minimum sensor cost; computational method is based on the analysis of cycles of process graph.

Bagajewicz (1997) was the first to formulate the sensor network problem as a mixed-integer programming (MINLP) model using binary variables to indicate whether a variable/stream is measured or not, sought to obtain minimum sensor cost. He also introduced new concepts regarding performance specifications: residual precision, gross errors resilience and error detectability. The problem was solved using a tree search algorithm, which guarantees global optimality but its computation requirement inhibits its use in large scale problems. A generalized model for grass root sensor network design, instrumentation upgrade as well as resource allocation were presented by Bagajewicz and Sanchez (2000). Finally, all literature review up to the year 2000 can be found in the book by Bagajewicz (2000).

Chmielewski et al. (2002) showed that an unmeasured variable can be modeled in data reconciliation formulation using a fake sensor with very high variance and used branch and bound method with linear matrix inequalities (LMI) transformation to obtain solution. The idea of using a fake sensor with very high variance for unmeasured variable enabled one to state certain types of performance constraints explicitly in analytical form. It also greatly facilitates the solving of the data reconciliation of partly measured systems, which is an inherent part of the SNDP. The idea was used by Bagajewicz and Cabrera (2002), who presented a MILP formulation, and many other researchers (Carnero et al., 2001, 2005; Muske and Georgakis, 2003; Gala and Bagajewicz, 2006a, 2006b).

Recently, multiobjective sensor network design became attractive and many researches have been reported. Bagajewicz and Cabrera (2001) addressed multiobjective sensor network design using pareto optimal solutions visualization techniques. Muske and Georgakis (2003) discussed the trade-off between measurement cost and process information that is used for control purpose and formulated a Pareto optimization problem for finding solutions. Sen et al. (1998) and Carnero et al. (2001, 2005) used genetic algorithms.

Gala and Bagajewicz (2006a, 2006b) presented an alternative tree enumeration method where at each node combinations of process graph cutsets are used. This method has been proven to be remarkably faster, especially after a decomposition technique is used. Most recently, Kelly and Zyngier (2008) presented a MILP model based on the Schur complements theorem to design sensor network for process monitoring purpose. The authors showed that the model can quickly find "good" solutions but locating global optimum solution is too time-consuming.

Departing from process monitoring criteria, Ali & Narasimhan (1993) introduced the concept of system reliability and proposed a method that maximizes system reliability. Raghuraj et al. (1999), Bhushan and Rengaswamy (2000, 2002a & 2002b) presented sensor network design formulation based on fault diagnosis criteria. Musulin et al. (2004) used genetic algorithm in the design of sensor network for principal components analysis monitoring. Bagajewicz et al. (2004) designed sensor network for simultaneous process monitoring and fault detection / resolution purpose using a MILP formulation. Bhushan et al. (2008) presented a framework for designing robust sensor network for reliable process fault diagnosis; the problem was then solved by using

constraint programming (Kotecha et al., 2007, 2008). Chen and Chang (2008) used graph-theoretic method to design sensor network for process fault identification.

Most of the aforementioned work was applied to linear systems, that is, when flowrates between units in process plants are to be estimated. Few researchers have published work on sensor network design for nonlinear processes, that is, when the underlying process model includes other balances like component and energy balances as well as other features like VLE relations or reactions in reactors. One such effort was presented by Ali and Narasimhan (1996) who developed a sensor network design program specifically for bilinear systems but maximizing system reliability instead of cost. Bhushan and Rengaswamy (2002a, 2002b) presented a general framework based on graph theory for finding a reliable sensor network from a fault diagnosis perspective. Heyen et al. (2002) used genetic algorithm to design cost-optimal sensor network that renders required precision of key variables; the computation algorithm can be applied to nonlinear systems by linearization of process constraints at the nominal operating conditions, assuming steady state. Singh and Hahn (2005, 2006) and Brewer et al. (2007) located sensors for state and parameter estimation of stable nonlinear systems.

All the mentioned works on SNDP is the cost-paradigm approach in which minimum sensor cost is the objective and the performance targets (the requirements) need to be selected by the plant engineers. However, practical engineers may find it hard to comprehend and determine what desired levels of targets are needed. It is well known that there is a trade-off between technical requirements and the economical requirement (minimum sensors cost), that is, if one increases the technical requirements (add more constraints or increase the desired levels in the constraints), in most of the cases the

optimum sensors cost is increased (more sensors are needed to satisfy all the constraints). However, there are situations that one relaxes the performance requirements (e.g. lower the desired levels or use less constraints) inconsiderably but obtains a significant reduction in sensors cost, or increases significantly the specifications but the extra cost incurred is small. Therefore, the right strategy to find optimum sensor network is to simultaneously optimize performance of the sensor network and the sensors cost. If the performance of sensor network can be translated into economic value or profit, then one can disregard the performance constraints and use economic value of performance of sensor network as a term in a composite objective function, which is value minus cost. The resulting sensor network design problem is an unconstrained optimization problem maximizing value minus cost of sensor network. This approach is value-paradigm SNDP in contrary to the conventional cost-paradigm SNDP. The approach has been conceptually discussed in the seminal paper by Bagajewicz, Chmielewski and Rengaswamy (2004). The work by Narasimhan and Rengaswamy (2007) is the only published work that discusses the value (as a performance measure) of a sensor network (from fault diagnosis perspective). Designing sensor network (for *process monitoring* purpose) maximizing value of sensor network is not yet addressed.

The two mentioned shortcomings will be addressed in our work. In summary, the objectives of this work are:

i. Developing efficient computational methods for the design of cost-optimal sensor network for process monitoring purpose of nonlinear systems.

ii. Studying the problem of sensor network design (from *process monitoring* perspective) that is based on the value of sensor network. Efficient computational methods to solve the problem are also proposed.

Three different methods for solving nonlinear SNDP are presented in this work: equation-based tree search method (chapter two), "level-by-level" tree search and approximate method (chapter three). Chapter four presents our study on the value-based SNDP for process monitoring purpose. Chapter five concludes this dissertation with summary of findings and discussions of future works.

1.5. References

Ali, Y., and Narasimhan, S. Sensor Network Design for Maximizing Reliability of Linear Processes. *AIChe J.* **1993**, 39(5), 820-828.

Ali, Y., and Narasimhan, S. Sensor Network Design for Maximizing Reliability of Bilinear Processes. *AIChe J.* **1996**, 42(9), 2563-2575.

Bagajewicz, M., Design and Retrofit of Sensors Networks in Process Plants. *AIChe J.* **1997**, 43(9), 2300-2306.

Bagajewicz, M. *Design and Upgrade of Process Plant Instrumentation.* Technomic Publishers, Lancaster, PA, **2000**.

Bagajewicz M. On the Definition of Software Accuracy in Redundant Measurement Systems. *AiChE J.* **2005**, 51(4), pp. 1201-1206.

Bagajewicz, M. and Cabrera, E. A New MILP Formulation for Instrumentation Network Design and Upgrade. *AIChe J.* **2001**, 48(10)**,** 2271-2282.

Bagajewicz, M. and Cabrera, E. Pareto Optimal Solutions Visualization Techniques for Multiobjective Design and Upgrade of Instrumentation Networks. *Ind. Eng. Chem. Res.* **2003,** *42,* 5195-5203.

Bagajewicz, M., Chmielewski, D and Rengaswamy, R**.** Integrated Process Sensor Network Design**.** Proceedings of the AiChe annual meeting, **2004**, Austin, Texas.

Bagajewicz, M., Fuxman, A. and Uribe, A**.** Instrumentation Network Design and Upgrade for Process Monitoring and Fault Detection**.** *AIChE J*. **2004**; 50(8), 1870-1880.

Bagajewicz, M. and Sanchez, M. Reallocation and Upgrade of Instrumentation in Process Plants. *Comput. Chem. Eng.* **2000***,* 24**,** 1945-1959.

Bhushan M. and R. Rengaswamy. Design of Sensor Network Based on the Signed Directed Graph of the Process for Efficient Fault Diagnosis. *Ind. Eng. Chem. Res.* **2000**, 39, 999-1019.

Bhushan M. and R. Rengaswamy. Comprehensive Design of Sensor Networks for Chemical Plants Based on Various Diagnosability and Reliabilty Criteria. I. Framework. *Ind. Eng. Chem. Res*. **2002a**, 41, 1826-1839.

Bhushan M. and R. Rengaswamy. Comprehensive Design of Sensor Networks for Chemical Plants Based on Various Diagnosability and Reliabilty Criteria. II. Applications. *Ind. Eng. Chem. Res.* **2002b**, 41, 1840-1860.

Bhushan M., S. Narasimhan and R. Rengaswamy. Robust sensor network design for fault diagnosis. *Comp. Chem. Eng.* **2008**, Vol. 32, 1067–1084.

Brewer J., Z. Huang, A.K. Singh, M. Misra and J. Hahn. Sensor Network Design via Observability Analysis and Principal Component Analysis. *Ind. Eng. Chem. Res.* **2007**, Vol. 46*,* 8026-8032

Chen J.Y. and C.T Chang. Development of an Optimal Sensor Placement Procedure Based on Fault Evolution Sequences. *Ind. Eng. Chem. Res.* **2008**, Vol. 47*,* 7335–7346

Chmielewski, D.,Palmer, T.,Manousiouthakis, V. On the Theory of Optimal Sensor Placement. *AIChe J.* **2002**, 48(5), 1001-1012.

Carnero, M., Hernandez J., Sanchez, M. and Bandoni, A. An Evolutionary Approach for the Design of Nonredundant Sensor Networks. *Ind. Eng. Chem. Res.* **2001**, 40, 5578-5584.

Carnero, M., Hernandez J., Sanchez, M. and Bandoni, A. On the Solution of the Instrumentation Selection Problem. *Ind. Eng. Chem. Res.* **2005,** *44,* 358-367

Gala, M. and Bagajewicz, M. J. Rigorous Methodology for the Design and Upgrade of Sensor Networks Using Cutsets. *Ind. Eng. Chem. Res*. **2006a**, 45(20), 6687-6697.

Gala, M. and Bagajewicz, M. J. Efficient Procedure for the Design and Upgrade of Sensor Networks Using Cutsets and Rigorous Decomposition. *Ind. Eng. Chem. Res*. **2006b**; 45(20), 6679-6686.

Heyen, G., Dumont, M. and Kalitventzeff, B. Computer-aided design of redundant sensor networks, in: J. Grievink, J. van Schijndel (Eds.), *Proceeding of 12th European Symposium on Computer-aided Process Engineering*, Elsevier Science, Amsterdam, **2002**, pp. 685–690.

Kelly J. D and Zyngier D. A New and Improved MILP Formulation to Optimize Observability, Redundancy and Precision for Sensor Network Problems. *AIChe J.* **2008**, 54(5), 1282-1291.

Kotecha P. R., Bhushan M., and R.D. Gudi. Constraint Programming Based Robust Sensor Network Design. *Ind. Eng. Chem. Res.* **2007**, Vol. 46*,* 5985-5999.

Kotecha P. R., Bhushan M., and R.D. Gudi. Design of robust, reliable sensor networks using constraint programming. *Comp. Chem. Eng.* **2008**, Vol. 32, 2030–2049.

Kretsovalis, A. and R. S. H. Mah, Effect of Redundancy on Estimation Accuracy in Process Data Reconciliation. *Chem. Eng. Sc.* **1987***,* 42**,** 2115.

Luong, M.; Maquin, D.; Huynh, C. and Ragot, J. Observability, Redundancy, Reliability and Integrated Design of Measurement Systems. *Proceeding of 2nd IFAC Symposium on Intelligent Components and Instrument Control Applications*, Budapest, Hungary, Jun 8-10, **1994**

Madron, F., and V. Veverka, Optimal Selection of Measuring Points in Complex Plants by Linear Models, *AIChe J.* **1992**, 38(2)**,** 227.

Mah, R.S.H. and Tamhane, A.C. Detection of Gross Errors in Process Data. *AIChE J.* **1982**, Vol. 28, 828-830.

Meyer M.; Le Lann, J.; Koehret, B. and Enjalbert, M. Optimal Selection of Sensor Location on a Complex Plant Using a Graph Oriented Approach. *Comput. Chem. Eng*. **1994**, *18* **(**Suppl), S535-S540.

Muske, K. R. and Georgakis, K. Optimal Measurement System Design for Chemical Processes. *AIChe J.* **2003**, 49(6), 1488-1494.

Narasimhan S. and Jordache C. *Data Reconciliation & Gross Error Detection: An Intelligent Use of Process Data*. Gulf Publishing, Houston, Texas, USA, **2000**

Narasimhan S. and R. Rengaswamy. Quantification of Performance of Sensor Networks for Fault Diagnosis. *AIChe J.* **2007**, 53(4), 902-917.

Raghuraj, R., Bhushan, M. and Rengaswamy, R**.** Locating Sensors in Complex Chemical Plants Based on Fault Diagnostic Observability Criteria. *AIChe J.* **1999**, 45(2), 310-322.

Sen, S., Narasimhan, S. and Deb, K. Sensor Network Design of Linear Processes Using Genetic Algorithms. *Comput. Chem. Eng.* **1998**, **22**, 385-390.

Singh, A. K. and Hahn, J. Determining Optimal Sensor Locations for State and Parameter Estimation for Stable Nonlinear Systems. *Ind. Eng. Chem. Res*. **2005,** 44*,* 5645-5659

Singh, A. K. and Hahn, J. Sensor Location for Stable Nonlinear Dynamic Systems: Multiple Sensor Case. *Ind. Eng. Chem. Res*. **2006,** 45*,* 3615-3623

Vaclavek V. and M. Loucka. Selection of Measurements Necessary to Achieve Multicomponent Mass Balances in Chemical Plant, *Chem. Eng. Sc.* **1976**, 31**,** 1199.

2. EQUATION-BASED TREE SEARCH METHOD FOR SOLVING NONLINEAR SENSOR NETWORK DESIGN PROBLEM

Nonlinear SNDP pose significant computational challenge for researchers because of the high level of interaction between units in the system. Large scale nonlinear SNDP have not yet efficiently solved. In this work the cutsets-based methods (that were previously developed for linear systems) were extended and generalized to solve nonlinear problems. An alternative tree search method developed specifically for problems with high level of specifications is also presented.

2.1. Overview

As mentioned in chapter one, most of the published works on SNDP were developed for linear systems only. Few researchers have published work on nonlinear SNDP: Ali and Narasimhan (1996) developed a sensor network design program specifically for bilinear systems but maximizing system reliability instead of cost. Bhushan and Rengaswamy (2002a, 2002b) designed a general SNDP from a fault diagnosis perspective that is applicable to nonlinear systems. Heyen et al. (2002) designed sensor network for general systems including nonlinear systems using genetic algorithm. Only the work of Heyen et al (2002) is closely related to our work because of the same perspective/objective, which is designing cost-optimal sensor network for process monitoring purpose, but their GA-based procedure does not guarantee optimality, neither local nor global. Thus, the nonlinear SNDP from the process monitoring

perspective with requirements on observability and redundancy has not yet been successfully tackled.

In this chapter, a branch and bound procedure similar to the one presented by Gala and Bagajewicz (2006a, 2006b) is presented. The algorithm is equation-based rather than cutset-based for reasons that are explained below. A specific strategy tailored for problems with high level of specifications is also presented. Three illustrated examples are provided.

2.2. Cutsets-based tree search methods

The optimization model to design minimum-cost sensor network as presented by Bagajewicz (1997) is (in its simplest form) as follows:

$$\left.\begin{array}{ll} Min \sum_{\forall i} c_i q_i & \\ s.t. & \\ \sigma_i(q) \le \sigma_i^* & \forall i \in M_S \\ q_i = 0,1 & \forall i \end{array}\right\} \qquad (2\text{-}1)$$

where q_i , an element of vector q, is a binary variable indicating that a sensor is used to measured variable i, c_i is the cost of such a sensor and M_s represents the set of variables where a performance specification is required (variables of interest or "key" variables).

Realizing that there is no general explicit analytical expression for $\sigma_i(q)$, Bagajewicz (1997) proposed a tree enumeration algorithm using the vector q as a basis (thus enumerating combinations of individual sensors). This algorithm guarantees optimum solution, however, for large scale problems, the computation requirement is so

intensive that the use of this algorithm becomes impractical. While the method of transforming the problem into a LMI-based convex MINLP (Chmielewski et al., 2002) allows the use of the classical branch and bound methods, this can only be applied to precision constraints, but not others and the method still suffers from scaling problem.

Gala and Bagajewicz (2006a) realized that instead of exploring single measurements, specific (and meaningful) subsets could be used. These subsets are called cutsets (taken from graph theory) and as Kretsovalis and Mah (1987) pointed out, they correspond to a set of variables with which a material balance can be written. Gala and Bagajewicz (2006a) proved that by using the union of cutsets in a tree enumeration scheme, one can guarantee optimal solutions and, most importantly, reduce the computational time considerably. The virtue of this algorithm is that only meaningful measurements that contribute to the redundancy or observability of variables of interest (key variables) are added through the use of cutsets. Moreover, when adding a cutset, several measurements may be added at the time instead of only one as in tree enumeration using single measurement. These two properties help cutsets-based tree search methods find feasible nodes in a branch much more rapidly than tree enumeration using single measurement does, especially for middle and large scale problems.

Although tree enumeration using cutsets is suitable for middle scale problem (number of streams ≥ 20), it still has one limitation, which is, for large scale problems (number of streams ≥ 40); the number of cutsets may be too large that the number of nodes in the tree that needs to be explored is prohibitively large, hence computation task is too intensive and computation time can take as long as several days. To overcome this limitation, Gala and Bagajewicz (2006b) proposed the "decomposition of process graph

network" algorithm to reduce computation time. The algorithm still made use of cutsets. However, the process graph is decomposed into sub-graphs so as to reduce the number of cutsets in the candidate lists, hence reducing the size of the tree. There are some "missing" cutsets (these are cutsets spanning over sub-graphs) in the candidate cutsets list when compared with the cutsets list of original process graph. Fortunately, these "missing" cutsets can be found while exploring down the tree using ring-sum operation on cutsets. The tree search procedure is almost the same as the procedure without decomposition except that:

i. The branching and stopping criterion are modified
ii. In each node, ring-sum operations between active cutsets are performed to find the mentioned "missing" cutsets.

The cusets-based tree search coupled with decomposition technique has been shown to be a very efficient method for solving linear large scale problems (Gala and Bagajewicz, 2006b).

We now explore the possibilities and difficulties of using cutsets and tree enumeration with non-linear problems.

2.3. Use of cutsets in the nonlinear case

For linear systems where usually total flowrates are variables of interest, a stream (for which the flowrate is the variable) is connected to not more than two nodes: it is either an input to a unit, an output from a unit or both an input to one unit and an output from another. As a result, a flowrate variable can always be represented by a stream in

process flowsheet and the proper representation of a linear system is the process digraph (Mah, 1990). Hence, for linear systems, the process constraint matrix that represents the overall material balance equations of the process is also the same as the incidence matrix that represents the connectivity of edges (streams) and vertices (nodes). This is not the case for nonlinear systems where a variable may occur in more than two equations and therefore a digraph cannot be used to represent the system. A bipartite graph is, instead, the appropriate structural representation for nonlinear systems (Mah, 1990). The digraph looks the same as the process flowsheet, the nodes are identified with equations and the edges are the flows. In a bipartite graph, two rows of nodes are made, one for the variables and the other for the equations. This is illustrated for the linear system in figure 2.1

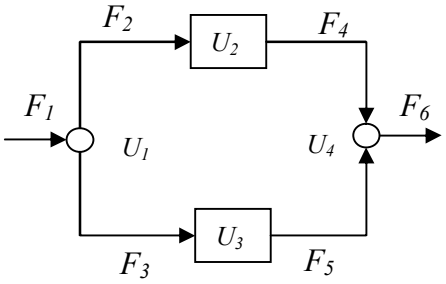


Figure 2.1a. Process digraph

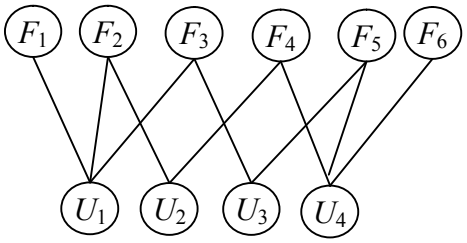


Figure 2.1b. Corresponding bipartite

Figure 2.1 - Process digraph and bipartite graph.

Consider a nonlinear process (figure 2.2) consisting of an adiabatic reactor and an adiabatic flash drum. The nature of the balance equations will determine the nonlinearity of the system. When only total flowrates (F_i) are variables of interest, the system is linear; when the variables concentration (C_i) and temperature (T_i) also need to be estimated, the system is nonlinear.

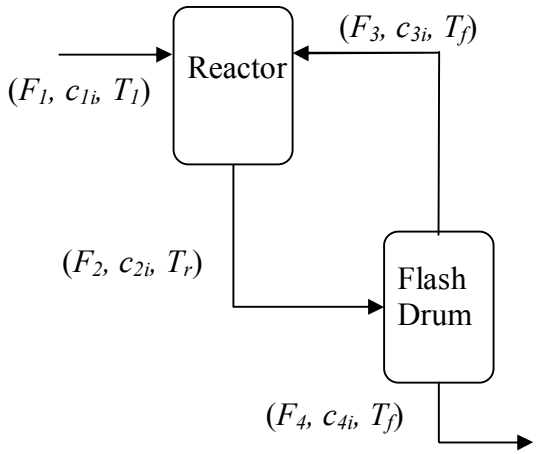


Figure 2.2 - Example of nonlinear systems

The corresponding equations are:

$$g_{R1A}(F_1,F_2,F_3,c_{1A},c_{2A},c_{3A},T_r)=F_1c_{1A}-F_2c_{2A}+F_3c_{3A}+\delta_A(k_oe^{-E/RT_r}c_{2A}^{\alpha})V=0 \qquad (2\text{-}2)$$

$$g_{R1B}(F_1,F_2,F_3,c_{1B},c_{2B},c_{3B},c_{2A},T_r)=F_1c_{1B}-F_2c_{2B}+F_3c_{3B}+\delta_B(k_oe^{-\frac{E}{RT_r}}c_{2A}^{\alpha})V=0 \qquad (2\text{-}3)$$

$$g_{R2A}(F_2,F_3,F_4,c_{2A},c_{3A},c_{4A})=F_2c_{2A}-F_3c_{3A}-F_4c_{4A}=0 \qquad (2\text{-}4)$$

$$g_{R2B}(F_2,F_3,F_4,c_{2B},c_{3B},c_{4B})=F_2c_{2B}-F_3c_{3B}-F_4c_{4B}=0 \qquad (2\text{-}5)$$

$$\begin{aligned} g_{H1}(F_1,F_2,F_3,c_{1A},c_{1B},c_{2A},c_{2B},c_{3A},c_{3B},T_1,T_r,T_f)&=F_1h_1(c_{1A},c_{1B},T_1)-F_2h_2(c_{2A},c_{2B},T_r)\\ &+F_3h_3(c_{3A},c_{3B},T_f)+(-\Delta H_{rxn}(T_r))k_oe^{-E/RT_r}c_{2A}^{\alpha}V=0 \end{aligned} \qquad (2\text{-}6)$$

$$\begin{aligned} g_{H2}(F_2,F_3,F_4,c_{2A},c_{2B},c_{3A},c_{3B},c_{4A},c_{4B},T_r,T_f)&=F_2h_2(c_{2A},c_{2B},T_r)\\ &-F_3h_3(c_{3A},c_{3B},T_f)-F_4h_4(c_{4A},c_{4B},T_f)-Q_{vap}=0 \end{aligned} \qquad (2\text{-}7)$$

$$g_{SA}(F_1,c_{3A},c_{4A},c_{4B},T_f)=c_{3A}/\sum_i c_{3i}=K_i(c_{4A},c_{4B},T_f)c_{4A}/\sum_i c_{4i} \qquad (2\text{-}8)$$

$$g_{SB}(F_1, c_{3A}, c_{4A}, c_{4B}, T_f) = c_{3B} / \sum_i c_{3i} = K_i(c_{4A}, c_{4B}, T_f) c_{4A} / \sum_i c_{4i} \quad (2\text{-}9)$$

Equations (2-2) to (2-5) are component balance equations (we assume that the system contains two components: A and B). Equations (2-6) and (2-7) are energy balance equations in which $h(\bullet)$ are enthalpies. In these equations, we assume that the reactor is adiabatic and that a fixed known amount of heat (Q_{vap}) is removed in the flash. Finally, equations (2-8) and (2-9) represent the vapor liquid equilibrium relationship. The term $k_o e^{-E/RT_r} c_{2A}^{\alpha}$ corresponds to reaction rate r_A (we assume only one irreversible reaction involving one reactant A). The corresponding bipartite graph is shown in figure 2.3

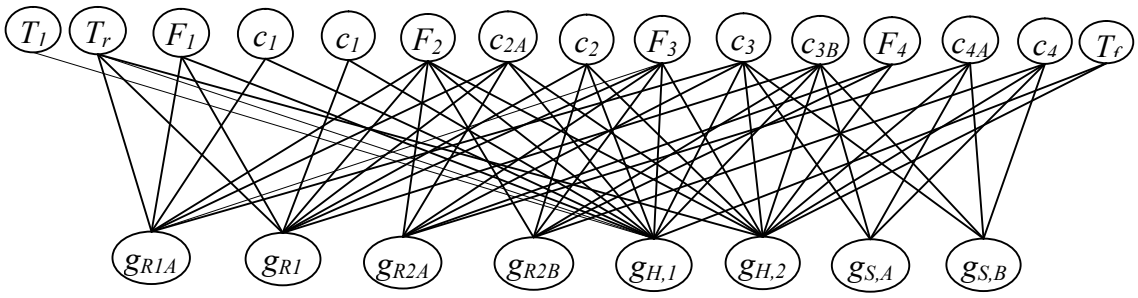


Figure 2.3 - Bipartite graph of the nonlinear process example

Cutsets of digraphs have been used by Gala and Bagajewicz (2006a, 2006b) to design sensor networks for linear systems because: i) each cutset represents a material balance equation involving its elements (streams or variables), which is in turn directly connected to observability or redundancy of variables (Kretsovalis and Mah, 1987), ii) the properties of cutsets and procedures to enumerate all cutsets for linear systems are

well-known. This procedure is illustrated in Figure 2.4 for a linear case and in Figure 2.5 for the corresponding ring sum operation in the context of bipartite graphs.

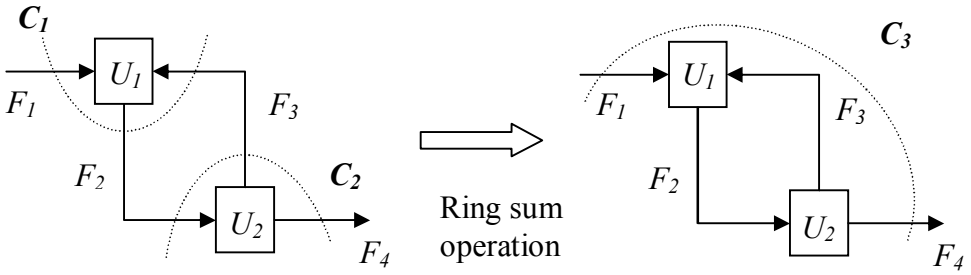


Figure 2.4a Figure 2.4b

Figure 2.4 - Process digraph and ring sum operation on cutsets of the linear system

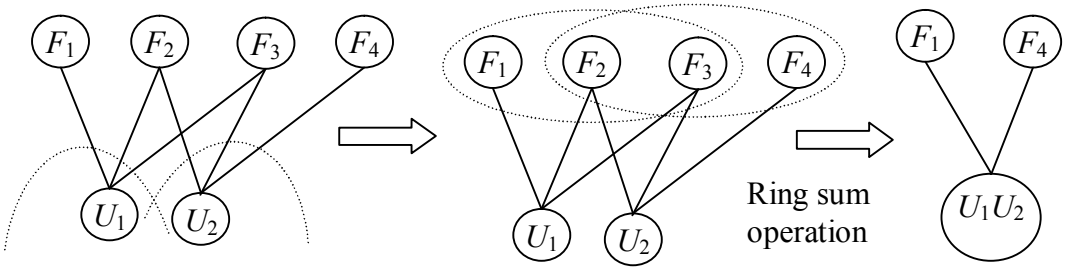


Figure 2.5a Figure 2.5b Figure 2.5c

Figure 2.5 - Bipartite graph and ring sum operation on cutsets of the linear system

The ring sum operation does not always apply properly to nonlinear cases. Indeed, consider equations g_{R1A} and g_{R2A}, which share two terms: $F_2 c_{2A}$ and $F_3 c_{3A}$ or four variables (F_2, c_{2A}, F_3 and c_{3A}). Substitution of $F_4 c_{4A} = F_2 c_{2A} - F_3 c_{3A}$ obtained from g_{R2A} into g_{R1A} renders the following equation:

$$F_1 c_{1A} - F_4 c_{4A} + \delta_A (k_o e^{-E/RT_r} c_{2A}^{\alpha}) V = 0 \qquad (2\text{-}10)$$

The ring sum operation results in these four variables being eliminated as seen in Figure 2-6.

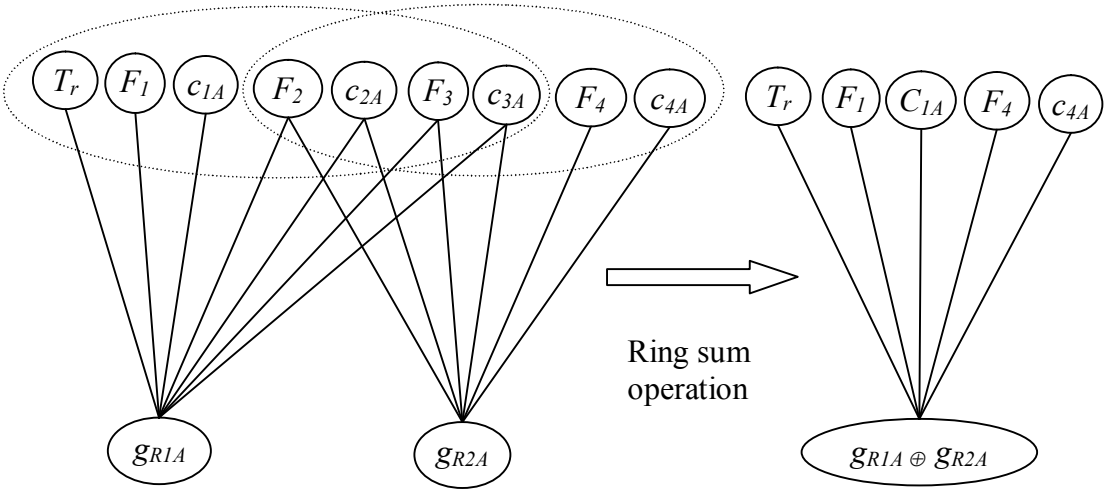


Figure 2.6 - Ring sum operation on cutsets of bipartite graph

It can be seen that the ring sum of these two cutsets leads to the elimination of all four variables that are in the intersection, including $c_{2A.}$. But, unless the reaction is zero order, $c_{2A.}$ is still present in the merged equation (2-10). Thus, the variable substitution operation generates in this case a result that is different from the ring sum operation. It is assumed here that one variable can be explicitly expressed as a function of others in one equation and formally substituted in a second equation, thus variable substitution can take place.

Thus, for nonlinear systems one needs to depart from using ring sum of cutsets strictly and look for an equivalent procedure. Such procedure would be equivalent of finding an alternative operation to the ring sum. Indeed, as stated above, cutsets are equations in the linear case, and since the ring sum is equivalent to taking two equations and generating a third, thus eliminating one (or more) variables.

The method proposed in this work is based on the same concept, but linearized equations are used instead of cutsets

2.4. Automatic generation of all equations

We now prove that the generation of new equations can be done automatically using Gaussian elimination on the linearized equations. We do this by proving the following claims first

Claim 1 (Necessary condition):

Consider two nonlinear equations

$$f_1(x_1, x_{2,} \ldots x_k, x_{k+1},.. x_m)=0 \tag{2-11}$$

$$f_2(x_k, x_{k+1},.. x_m, x_{m+1},.. x_n)=0 \tag{2-12}$$

If equation merging or variable substitution is performed targeting variable x_k, and a set of other variables , namely x_{k+1}, **…** x_m , which are considered consecutive without loss of generality, are also eliminated, then:

a) The partial derivatives with respect to these eliminated variables in both equations are equal, that is:

$$\frac{\partial f_1}{\partial x_t} = \frac{\partial f_2}{\partial x_t} \qquad \forall t = k+1,\ldots,m \tag{2-13}$$

b) Variable substitution in the linearized system also eliminates the same variables.

Proof:

Assume first (without loss of generality) that f_1 and f_2 can be expressed in terms of x_k as follows:

$$f_1(x_1, x_{2,\ldots} x_k, x_{k+1},.. x_m)= \Sigma_i r_{1i}(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m)+ x_k=0 \quad (2\text{-}14)$$

$$f_2(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m) =\Sigma_i r_{2i}(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m) + x_k = 0 \quad (2\text{-}15)$$

In other words, x_k can be isolated. Then, substitution of x_k in equation f_2 renders:

$$f_{12}(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m, x_{m+1},.. x_n)=$$

$$=\Sigma_i r_{1i}(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m)- \Sigma_i r_{2i}(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m)=0 \quad (2\text{-}16)$$

For variables to be eliminated, then pairs of r_{1i} and r_{2i} need to be exactly the same expressions with the same combinations of variables so that they cancel. Without loss of generality, assume now these variables are in both equations in the terms r_{11} and r_{21} only, that is

$$r_{11}(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m)=r_{21}(x_1, x_{2,\ldots} x_{k-1}, x_{k+1},.. x_m)=0 \quad (2\text{-}17)$$

Because (and only because) these terms disappear from the final version of (2-16), we can say that they ARE the same expression, which in turn, allows us to say the derivatives are also formally the same expression. This proves part a)

Proving part b) is now easy. Indeed, linearizing f_1 and f_2, one obtains:

$$\sum_{j \neq k} a_{1j} x_j + x_k = 0 \tag{2-18}$$

$$\sum_{j \neq k} a_{2j} x_j + x_k = 0 \tag{2-19}$$

where $a_{ki} = \dfrac{\partial\left[\sum_i r_k\right]}{\partial x_i}$. Substituting x_k obtained from (2-18) into (2-19), one obtains.

$$\sum_{j \neq k} (a_{2j} - a_{1j}) x_j = 0 \tag{2-20}$$

Thus, if any variable is to be eliminated, say x_{k+1}, then $a_{2k+1} = a_{1k+1}$, which is the same as (2-13)

Q.E.D.

Claim 2 (Sufficient condition):

Consider the linearized equations (2-18) and (2-19). If Gaussian elimination is performed between these two equations, and aside from variable x_k, the set of other variables, namely x_{k+1} , … x_m ,are also eliminated, then, the same variables will be

eliminated if equation merging or variable substitution is performed on equations (2-14) and (2-15), provided that equation (2-13) holds symbolically (i.e. they are the same symbolic expression), not only numerically.

Proof:

The proof is straightforward: If a variable is eliminated, say x_{k+1}, then $a_{2k+1} = a_{1k+1}$, which is the same as (2-13) numerically. We only need to make sure that (17) holds, and this is true only if (2-13) or (2-17) hold true symbolically.

Q.E.D.

Corollary: Both claims are valid if explicit expressions in terms of x_k cannot be obtained.

Proof:

In this case, we would have

$$f_1(x_1, x_{2, \ldots} x_k, x_{k+1}, .. x_m)=0 \rightarrow \Sigma_i r_{1i}(x_1, x_{2, \ldots} x_{k-1}, x_{k+1}, .. x_m)$$

$$+ \ z(x_1, x_{2, \ldots} x_{k,}, x_{k-1}, x_{k+1}, .. x_m) =0 \qquad (2\text{-}21)$$

Thus, if x_k is to be formally eliminated from both equations then the term $z(x_1, x_{2, \ldots} x_{k,}, x_{k-1}, x_{k+1}, .. x_m)$, needs to exactly appear in both equations. With this the proof can be continued exactly as before.

Q.E.D.

We can now present the procedure to find and enumerate all equations of the problem:

i. Linearize the process model to arrive at the linearized model written in the matrix form $\boldsymbol{Ax = b}$, where $\boldsymbol{A}$ is the process constraint matrix.

ii. For all pairs of equations, perform the Gaussian elimination operation to find new equations and put them at the end of the list of equations. If any pair of equations has more than one common variable, all possible new equations are to be found by choosing different variables to eliminate. Note that only combinations of equations with at least one common variable are performed.

iii. If a resulting new equation is the same as any equation already in the list, disregard that equation.

Because all combinations of original equations are considered, this procedure guarantees that all possible new equations are found. Combinations between a new equation and an original equation or between a new equation and another new equation are not necessary because they can be obtained by combining original equations.

There are two steps in the design procedure that call for the linearization of the nonlinear equations around nominal operating condition:

i. Finding new equations from pairs of "original" equations using the variable substitution or equivalently the Gaussian elimination operation

ii. Solving the associated data reconciliation problem, where an analytical solution is obtainable only when the model is linear or linearized, in order to check whether the candidate sensor network satisfies the design specifications.

Changing operating conditions would not cause any effect in the equations generating step (the resulting equations are unchanged, only the coefficients in the equations change); but it may have an effect in the step of checking design specifications: for example, a feasible solution can become infeasible if the operating windows moves to

another region. As a result, different regions of operating conditions may lead to different optimal solutions and the obtained optimal solution is guaranteed to be valid only within the current operating windows. Designing optimal sensor network that is valid for a wide range of operating conditions requires a new problem formulation and a tailored computational method, which is beyond the scope of this work. However, in the examples section, a brief discussion of the sensitivity of the solution as the process variables fluctuate around their nominal values is provided.

2.5. Equation-based tree search algorithm

The tree enumeration algorithm using equations for the design of nonlinear sensor networks is the same as the one used by Gala and Bagajewicz (2006a), except that instead of using cutsets, we use equations. The procedure is briefly described below:

1. Find all the equations of the problem using the procedure described above.
2. Pick up only the equations containing key variables (called candidate equations) because other equations (not containing key variables) do not contribute to the observability or redundancy of key variables.
3. Sort these candidate equations in ascending order of their cost (the cost of an equation is equal to sum of the costs of the sensors used to measure variables contained in that equation).
4. Start with the root node with no equation being added i.e. $\boldsymbol{e} = \{0, 0, 0\ldots\}$, trivially infeasible.

5. Using the branch first rule, develop each branch by making one element of '***e***' active and adding one candidate equation at a time which is chosen from the remaining equations using a branching criterion.
6. While performing the branching criteria, if any set of equations has already been evaluated in previous nodes, that node is not continued. This occurs frequently because one set of measurements can be a result of the union of different combinations of equations.
7. This is continued until the stopping criterion is met. In such case, the algorithm backs up two levels and develops the next branch.

Branching Criterion

While exploring the tree from one node to the other, either going down the tree or exploring the sister node, the newly added equation is chosen in such a way that the cost obtained by its union with the existing active equations is minimal.

Stopping Criterion

Because adding an equation always increases the cost, whenever a feasible node is found (one that satisfies all the constraints of the problem), the tree is not explored further down nor any sister branch.

To overcome the computational limitations of the above procedure, Gala and Bagajewicz (2006b) proposed the "decomposition of process graph network" algorithm to reduce computation time. The algorithm still makes use of cutsets to find the optimum sensor network but the process graph is decomposed into sub-graphs so as to reduce the

number of original cutsets, hence reducing the size of the tree. While exploring the tree, if cutsets from different subgraphs are used in a node of the tree, the ring sum operation on those cutsets is performed to generate all the cutsets that are missing and then the union operation among the resulting cutsets plus the originals is performed. These are briefly described below:

*Operations in a node: s*uppose that the current node contain cutsets from three different sub-graphs C_{A1}, C_{B1} *and* C_{C1}, then:

- All the possible combinations of ring sum and union operation of cutsets are found: $C_{A1} \times C_{B1} \, U \, C_{C1}$; $C_{A1} \, x \, C_{C1} \, U \, C_{B1}$; $C_{A1} \, U \, C_{B1} \times C_{C1}$; $C_{A1} \times C_{B1} \times C_{C1}$ *w*here *U*: union operation; *x*: ring sum operation
- Checking the feasibility of all the resulting solutions.

Stopping criterion : stop if

(Current feasible node cost – Connecting streams cost in this node

+ Min Instrument cost) $\geq$ *Best feasible node cost found*

For nonlinear systems, the same technique as described in Gala and Bagajewicz (2006b) is used with some modifications (the same branching and stopping criterion are used). These modifications are:

i. The ring sum operation on cutsets is replaced by our variable elimination operation on equations

ii. The decomposition is performed on the bipartite graph of the nonlinear system instead of the process digraph of linear system.

2.6. Inverted tree strategy

If a large number of key variables (those whose values are of interest) and/or good level of precision and residual precision is required, a large number of sensors needs to be used to meet the requirements. This is accentuated if error detectability and resilience requirements are added. In such design cases, the "forward" tree search methods (the Equation-based method presented above as well as the as the tree search methods presented in Bagajewicz (1997) and Gala and Bagajewicz (2006a, 2006b)) exhibit the following problems:

i. The number of active elements in feasible nodes is large, that is, the search procedure needs to explore deeper down into the tree before it finds a feasible solution

ii. The number of nodes explored is large and the computational time is long. Moreover, for the equation-based tree search methods, a large number of key variables leads to a large number of equations that contain at least one key variable (i.e. large tree size)

iii. The number of feasible nodes is low and they are all located deep in the tree towards the end of it.

To ameliorate this shortcoming (having to explore the tree very deep), an inverted tree search method is proposed. The idea behind this method is to explore the tree in the

reverse direction, that is, it to start with a root node containing *all* sensors and continue *removing* sensors when going up the tree until an infeasible node is found (stopping criterion). Because the level of the feasible nodes explored by the tree search is low, the number of nodes explored is reduced, which results in a shorter computation time. The same thing is also argued if equations are used instead of sensors (measurements).

There are two "forward" tree search methods as discussed above: one uses list of equations / cutsets and the other uses list of measurements. We investigate the reverse versions of these two tree search methods. The inverted tree search using list of measurements is described next:

- Start tree search with the root node containing all sensors, an automatically feasible node.
- Removing sensors out of root node by using the tree enumeration algorithm and a branching criterion, which is to remove the most expensive sensor among all sensors in the current node so that the sensors cost is minimized.
- Always start developing branches with feasible nodes containing large number of sensors. The number of sensors in nodes decreases and cost is reduced when going up the tree.
- Stop going up (stop removing sensors) when the current node becomes infeasible (Stopping Criterion). If keeping going up, the cost is reduced but the node is infeasible.

An inverted tree search using equations (root node containing *all* equations and continue *removing* equations when going up the tree) is much less efficient, in principle, because of two problems that lead to longer computation time:

(i) The number of equations is large (much larger than the number of variables), hence the tree depth is larger

(ii) The search needs to explore further up into the tree before it finds an infeasible node and stop. The reason for this is that the union of only a small number of equations can result in the sensor network containing *all* sensors. Thus, if only a small number of equations is removed out of the root node, the resulting sensor network (obtained as union of equations in current node) usually still contains *all* sensors. As a result, a significant number of equations needs to be removed before the number of sensors in resulting sensor network is reduced and the node becomes infeasible.

In summary, three methods are proposed to solve the nonlinear sensor network design problem: the inverted tree search using list of measurements (referred to by the short name of "Inverted All Variables" method) and the "forward" equations-based tree search methods that has two versions: i) without decomposition (referred to as "All Equations" method), ii) with decomposition (referred to as "Decomposed Equations" method). The characteristics of these three methods together with the forward tree search method using list of measurements (Bagajewicz, 1997 referred to as "All Variables" method), which is used to validate the optimality of obtained solutions, are summarized in Table 2.1:

Table 2.1. Summary of proposed tree search methods

Method	**Search Strategy**	**Base unit**	**Decomposed**
All Variables	Forward	Measurements	No
All Equations	Forward	Equations	No
Decomposed Equations	Forward	Equations	Yes
Inverted All Variables	Reverse	Measurements	No

2.7. Illustrated examples

The proposed Equations-based methods and the "Inverted All Variables" method guarantee optimality; their computational efficiency is tested using the following examples. The "All Variables" method (Bagajewicz, 1997) is used to validate the optimality of the solutions obtained by the proposed methods. The proposed algorithms were implemented in a Fortran program running on a 2.8 GHz Intel Pentium, 1028 MB RAM PC computer.

Three examples are considered: a CSTR process (small scale problem), a mineral flotation process (middle scale problem) and the TE process (large scale problem). Based on our experience, we qualitatively classify three types of problems that can be solved by our sensor network design program using the number of variables involved: i) small scale: 1-18 variables, ii) middle scale: 19-39 variables and iii) large scale: 40 variables and above. As usual, these are heuristic observations and although the number of variables is indicative of size, as we shall see below, the tightness of the specifications may make the same size problem to be solved much faster/slower.

2.7.1. Example 2.1: CSTR process

Consider the CSTR process which was introduced by Bhushan[6] and is given in Figure 2.7

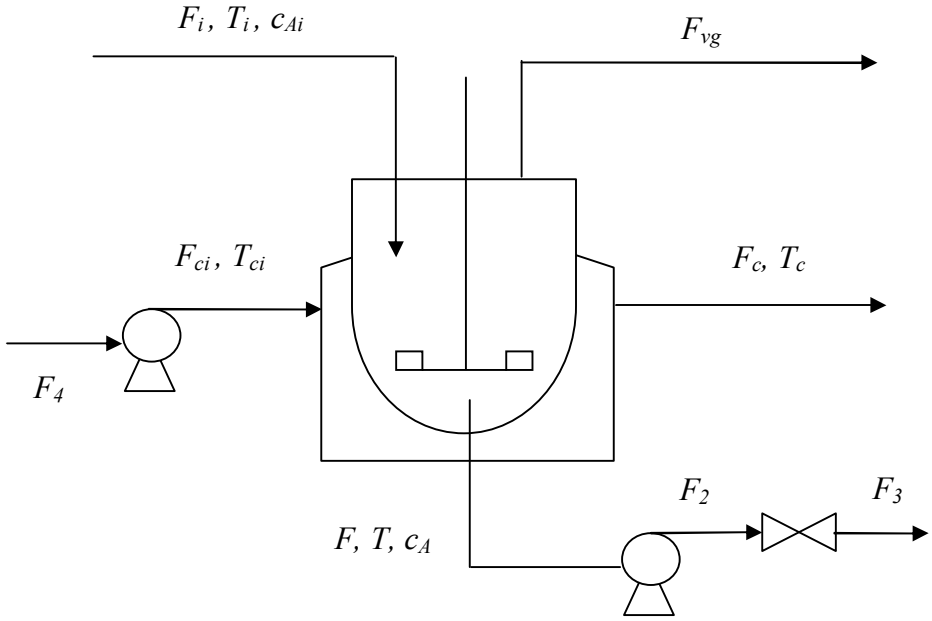


Figure 2.7 - The CSTR problem

The variables of interest are [F_i, c_{Ai}, c_A, T, T_i, T_c, F_c, T_{ci}, F_{vg}, F, F_2, F_3, F_4]. There are five equations (including both mass and energy balances) written around the reactor and its jacket (equations 2-22 to 2-26), three mass balance equations written for pumps and valves (equations 2-27, 2-28, 2-29)

$$e_1(F_i, c_{Ai}, c_A, T) = \frac{F_i}{V}(c_{Ai} - c_A) - c_d c_A k_0 e^{-\frac{E}{RT}} = 0 \tag{2-22}$$

$$e_2(F_i, c_A, T, T_i, T_c) = \frac{F_i}{V}(T_i - T) + \frac{c_d c_A k_0 e^{-\frac{E}{RT}}(-\Delta H)}{\rho C_p} - \frac{UA(T - T_c)}{V \rho C_p} = 0 \tag{2-23}$$

$$e_3(T,T_c,F_c,T_{ci}) = \frac{F_c}{V_j}(T_{ci}-T_c) + \frac{UA(T-T_c)}{V_j\rho_j C_{pj}} = 0 \tag{2-24}$$

$$e_4(c_A,T,F_{vg}) = c_d c_A k_0 e^{-\frac{E}{RT}} V - F_{vg} = 0 \tag{2-25}$$

$$e_5(F_i,F) = F_i - F = 0 \tag{2-26}$$

$$e_6(F_2,F_3) = F_3 - F_2 = 0 \tag{2-27}$$

$$e_7(F,F_2) = F_2 - F = 0 \tag{2-28}$$

$$e_8(F_c,F_4) = F_4 - F_c = 0 \tag{2-29}$$

The nominal operation conditions are given in Table 2.2 (value of flowrate is given in ft^3/hr, temperature : ^oR, concentration: $lb.mole/ft^3$).

Table 2.2 - Nominal operating condition for the CSTR example

Variable	F_i	F_c	F_{vg}	F	F_2	F_3	F_4
Value	40	56.626	10.614	40	40	40	56.626
Variable	c_{Ai}	c_A	T	T_i	T_c	T_{ci}	
Value	0.5	0.2345	600	530	590.51	530	

The linearized model matrix is:

$$A_{nl} = \begin{array}{c} \begin{matrix} F_i & c_{Ai} & c_A & T & T_i & T_c & F_c & T_{ci} & F_{vg} & F & F_2 & F_3 & F_4 \end{matrix} \\ \begin{pmatrix} -0.00531 & -0.8333 & 1.7763 & 0.00923 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.4583 & 0 & -754.4 & 5.9503 & -0.8333 & -125 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -93.8067 & 0 & 108.5 & 15.7169 & -14.708 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -45.2612 & -0.443 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \\ e_8 \end{matrix} \end{array}$$

The process of generation of new equations is now illustrated. Take for example the original equations e_1 and e_2, which have three common variables F_i, c_A, T. Three new equations result from the three possible Gaussian elimination operations: $e_9 = \{c_{Ai}, c_A, T, T_i, T_c\}$ obtained by eliminating the common variable F_i, $e_{10} = \{F_i, c_{Ai}, T, T_i, T_c\}$ obtained by eliminating the common variable c_A, and $e_{11} = \{F_i, c_{Ai}, c_A, T_i, T_c\}$ obtained by eliminating the common variable T. These three new equations are shown in Table 2.3.

Table 2.3 - New equations for the CSTR example obtained from the combination of e_1 and e_2

(a) Variables involved

Equation	Variables involved
e_9	$\{c_{Ai}, c_A, T, T_i, T_c\}$
e_{10}	$\{F_i, c_{Ai}, T, T_i, T_c\}$
e_{11}	$\{F_i, c_{Ai}, c_A, T_i, T_c\}$

(b) Expressions

Equation	Expressions
e_9	$\frac{c_d c_A k_0 e^{-E/RT}(T_i - T)}{(c_{Ai} - c_A)} + \frac{c_d c_A k_0 e^{-E/RT}(-\Delta H)}{\rho C_p} - \frac{UA(T - T_c)}{V \rho C_p} = 0$
e_{10}	$\frac{F_i}{V}(T_i - T) + \frac{c_d k_0 e^{-E/RT}(-\Delta H) F_i c_{Ai}}{\rho C_p (F_i + V c_d k_0 e^{-E/RT})} - \frac{UA(T - T_c)}{V \rho C_p} = 0$
e_{11}	$\frac{F_i}{V}(T_i - T) + \frac{F_i}{V}\frac{(c_{Ai} - c_A)(-\Delta H)}{\rho C_p} - \frac{UA}{V \rho C_p}\left(- \frac{E}{R \ln(\frac{F_i}{V}\frac{(c_{Ai} - c_A)}{c_d c_A k_0}} - T_c \right) = 0$

(c) Linearized Expressions

Equation	Linearized expressions
e_9	$-0.833\, c_{Ai} - 0.972\, c_A, + 0.031\, T - 0.003\, T_i - 0.456 T_c$
e_{10}	$-0.002\, F_i - 0.83 c_{Ai} + 0.0232\, T - 0.002\, T_i - 0.294\, T_c$
e_{11}	$-0.0076\, F_i - 0.833\, c_{Ai} + 2.946\, c_A + 0.0013\, T_i + 0.194\, T_c$

It is clear from this example that the number of equations in nonlinear systems is much larger than the number of cutsets in linear systems of the same flowsheet size, not only because more equations are written for each unit, but also because any combination of cutsets results in just one new cutset in linear systems, while several new equations can be obtained from a combination of equations in nonlinear systems as illustrated above.

The case studies for this process using the proposed algorithms are shown next. For the Decomposed Equations method, a single decomposition is performed, that is, the graph is decomposed into two subgraphs corresponding to two subset constraint matrices, one contains rows 1 to 4 and one contains rows 5 to 8 (the cutting is done between row 4 and 5). This is shown in Figure 2.8 where the original bipartite graph is decomposed into two sub-graphs (A & B), each contains 4 nodes (4 original equations). The total number of equations obtained from all eight original equations is 88, while the number of equations obtained from the first four original equations (i.e. sub-graph A) is 28 and from the last four original equations (i.e. sub-graph B) is 6. Thus, in the decomposition method, the total number of equations (i.e. the tree size) is 34 (=28 plus 6) instead of 88. The rest of the equations are found while exploring the tree by using variable elimination operation on any pair of equations originated from any two different sub-graphs.

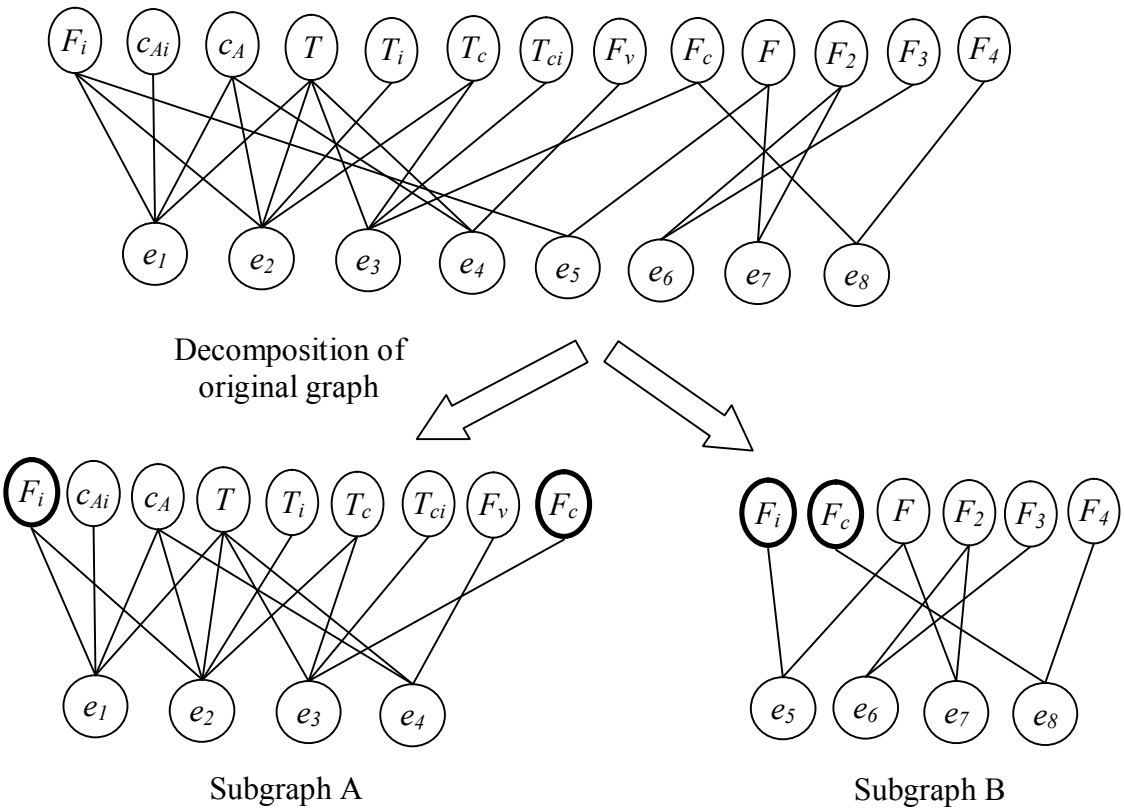


Figure 2.8 - Bipartite graph of the CSTR example with decomposition (common variables are shown in thicker circles)

The costs of sensors that measure variables $V_1, V_2, \ldots, V_{13}$ are 100, 270, 300, 50, 55, 60, 105, 45, 85, 90, 95, 80, 82 respectively. The sensor precisions are 1% (for all sensors).

Three design cases are considered corresponding to three levels of design specification: low specification (CSTR1), moderate (CSTR2) and high specification (CSTR3), which are shown in table 2.4. In table 2.4 as well as similar tables in the other two examples, rows 2 to 6 show detail of specifications for each design case; the stated threshold values (rows 5, 6) are applied to all the key variables listed in row 3, any key

variable with specific threshold will be mentioned separately. Rows 7 & 8 show the optimal solution obtained by the four methods, which include two types of information: the variables to measure (row 7) and the total sensor cost (row 8).

Table 2.4 - Design case studies for the CSTR example

Case Study	CSTR1 Low Spec.	CSTR2 Moderate Spec.	CSTR3 High Spec
No. of key variables	3	4	8
Key variables	c_A, T, F	c_A, T, T_{ci}, F	c_{Ai},c_A,T, T_c, F_c, $T_{ci,}$ F, F_3
Requirement	Observability	Redundancy	Observability
Precision thresholds	0.95%	1.5%	1.5%
Residual precision thresholds		2.5%	2.5%
Measured variables	c_{Ai} ,c_A ,F_{vg} ,F_3	c_{Ai},c_A,T,T_i,$T_{ci,}$$F$,$F_3$,$F_4$	c_{Ai},c_A, T, T_i, T_c, F_c, $T_{ci,}$ F, F_3 and F_4
Sensors cost	735	972	1137

The computation time and the number of nodes explored of the four methods are shown in Table 2.5.

Table 2.5 - Results for the CSTR example

Case Study		CSTR1 (Low Spec.)	CSTR2 (Moderate Spec.)	CSTR3 (High Spec.)
All-Equations (no decomposition)	Total computation time	8 sec.	26 sec.	27 sec.
	Computation time to generate equations	1 sec.	1 sec.	1 sec.
	Number of equations generated	85	87	87
	Number of nodes explored	1611	4,695	7,640
Decomposed Equations (two subgraphs)	Total computation time	4 sec.	8 sec.	14 sec.
	Computation time to generate equations	< 1 sec.	< 1 sec.	<1 sec.
	Number of equations generated	33	33	34
	Number of nodes explored	1,737	2,914	5,456
All Variables	Computation time	3 sec.	5 sec.	6 sec.
	Number of nodes explored	4,653	7,088	8,102
Inverted All Variables	Computation time	2 sec.	2 sec.	1 sec.
	Number of nodes explored	2,520	682	117

When comparing the computation times, the equation-based method is sometimes faster than that of the All Variables method because of the smaller number of nodes explored and sometimes slower because the equation-based method initially requires finding all the equations of the system and requires checking the branching criterion in every node of the tree. The computation time of the Decomposed Equations method is shorter than that of the All Equations method as expected.

The method with shortest computational time in this example is the "Inverted All Variables" method, which can be seen to be faster than its "forward" counterpart, the "All Variables" method, especially for the high specification design case (CSTR3).

Because in this example the equation-based methods do not offer much advantage in terms of reduced number of nodes explored and requires extra time for performing branching at any node, the real test for the advantage of the equation-based methods has to come from applying them to a larger example.

The sensitivity of the obtained solution is now briefly discussed before the next example is presented. As we linearize the nonlinear equations around the steady-state values of the process variables, the obtained solution is valid only within the current operating window. Take for example the design case CSTR1, if the steady-state values of the process variables (except the reaction temperature) change within 40% the nominal values, the precision values of the estimators change up to 95% of the precision values in the base case (corresponding to the nominal values) but still satisfy the design specifications, that is, the obtained solution is valid when the fluctuation is less than 40% of the nominal value. If the fluctuation is more than 40% of the nominal values (except the reaction temperature), the new optimal solution is to measure 7 variables with cost of 737. The obtained solution is more sensitive to the variation of the reaction temperature, an important variable whose fluctuation significantly affects all other variables in the process. The variation range of the reaction temperature within which the obtained solution is valid is 26% of the nominal value. These results point out that the solution is valid within the normal variation range of process variables (30% of the nominal values) but it is not valid for a wider range of process operating conditions.

2.7.2. Example 2.2: Mineral flotation process

Consider a middle scale process, the mineral flotation process introduced by Smith and Ichiyen (1973) shown in Figure 2.9.

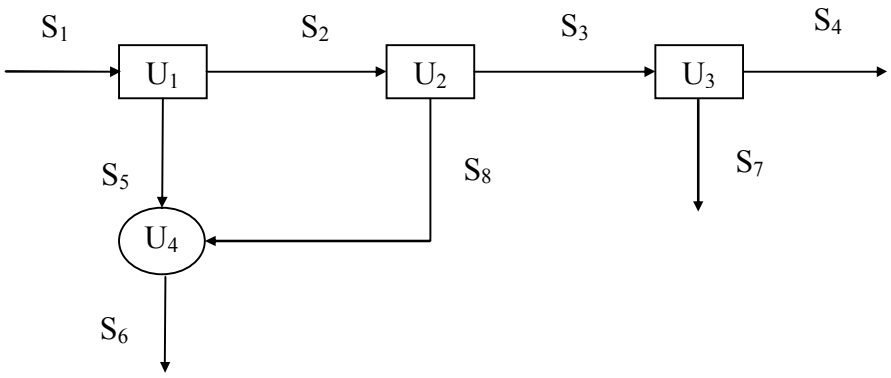


Figure 2.9 - The mineral flotation process

The process consists of three flotation cells (separators) and a mixer. Each stream consists of two minerals, copper (component A) and zinc (component B), in addition to gangue material. The total flowrate F, the composition of copper C_A and zinc C_B of all streams are variables of interest, so the total number of variables under consideration is 24 (8 flowrates and 16 compositions). Let us assume that each variable can be measured separately by a sensor. The process model consists of three types of material balance equations: the total flowrate balance, the copper component (A) flowrate balance and the zinc component (B) component balance. These three types of balance equations are written for unit 1 next. Balance equations for other units can be written in the same fashion:

$$F_1 - F_2 - F_5 = 0 \tag{2-30}$$

$$F_1 C_{1A} - F_2 C_{2A} - F_5 C_{5A} = 0 \tag{2-31}$$

$$F_1 C_{1B} - F_2 C_{2B} - F_5 C_{5B} = 0 \tag{2-32}$$

The total number of original balance equations is 12 (3 per unit). The component balance equations are nonlinear, hence the system is nonlinear (it is bilinear system). The nominal operating condition is given in table 2.6 (taken from Narasimhan and Jordache, 2000):

Table 2.6 - Nominal operation condition for mineral flotation process

Streams	1	2	3	4	5	6	7	8
F_i (kmol/hr)	100	92.67	91.57	84.48	7.33	8.43	7.09	1.1
C_{iA} (% mol)	0.019	0.0045	0.0013	0.001	0.2027	0.2116	0.0051	0.2713
C_{iB} (% mol)	0.0456	0.0437	0.0442	0.0041	0.069	0.0495	0.5227	0.001

When the balance equations are linearized, the process model can be written in the following form $\boldsymbol{Ax} = \boldsymbol{b}$, where

$$A = \begin{array}{c} \begin{array}{cccccccccccccccccccccc} F_1 & F_2 & F_3 & F_4 & F_5 & F_6 & F_7 & F_8 & C_{1A} & C_{1B} & C_{2A} & C_{2B} & C_{3A} & C_{3B} & C_{4A} & C_{4B} & C_{5A} & C_{5B} & C_{6A} & C_{6B} & C_{7A} & C_{7B} & C_{8A} & C_{8B} \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.02 & -0.005 & 0 & 0 & -0.203 & 0 & 0 & 0 & 100 & 0 & -92.67 & 0 & 0 & 0 & 0 & 0 & -7.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.005 & -0.001 & 0 & 0 & 0 & 0 & -0.271 & 0 & 0 & 92.67 & 0 & -91.57 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.1 & 0 \\ 0 & 0 & 0.001 & -0.001 & 0 & 0 & -0.005 & 0 & 0 & 0 & 0 & 0 & 91.57 & 0 & -84.48 & 0 & 0 & 0 & 0 & 0 & -7.09 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.203 & -0.212 & 0 & 0.271 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.33 & 0 & -8.43 & 0 & 0 & 0 & 1.1 & 0 \\ 0.05 & -0.044 & 0 & 0 & -0.069 & 0 & 0 & 0 & 0 & 100 & 0 & -92.67 & 0 & 0 & 0 & 0 & 0 & -7.33 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.044 & -0.044 & 0 & 0 & 0 & 0 & -0.001 & 0 & 0 & 0 & 92.67 & 0 & -91.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.1 \\ 0 & 0 & 0.044 & -0.004 & 0 & 0 & -0.523 & 0 & 0 & 0 & 0 & 0 & 0 & 91.6 & 0 & -84.5 & 0 & 0 & 0 & 0 & -7.09 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.069 & -0.05 & 0 & 0.001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.33 & 0 & -8.43 & 0 & 0 & 0 & 1.1 \end{pmatrix} \end{array}$$

The first four rows in the constraint matrix A corresponds to the total flow balances, row 5 to row 8 represent for copper component balances and the rest corresponds to zinc component balances. All sensor precisions are 2%. The sensor costs are given in table 2.7.

Table 2.7 - Sensor costs for mineral flotation process example

Streams	1	2	3	4	5	6	7	8
F_i	50	55	45	60	40	48	52	58
C_{iA}	300	310	240	260	250	360	320	335
C_{iB}	290	350	330	340	280	270	295	275

Three design cases at three different levels of design specification are considered: low specification (MFP1), moderate (MFP2) and high specification (MFP3). They are shown in table 2.8.

Table 2.8 - Design case studies for the mineral flotation process example

Case Study	MFP1 Low Spec.	MFP2 Moderate Spec.	MFP3 High Spec
No. of key variables	4	4	12
Key variables	F_1, C_{1A}, F_7 and C_{7B}	F_1, C_{1A}, F_7 and C_{7B}	F_1, F_4, F_6, C_{1A}, C_{1B}, F_7, C_{4A}, C_{4B}, C_{6A}, C_{6B}, C_{7A}, C_{7B}
Requirement	Observability	Redundancy	Observability
Precision thresholds	1.5% (F_1, C_{1A}) 2% (F_7 , C_{7B})	1.5% (F_1, C_{1A}) 2% (F_7 , C_{7B})	1.5% (F_1, F_4, F_6, C_{1A}, C_{1B}) 2% (F_7, C_{4A}, C_{4B}, C_{6A}, C_{6B}, C_{7A}, C_{7B})
Residual precision thresholds		5% for F_1, F_7 only	
Measured variables	F_1, F_3, F_5, F_6, F_7, F_8, C_{1A}, C_{2A}, C_{5A}, C_{7B}	F_1, F_3, F_5, F_6, F_7, F_8, C_{1A}, C_{2A}, C_{3B}, C_{4B}, C_{5A} and C_{7B}	F_1, F_3, F_5, F_6, F_7, F_8, C_{1A}, C_{2A}, C_{3B}, C_{4A}, C_{4B}, C_{5A}, C_{6B}, C_{7A} and C_{7B}
Sensors cost	1448	2118	2968

In all case studies, the decomposition is made by operating directly on the constraint matrix, that is, the constraint matrix is "cut" into three subset matrices: {row 1 to row 4}, {row 5 to row 8} and {row 9 to row 12}. The submatrices for these systems are indicated by dotted rectangles in matrix A. The common variables between the subset matrices (called connecting streams in the case of linear systems) are the eight flowrate variables: from F_1 to F_8. The computation time and number of nodes explored are shown in table 2.9.

$$A = \begin{array}{c} \begin{matrix} F_1 & F_2 & F_3 & F_4 & F_5 & F_6 & F_7 & F_8 & C_{1A} & C_{1B} & C_{2A} & C_{2B} & C_{3A} & C_{3B} & C_{4A} & C_{4B} & C_{5A} & C_{5B} & C_{6A} & C_{6B} & C_{7A} & C_{7B} & C_{8A} & C_{8B} \end{matrix} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.02 & -0.005 & 0 & 0 & -0.203 & 0 & 0 & 0 & 100 & 0 & -92.67 & 0 & 0 & 0 & 0 & 0 & -7.33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.005 & -0.001 & 0 & 0 & 0 & 0 & -0.271 & 0 & 0 & 92.67 & 0 & -91.57 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.1 & 0 \\ 0 & 0 & 0.001 & -0.001 & 0 & 0 & -0.005 & 0 & 0 & 0 & 0 & 0 & 91.57 & 0 & -84.48 & 0 & 0 & 0 & 0 & 0 & -7.09 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.203 & -0.212 & 0 & 0.271 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.33 & 0 & -8.43 & 0 & 0 & 0 & 1.1 & 0 \\ 0.05 & -0.044 & 0 & 0 & -0.069 & 0 & 0 & 0 & 0 & 100 & 0 & -92.67 & 0 & 0 & 0 & 0 & 0 & -7.33 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.044 & -0.044 & 0 & 0 & 0 & 0 & -0.001 & 0 & 0 & 0 & 92.67 & 0 & -91.6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1.1 \\ 0 & 0 & 0.044 & -0.004 & 0 & 0 & -0.523 & 0 & 0 & 0 & 0 & 0 & 0 & 91.6 & 0 & -84.5 & 0 & 0 & 0 & 0 & 0 & -7.09 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.069 & -0.05 & 0 & 0.001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.33 & 0 & -8.43 & 0 & 0 & 0 & 1.1 \end{pmatrix} \end{array}$$

Table 2.9 - Results for mineral flotation process example

Case Study		MFP1 (Low Spec.)	MFP2 (Moderate Spec.)	MFP3 (High Spec.)
All-Equations	Computation time	6 hrs 22 min	45 hrs 44 min	Not used
	Computation time to generate equations	2 seconds	2 seconds	
	Number of equations generated	2,225	2,225	
	Number of nodes explored	5,645	35,289	
Decomposed Equations (3 subgraphs)	Computation time	13 seconds	40 seconds	1hr 29 min
	Computation time to generate equations	< 1 second	< 1 second	< 1 second
	Number of equations generated	27	27	33
	Number of nodes explored	5,077	13,622	200,245
All Variables	Computation time	23 min, 41 sec	2 hr, 16 min	7h 35 min
	Number of nodes explored	529,130	3,743,327	12,366,120
Inverted All Variables	Computation time	17 min, 20 sec	11 min, 18 sec	49 seconds
	Number of nodes explored	376,432	130,733	19,722

In the case study MFP1, the number of equations containing at least one of the four key variables $\{F_1, F_7, C_{1A}, C_{7B}\}$ in the equations-based tree search without decomposition

(All Equations method) is 2225. When the decomposition technique is used, the number of equations containing at least one key variable reduces significantly to 27, which is the main reason why the computation time reduces remarkably to less than 1 minute in both design cases. For comparison, in the linear Madron example solved by Gala and Bagajewicz (2006a), which has the same scale as our mineral example problem with 5 out of 24 variables required to be observable, the number of cutsets (depth of the tree) is 154. Although the All Variables method explores a lot more nodes than the All Equations method, the time is considerably smaller because the branching criterion in the All Equations method requires expensive computation to pick up the equation leading to the minimum cost sensor network among roughly 2210-2220 candidates (= total number of equations minus the number of equations active in the current node).

The larger tree size in this nonlinear example leads to much longer computation time because of two main reasons:

i. A larger tree depth requires an exponentially longer computation time to explore the tree

ii. A larger number of equations requires a longer time to perform the branching criterion in a node (picking the equation leading to the minimum-cost sensor network among all candidates). In fact, while example 2 of the Madron problem requires only 37 seconds to explore 937 nodes in the cutsets-based tree search (Gala and Bagajewicz, 2006a), in this example it takes roughly one hour to explore every 1000 nodes in the equations-based tree search without decomposition (All Equations method). Thus, reducing the tree depth by using a decomposition technique is *very beneficial* for

nonlinear systems. Although capable of finding the optimal solution, the All Equations method (equation-based without decomposition) is far less computationally efficient than the Decomposed Equations method or even the simpler All Variables method.

Tables 2.5 and 2.9 show that the computation time of the All Equations increases from less than one minute in the 13-variable CSTR example to several hours or almost 2 days in the 24-variable mineral flotation process example. We conclude that the All Equations method is severely affected by the scaling problem, which is due to the fact that the number of equations generated (the tree size) usually increases combinatorially with the size of the problem. The number of equations generated in nonlinear systems is also much larger than the number obtained in the same sized linear systems. The main reasons are:

i. Any combination of original equations may result in multiple new equations (not just only one as in linear systems)

ii. The possibility that there is common variable(s) between a pair of equations is high because one variable (such as the reactor temperature or the flowrate variables) can appear in several equations (instead of at most 2 in linear systems) as can be seen above (recall that ring sum operation on a pair of cutsets or variables substitution on a pair of equations can be performed only when there exists common stream(s) or common variable(s) between them).

For the mineral process example, the two best methods are the Decomposed Equations method and the Inverted All Variables method. When low or moderate specification is used (MFP1, MFP2), the Decomposed Equations method is the most efficient: it can find optimal solution within less than a minute. When high specification is used (MFP3), the Inverted All Variables method is the best because this method is tailored for such design case: in the design case MFP3, the number of sensors in feasible nodes is at least 15, hence the All Variables method (forward tree search) needs to explore at least 15 levels before it stops, while the inverted tree search explores not more than 9 (=24 minus 15) levels before it stops. This explains the remarkable improvement in computation time by using the inverted tree search.

2.7.3. Example 2.3: The Tennessee Eastman process

Consider the well-known challenge problem, the TE process, which is given in figure 2.10. The simplified TE model described by Ricker and Lee (1995) is used. The steady state operation conditions are generated from the Fortran file that implements the TE model available at Ricker's website (http://depts.washington.edu/control/LARRY/TE/download.html). The steady state equations used are:

$$y_{i,6}F_6 - y_{i,7}F_7 + \sum_{j=1}^{3} v_{ij}R_j = 0 \qquad i = A,B,...,H \qquad \text{(2-33)}$$

$$y_{i,7}F_7 - y_{i,8}(F_8 + F_9) - x_{i,10}F_{10} = 0 \qquad i = A,B,...,H \qquad \text{(2-34)}$$

$$z_{i,1}F_1 + z_{i,2}F_2 + z_{i,3}F_3 + F_{i,5} + y_{i,8}F_8 + F_i^* - y_{i,6}F_6 = 0 \qquad i = A, B, ..., H \tag{2-35}$$

$$(1-\phi_i)x_{i,10}F_{10} - x_{i,11}F_{11} \qquad i = G, H \tag{2-36}$$

where ϕ_i (i = G,H) : separation factor of component i in the stripper, $z_{i,j}$, $y_{i,j}$ and $x_{i,j}$: molar fraction of chemical i in stream j, which can be feed stream ($z_{i,j}$), liquid stream ($x_{i,j}$) or gas stream ($y_{i,j}$); v_{ij} : stoichiometry factor of chemical i in reaction j. The reaction rates R_j are given by the following expressions:

$$R_1 = \beta_1 V_{Vr} \exp\left[44.06 - \frac{42600}{RT_r}\right] P_{A,r}^{1.08} P_{C,r}^{0.311} P_{D,r}^{0.874} \tag{2-37}$$

$$R_2 = \beta_2 V_{Vr} \exp\left[10.27 - \frac{19500}{RT_r}\right] P_{A,r}^{1.15} P_{C,r}^{0.370} P_{D,r}^{1.00} \tag{2-38}$$

$$R_3 = \beta_3 V_{Vr} \exp\left[59.50 - \frac{59500}{RT_r}\right] P_{A,r}(0.77 P_{D,r} + P_{E,r}) \tag{2-39}$$

where β_j: "tuning" factor of reaction j; $V_{v,r}$: liquid volume in the reactor, T_r : temperature in the reactor, $P_{i,r}$: partial pressure of chemical i in the reactor.

The variables, their nominal operating conditions and costs of associated sensors are given in table 2.10. Sensor precision of 2% (for all variables) is used.

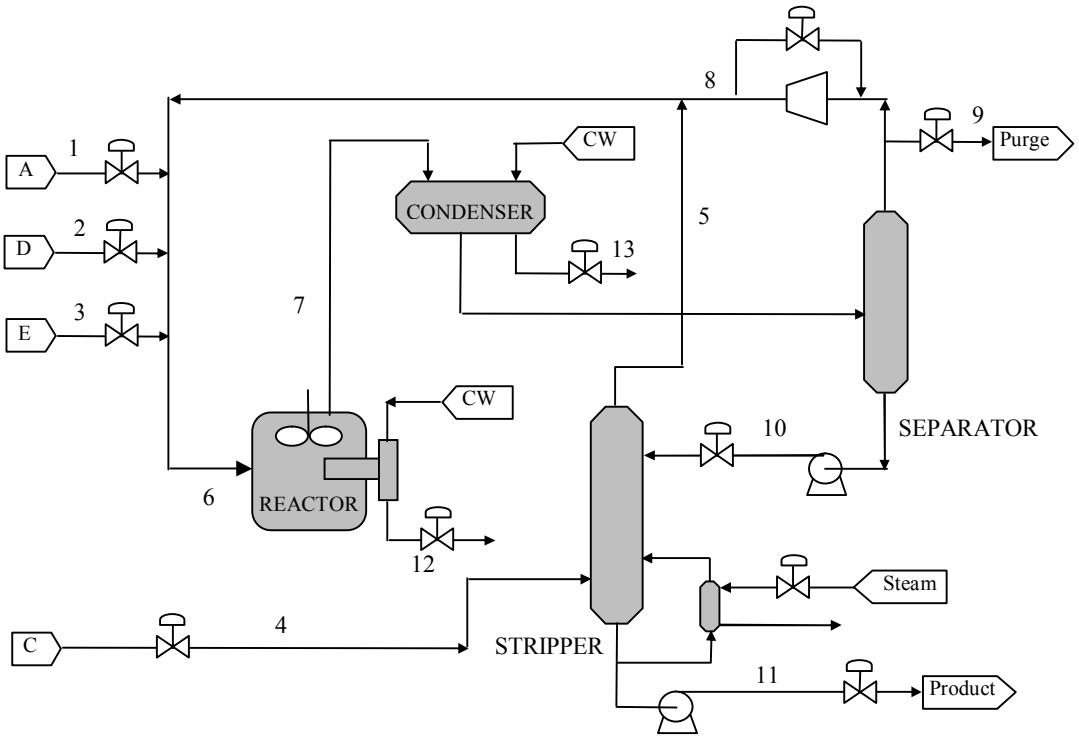


Figure 2.10 - The Tennessee Eastman Process (following Downs and Vogel, 1993)

Table 2.10 - Data for the Tennessee Eastman Problem

Variables	Nominal operating condition	Sensor cost	Variables	Nominal operating condition	Sensor cost
F_6	1889.9	300	$Y_{E,8}$	0.186	740
F_7	1475.2	300	$Y_{F,8}$	0.023	730
F_{10}	258.56	200	$Y_{G,8}$	0.048	740
F_{11}	211.3	200	$Y_{H,8}$	0.023	750
$Y_{A,6}$	0.322	770	$Y_{A,9}$	0.33	720
$Y_{B,6}$	0.089	780	$Y_{B,9}$	0.138	730
$Y_{C,6}$	0.264	730	$Y_{C,9}$	0.24	740
$Y_{D,6}$	0.069	740	$Y_{D,9}$	0.013	750
$Y_{E,6}$	0.187	750	$Y_{E,9}$	0.186	760
$Y_{F,6}$	0.016	760	$Y_{F,9}$	0.023	770

Variables	Nominal operating condition	Sensor cost	Variables	Nominal operating condition	Sensor cost
$Y_{G,6}$	0.035	810	$Y_{G,9}$	0.048	780
$Y_{H,6}$	0.017	820	$Y_{H,9}$	0.023	790
$Y_{A,7}$	0.272	750	$Y_{D,10}$	0.002	700
$Y_{B,7}$	0.114	760	$Y_{E,10}$	0.136	710
$Y_{C,7}$	0.198	700	$Y_{F,10}$	0.016	720
$Y_{D,7}$	0.011	710	$Y_{G,10}$	0.472	720
$Y_{E,7}$	0.177	720	$Y_{H,10}$	0.373	730
$Y_{F,7}$	0.022	730	$Y_{G,11}$	0.537	730
$Y_{G,7}$	0.123	780	$Y_{H,11}$	0.438	740
$Y_{H,7}$	0.084	790	P_r	2806	100
$Y_{A,8}$	0.33	780	T_r	393.6	500
$Y_{B,8}$	0.138	770	P_s	2734.7	100
$Y_{C,8}$	0.24	760	T_s	353.3	500
$Y_{D,8}$	0.013	750			

Values of flowrates F_i are given in kmol/hr, P_r, P_s: pressure in reactor and separator, respectively (KPa); T_r, T_s: temperature in reactor and separator, respectively (K); subscripts *A, B, C, D, E, F, G, H* denote components; subscripts *6, 7, 8, 9, 10, 11* denote stream number. The variables listed in table 2.10 are considered as candidates for measurements, other variables in the TE process (e.g. input flowrates F_1, F_2, F_3) are assumed to be either known by measurements (forced measurements) or of little importance for consideration. The total number of equations involving listed variables is 28.

Three design cases are considered, they are shown in table 2.11.

Table 2.11 - Design case studies for the TE process example

Design case	TE1 Low Spec.	TE2 Moderate Spec.	TE3 High Spec.
No. of key variables	6	17	39
Key variables	F_6, y_{A6}, y_{G6}, y_{H6}, F_7, F_{10}	F_6, y_{A6}, y_{G6}, y_{H6}, F_7, y_{G7}, y_{H7}, y_{A9}, y_{G9}, y_{H9}, F_{11}, y_{G11}, y_{H11}, P_r, T_r, P_s, T_s	All variables *except* {y_{D9}, y_{E9}, y_{F9}, y_{G9}, y_{H9}, F_{10}, y_{D10}, y_{E10}}
Requirement	Observabil-ity	Observability	Redundancy
Precision thresholds	2%	2%	1.5% 1.6% (y_{G8}, y_{H8})
Residual precision thresholds			4% for *all key variables except* {y_{G8}, y_{H8}, P_r, T_r, P_s, T_s}
Measured variables	F_6, y_{A6}, y_{G6}, y_{H6}, F_7, F_{10}	F_6, y_{A6}, y_{G6}, y_{H6}, F_7, y_{A7}, y_{A9}, y_{G9}, y_{H9}, y_{G10}, y_{H10}, F_{11}, y_{H11}, P_r, T_r, P_s, T_s	*all variables except* (y_{E9}, F_{10}, y_{E10}, P_r) (43 variables in total)
Sensors cost	3,200	9,630	29,640

For this example, the All Equations method was not used because: i) the computation time to generate all balance equations from $(2^{28} - 1)$ combinations of 28 original equation is too long; ii) its large tree size (large number of equations) leads to a long computation time. The computation time and number of nodes explored for the other methods are given in table 2.12.

Table 2.12 - Results for the TE process example

Case Study		TE1	TE2	TE3
All-Equations method		Not Used		
Decomposed Equations (9 subgraphs)	Computation time	2 min 40 seconds	> 74 hours (all current best solutions obtained in 14 hours)	>10 hours (Suboptimal solution found in one second)
	Computation time to generate equations	< 1 sec	< 1 sec	< 1 sec
	Number of equations generated	61	67	68
	Number of nodes explored	11,628	> 1.8 millions (All "current best" solutions obtained within first 510,000 nodes explored)	>500,000 (Suboptimal solution found at node 44)
All Variables	Computation time	9 hr 8 min	> 3 days (45 days estimated)	Not used
	Number of nodes explored	1,867,295	> 6 millions	
Inverted All Variables	Computation time	Not used	Not used	4 min 16 sec
	Number of nodes explored			1,726

For the case studies with low and moderate specification (TE1, TE2), the Inverted All Variable method is not used because the inverted tree search will perform poorly for design case with low specification like TE1. The case study TE2 is similar to the case study MFP1 (mineral flotation example) where the number of sensors in the optimal (or

sub-optimal) solution accounts for less than a half of all sensors; hence the performance of the inverted tree search is predicted to be comparable to that of the All Variables method, which performs poorly as shown in table 12.

The proposed methods can find optimal solution for the two design cases with low and high specification (TE1 and TE3, respectively) with the Decomposed Equations method being the best method for the design case TE1 while the Inverted All Variables method is the best method for TE3 as expected. Optimal solution for these two design cases are found within less than 5 minutes.

For design case with moderate specification (TE2), the Decomposed Equations method is the only viable option when the factor of acceptable computation time is desired. In fact, after roughly 3 days of running time and 6 millions number of nodes explored, the "current best" solution found by the All Variables method consists of 21 sensors with the cost of 14,120, far worse than the solution found by the Decomposed Equations method. We assume that the computation time of the All Variables method is comparable to the computation time of the same sized linear system, the CDU process presented in Gala and Bagajewicz (2006b), which was estimated to be 45 days. The Decomposed Equations method was then used: the number of decompositions is 8 (original graph is decomposed into 9 sub-graphs), the number of equations generated is 67. All the "current best" solutions (the incumbent) were found within the first 510,000 nodes explored, 14 hours computation time. After that the tree search procedure kept running for 60 hours, explored further 1.3 millions nodes without finding any better solution before it was terminated (so in total, 74 hours running time and 1.8 millions number of nodes explored). The current best solution is reported here, which is to

measure $F_{6,}$ y_{A6}, y_{G6}, y_{H6}, $F_{7,}$ y_{A7}, y_{A9}, y_{G9}, y_{H9}, y_{G10}, y_{H10}, $F_{11,}$ y_{H11}, P_r, T_r, P_s, and T_s. It has a cost of 9,630. This solution contains only 14 of the 17 key variables: $F_{6,}$ y_{A6}, y_{G6}, y_{H6}, $F_{7,}$ y_{A9}, y_{G9}, y_{H9}, $F_{11,}$ y_{H11}, P_r, T_r, P_s, and T_s.

Finally, Case Study TE3 is one that will require a lot of measurements, because the requirements are very strict. In this case, the best solution obtained by the Decomposed Equations after roughly 500,000 nodes explored, 10 hours computation time (before it was terminated) is to measure all variables except three variables (y_{E9}, $F_{10,}$ y_{E10}) with a cost of 29,740. This solution contains one more sensor for measuring P_r and a cost slightly higher (100 more) when compared with the optimal solution found by inverted tree search. Thus, even in the case that the Decomposed Equations method has to be terminated when the computation time becomes unacceptably long (e.g. with large scale problems with high number of key variables) such that the finding of optimal solution is not guaranteed, the method finds sub-optimal solutions very near to the global optimal solution within an acceptable time (in fact, this sub-optimal solution is found at node 44, just after 1 second running time).

The results shown here point out that

i. When the level of desired properties (the specifications) of sensor network is either low or high (e.g. either a small or a large number of key variables is involved), even large scale nonlinear sensor network problem like the TE problem can be solved efficiently using either the Decomposed Equations method (for a low level of specifications) or inverted tree search method (for a high level of specifications)

ii. For realistic design cases of large scale nonlinear problem where the level of desired properties is neither low nor high, the Decomposed Equations method is able to find optimal or near-optimal solution within an acceptable time, however the optimality is not guaranteed because the computation process has to be terminated half-way.

2.8. Choice of strategy

It can be seen that the inverted tree search remarkably improves the computational time when a high level of specifications is desired. A step further is to intelligently select which strategy to use corresponding to a specific design case in order to have the shortest computational time. The criterion of when to choose the inverted tree search instead of a forward tree search, inferred empirically by our observations of testing problems, is to choose the inverted tree search when the followings conditions are met:

i. The number of variables is more than 15

ii. The average number of sensors in feasible solutions is more than 70% of the total number of variables (based on the first 10 feasible solutions found).

The first condition is needed because: a) the Equations-based methods (forward strategy) can solve problems involving not more than 15 variables very efficiently, hence Inverted tree search is not needed for this type of small scale problem, b) if number of variables is less than 15, usually the computation time is short and the time to compute the average number of sensors in feasible solutions offsets the gain in computation time (if any) by using the inverted tree search.

The procedure to quickly calculate the average number of sensors in feasible solutions that is as close as possible to the number of sensors in the optimal solution has three steps:

i. Prepare two lists of variables: one list of key variables (list one) and a list of non-key variables (list two)

ii. If measuring *all* key variables is a feasible solution, then use the number of key variables as final result and stop, otherwise go to step iii

iii. Employ the tree enumeration procedure using non-key variables from list two to find the first ten combinations of *all* key variables (list one) and non-key variables (from list two) that are feasible solutions.

The average number of sensors in feasible solutions and the associated computation time for the four mineral process design cases are shown in table 2.13:

Table 2.13 - Results used for selecting the right tree search strategy

Design cases	MFP1	MFP2	MFP3
Computation time (second)	2	16	1
Average number of sensors in feasible solution	16.45	16.55	20.36
Number of sensors in optimal solution	10	12	15

According to the results shown in table 2.13 and the criteria presented above, the design case MFP3 should be solved by using inverted tree search. The calculation results shown in table 2.9 confirm that this is the "right" choice. The two design cases MFP1 & MFP2 should be solved by using the Decomposed Equations method (forward strategy).

2.9. Conclusions

In this chapter, the equations-based tree search method for the design of nonlinear sensor network was presented. The proposed method is guaranteed to find optimal solution and is computationally efficient for small scale and middle scale problems. However, its performance is not always satisfactory when dealing with large scale problems. Another version of the tree search method, the inverted tree search using the list of variables, was also presented. The inverted strategy is tailored for design cases with high level of specifications and is shown to remarkably improve the computation time, especially with large scale nonlinear problems like the TE process where it solved a high specifications design case within a few minutes.

For realistic large scale nonlinear design problems (those with moderate level of specifications), the equation-based method is not yet efficient enough; thus a more efficient method is needed. This method is presented in the next chapter

2.10. Nomenclature

- c_i : cost of sensor i
- q_i : binary variable indication whether sensor i is used
- σ_i : precision of estimator i
- σ_i^* : precision threshold of estimator i
- F_i : flowrate of stream i
- h_i : enthalphy of stream i
- c_{ij} : concentration of component j in stream i
- δ_i: stoichiometric coefficient of component i in chemical reaction.

- T_r & P_r: temperature and pressure of reactor
- T_s & P_s: temperature and pressure of separator
- $P_{i,r}$: partial pressure of chemical i in the reactor
- T_f : flash drum temperature
- F_i, T_i , c_{Ai} : inlet flowrate, temperature and concentration of A of the CSTR reactor
- F, T , c_A : outlet flowrate, temperature and concentration of A of the CSTR reactor
- F_{ci}, T_{ci} : inlet flowrate and temperature of coolant in the CSTR reactor
- F_c, T_c : outlet flowrate and temperature of coolant in the CSTR reactor
- F_{vg} : flowrate of vent gas leaving the CSTR reactor
- U, A : heat transfer coefficient and heat transfer area in the CSTR reactor
- C_d : catalyst activity
- V_j : volume of jacket
- V: reactor volume
- $V_{v,r}$: liquid volume in the reactor
- C_p, ρ : heat capacity and density of fluid mixture in the CSTR reactor
- C_{pj}, ρ_j : heat capacity and density of coolant
- K_i : vapor-liquid equilibrium ratio of component i
- E : activation energy of chemical reaction
- k_0 : pre-exponential factor or frequency factor
- α : reaction order
- ν_{ij} : stoichiometry factor of chemical i in reaction j
- ϕ_i : separation factor of component i in the stripper
- β_j: "tuning" factor of reaction j

- R : universal gas constant
- R_j : reaction rate of reaction j
- ΔH_{rxn} (or simply ΔH): heat of reaction.
- Q_{vap} : amount of heat removed in the flashing process
- $z_{i,j}$, $x_{i,j}$ and $y_{i,j}$: molar fraction of component i in stream j, which can be feed stream ($z_{i,j}$), liquid stream ($x_{i,j}$) or gas stream ($y_{i,j}$)

2.11. References

Ali, Y., and Narasimhan, S. Sensor Network Design for Maximizing Reliability of Bilinear Processes. *AIChe J.* **1996**, 42(9), 2563-2575.

Bagajewicz, M., Design and Retrofit of Sensors Networks in Process Plants. *AIChe J.* **1997**, 43(9), 2300-2306.

Bagajewicz, M. *Design and Upgrade of Process Plant Instrumentation.* Technomic Publishers: Lancaster, PA, **2000**.

Bhushan M. and R. Rengaswamy. Comprehensive Design of Sensor Networks for Chemical Plants Based on Various Diagnosability and Reliabilty Criteria. I. Framework. *Ind. Eng. Chem. Res*. **2002a**, 41, 1826-1839.

Bhushan M. and R. Rengaswamy. Comprehensive Design of Sensor Networks for Chemical Plants Based on Various Diagnosability and Reliabilty Criteria. II. Applications. *Ind. Eng. Chem. Res.* **2002b**, 41, 1840-1860.

Chmielewski, D.,Palmer, T.,Manousiouthakis, V. On the Theory of Optimal Sensor Placement. *AIChe J.* **2002**, 48(5), 1001-1012.

Downs, J. J. and Vogel, E. F. A Plant-wide Industrial Process Control Problem. *Comput. Chem. Eng.* **1993**, *17(3)*, 245-255.

Gala, M. and Bagajewicz, M. J. Rigorous Methodology for the Design and Upgrade of Sensor Networks Using Cutsets. *Ind. Eng. Chem. Res*. **2006a**, 45(20), 6687-6697.

Gala, M. and Bagajewicz, M. J. Efficient Procedure for the Design and Upgrade of Sensor Networks Using Cutsets and Rigorous Decomposition. *Ind. Eng. Chem. Res*. **2006b**; 45(20), 6679-6686.

Heyen, G., Dumont, M. and Kalitventzeff, B. Computer-aided design of redundant sensor networks, in: J. Grievink, J. van Schijndel (Eds.), *Proceeding of 12th European Symposium on Computer-aided Process Engineering*, Elsevier Science, Amsterdam, **2002**, pp. 685–690.

Kretsovalis, A. and R. S. H. Mah, Effect of Redundancy on Estimation Accuracy in Process Data Reconciliation. *Chem. Eng. Sc.* **1987***,* 42**,** 2115.

Mah, R.S.H. *Chemical Process Structures and Information Flows.* Butterworths: Stoneham, MA, USA, **1990**.

Ricker N.L and Lee J.H. Nonlinear Modeling and State Estimation for the Tennessee Eastman Challenge Process. *Comput. Chem. Eng.* **1995**, 19(9), 983-1005.

3. NEW EFFICIENT METHODOLOGY FOR NONLINEAR SENSOR NETWORK DESIGN PROBLEMS

The instrumentation network design and upgrade problem can only be solved to optimality using a branch and prune tree search strategy, usually a depth first one. In this chapter, an efficient tree search strategy that explores the tree horizontally and exploits certain cost property of the different nodes in the tree is presented. This method guarantees optimality and its performance is much better than the depth first strategy. In case this rigorous horizontal search method is not efficient enough for certain types of problems, an approximate method is proposed to complement for this horizontal search method. The approximate method is shown to be very efficient and it is able to locate optimal solutions for all the design case studies.

3.1. Overview

Two main groups of computational methods have been used to solve the SNDP for *process monitoring* purpose: mathematical (integer) programming methods (these methods guarantee optimality but they usually exhibit scaling problem) and stochastic methods (e.g. genetic algorithms, these methods do not guarantee optimality). In the first group, while other researchers transformed problems into well-established optimization models such as mixed integer linear programming MILP (which was then solved by

using GAMS), our group particularly used the branch and bound "tree search" methods (which were implemented in Fortran).

The nonlinear SNDP is the most computationally challenging problem. The first attempt to solve the nonlinear SNDP, the equation-based tree search method, has some success but it fails to locate optimal solution within an acceptable time for realistic large scale nonlinear problems. In this chapter, new efficient methods that address the shortcomings of the equation-based method are presented. Two methods are proposed, which are sequentially presented in two parts of this chapter

The first part presents a new tree search method : instead of exploring the tree through its branches, i.e. adding one instrument at a time and pruning branches using a certain stopping criteria, we resort to look at configurations with the same number of instruments, that is, looking at all branches at the same level of the tree. The former is called depth-first tree exploration, while what we propose is known as breadth-first strategy (Diwekar, 2008). In addition, a few modifications to these strategies are proposed and the stopping criterion that is particular to minimum cost problems (like the one we intend to solve) is also provided. The proposed method belongs to the breadth first strategy category (in the sense that it expands first all successor / sister nodes of the current node rather than going down the tree) but it is not exactly the breadth-first branch and bound method as described in optimization textbooks, so we name it "level-by-level" search.

The second part presents a heuristic local search attempting to locate optimal solution starting from sub-optimal solutions provided by equation-based tree search method. The local search is a two-step procedure. This local search method is meant to

be complementary for the "level-by-level" tree search and equation-based tree search: it will be used in final step if these two tree search methods (which are rigorous methods that guarantee optimality) fail to identify optimal solution within an acceptable computational time.

3.2. Tree search methods

Solution strategies used in previous works:

- Transforming the problem into well-established optimization problems (MILP, convex optimization using linear matrix inequalities techniques) by introducing auxiliary variables. Applied to small scale linear systems.
- Branch and bound (tree search) method. The base unit can be single measurement (Bagajewicz, 1997) or cutset of process graph (Gala and Bagajewicz, 2006a, 2006b) or process balance equations (Nguyen & Bagajewicz, 2008).

The direct enumeration tree search method is illustrated in figure 3.1. The procedure is as follows:

- Start with a root node with no variables being measured (q = 0), it is trivially infeasible.
- Develop each branch and making one element of q active until the stopping criterion is met. Then back up one level and develop next branch using a branching criteria.

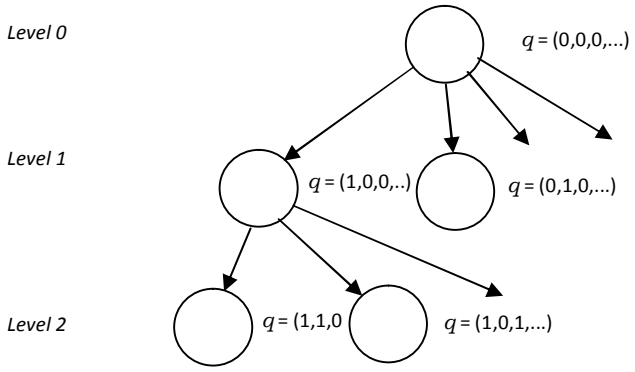


Figure 3.1 - Tree search method

Branching Criteria: Sensors are added to nodes in the direction of minimum cost, that is, the sensor chosen to be added to the current node has the cheapest cost among all candidates

Stopping Criteria: In each node, the following two operations are performed in sequential order to determine if we need to continue exploring the current branch: i) stop if the cost of the current node is more than the current best because even if the cost is feasible it cannot compete with the current best, ii) stop if the node is feasible; update the current best if the cost of this feasible node is less than the current best.

This stopping criterion is valid for both depth first tree search and level traversal (breadth first) tree search (described below). In depth first tree search, if the stopping criterion is met, one should stop, back up one level and develop the next branch, as any node below will be more expensive.

The depth first tree search method is not efficient for medium and large scale problems because computational time increases exponentially with the size of the

problem. To deal with these problems, tree search methods based on cutsets were proposed (Gala and Bagajewicz, 2006a, 2006b). Later, Nguyen and Bagajewicz (2008) extended the cutset-based method to solve nonlinear problems by using process balance equations and incidence matrix manipulation instead of cutsets and graph decomposition (called equation-based method, which is presented in chapter 2). Nguyen and Bagajewicz (2008) also presented a technique based on an inverted tree search. The idea behind this method is to explore the tree in the *reverse* direction, that is, it to start with a root node containing *all* sensors and continue *removing* sensors when going up the tree until an infeasible node is found (stopping criterion). This method is very efficient for problems with high level of specifications (e.g. when there are many key variables or redundancy is required) where feasible solutions contain a large portion of available sensors.

Table 3.1 summarizes the most suitable methods (that were developed in our group) for each case

Table 3.1 - Most suitable method for solving sensor network design problem

Level of specifications	Linear systems	Nonlinear systems
Low	Cutsets-based or measurement-based tree search	Equations-based or measurement-based tree search
Medium	Cutsets-based	Equations-based
High	Cutsets-based or Inverted tree search	Equations-based or Inverted tree search

Note that in table 3.1, the cutsets-based and equations-based methods are meant to be the ones with decomposition (which is always better than the corresponding versions without decomposition). It can be noted that the cutsets-based and equations-based

methods (with decomposition) are the best choice for all levels of specifications while occasionally these methods are outperformed by the measurement-based tree search or inverted tree search for problems with low level and high level of specifications, respectively.

A level traversal (breadth first) technique that takes advantage of certain properties of trees that are constructed using the minimum cost branching criteria is now presented.

3.3. Level traversal search

Let us start first by defining the following terms:

Sister nodes: sister nodes of a current node are the ones that: i) are at the same level (containing the same number of active elements) and in the right hand side of the current node, ii) share the shame root (parent) with the current node

Families of nodes: a set (family) of nodes that have the same number of sensors (same number of active elements) and share the same root (same parent)

Head of family: the leftmost node in the family of nodes, that is the cheapest node.

To illustrate the above concepts, consider a list of sensors in ascending order 123456. At level three, the following nodes (consisting of three sensors) (123, 124, 125, 126) are said to form a family of nodes with parent (root) 12 (Figure 3.2). The next families of nodes at the same level are (134, 135, 136) with parent root 13, (145, 146),

with parent root 14, and finally, (156), with parent root 15. The heads of families are 123, 134, 145 and 156.

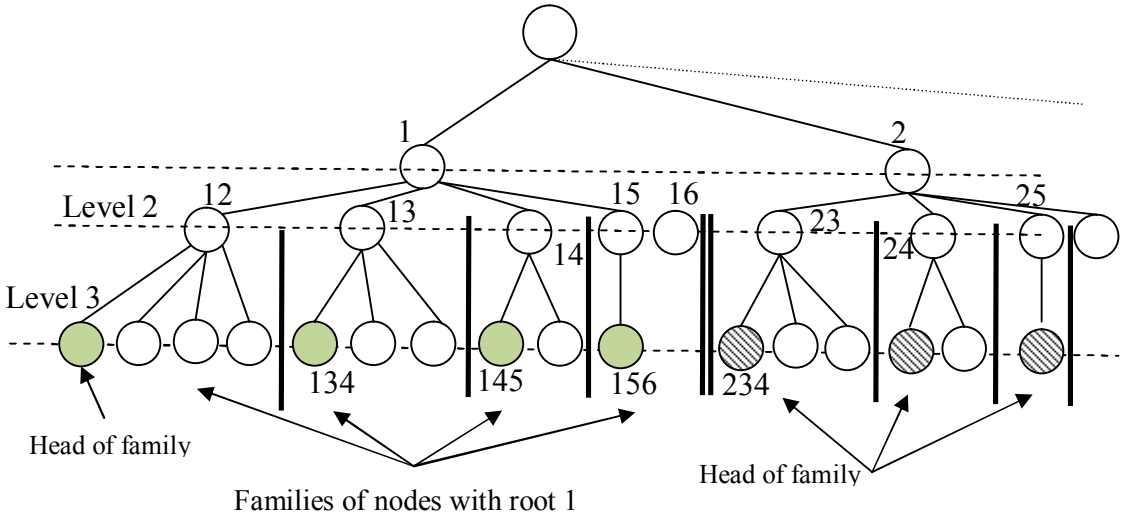


Figure 3.2 - Families of nodes

We now note the following properties when the tree is built using the cheapest candidate (minimum cost branching criteria).

- Property 1: There is cost monotonicity within each family, that is, the cost increases as one moves within each family to the right.
- Property 2: There is cost increasing monotonicity among heads of family that share the same root.

Property 1 is straightforward: it stems from the minimum cost branching criteria. Property 2 is also self-evident from the branching criteria. For example Node 123 has smaller cost than node 134 and so on. This is because they share the same root (node 1). However, a member of one family can in fact have a larger cost than members of any

family on the right. For example, node 125 (third member of the first family) can have higher cost than node 134.

Properties 1 and 2 can be used efficiently in a level traversal strategy. Suppose that one node at level 3 is found to satisfy the stopping criteria (it is feasible and its cost is smaller than the current upper bound, or simply if it is more costly than the current upper bound). Assume that the node is a head of family. In such case one can directly omit looking at all families sharing the same root and move to the families in the next root. For example if the head of family "123" is found to satisfy the stopping criterion, then all the sister nodes of this node (124, 125, 126) and the families on the right side with heads 134, 145,and 156 have higher cost (property 1). Thus, the traversal search should continue looking at the families that have different roots, but only comparing the current best node with heads of families corresponding to all other different roots. For example, the node 123 has smaller cost that node 234 and smaller cost than 245 and so on. Therefore, if 123, the head of the first family, is the current node, then one can dismiss all other families. However, if the current node is not a head of the first family, but head of other families on the left, monotonicity also holds. For example 134 is cheaper than 234, and cheaper than 245 and so on. This monotonicity breaks at some point. For example, node 156 can be more costly than 234, but if it is cheaper, then one can dismiss all nodes to the right of 234. This suggests a strategy in which the current feasible node is compared to heads of families on the left only until the monotonicity breaks.

This discussion points out that:

- The level traversal tree search strategy is very efficient if a current best is identified in left hand side of the tree, in such case lots of nodes in the right side of that current best node can be eliminated. For example, if node "1234" is identified to be current best (the leftmost node in the level consisting of 4 sensors), then no other nodes in this level can compete with the node 1234 and we can quickly move to the upper levels.
- Conversely, if the level traversal tree search cannot identify a node satisfying the stopping criterion until the end (the nodes on the rightmost side) of the current level, the tree search has to explore the whole level (explore all nodes from left to right). If this situation occurs, the level traversal tree search is not efficient to solve large scale problems.

The calculation procedure is then described next

The calculation procedure starts with a depth first search using the branching criteria based on adding the cheapest sensor until a feasible node is found or when a certain amount of nodes have been explored. This strategy is not efficient for medium and large scale problems, so the depth first procedure stops when the number of nodes explore reach a pre-defined limit. Assume that the current best node has been identified.

Because the current best (identified by depth first search) is unlikely to be global optimum, the tree search continues seeking for global optimum by exploring the nodes in the right hand side and at the same level with the current best (that is, breadth first or level traversal strategy). In this strategy, within a family of nodes, tree search looks for a node satisfying stopping criterion and updates the current best if applicable. Because

sensors are added in the direction of minimum cost, the sister nodes of this node have higher cost than the current best so they are disregarded (Figure 2.3). The search should continue with the next families of nodes at the same level and families in upper levels until we identify a level where all nodes are infeasible (Figure 2.4).

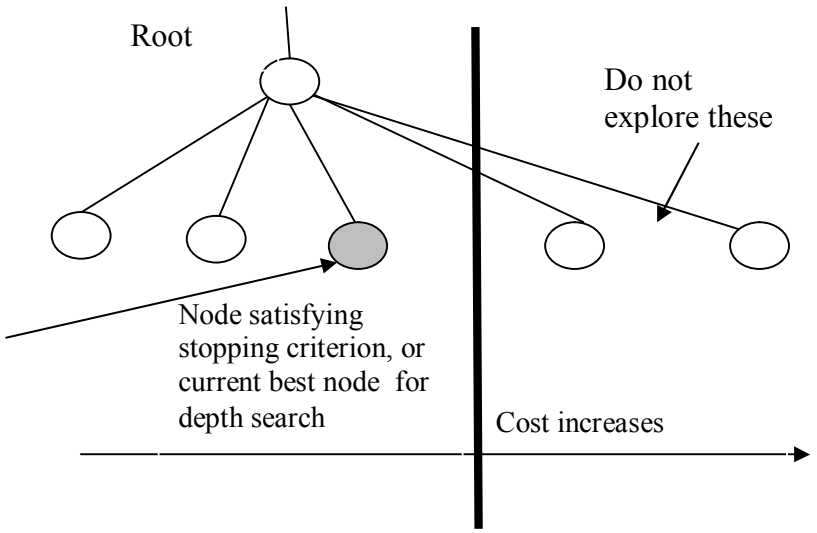


Figure 3.3 - Stopping criterion

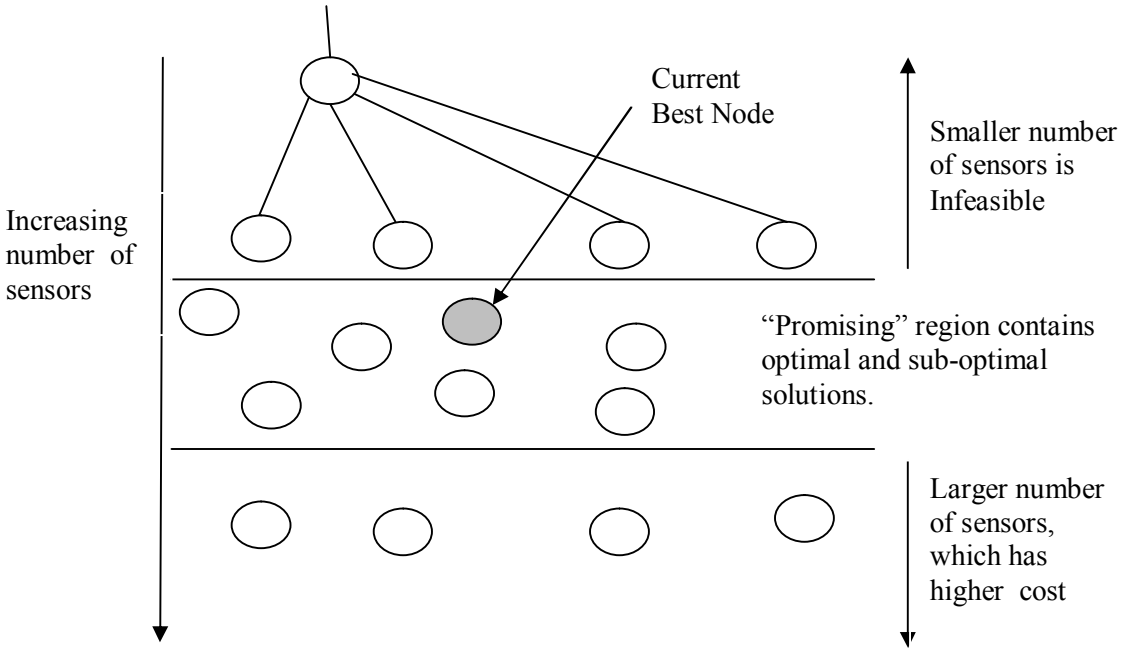


Figure 3.4 - Searched space in horizontal search

Thus, the essence of the new method is to search the tree horizontally within the "promising" region only, defined as some number of levels above the currently identified. Compared with the depth first tree search method, this method saves computational time by skipping the region above the "promising region" where nodes are infeasible. The question is how to explore this region horizontally and how to guarantee global optimality. For this two methods are proposed

3.4. Level-by-level search

The procedure of this method is as follows:

1. Run the depth first tree search method. Record the current best solutions and the associated depth level (number of sensors) in those solutions
2. Stop if the number of nodes explored reaches the predefined limit. This limit depends on the size of the problem. The current best solution found is denoted as *XQ* and its depth level (the number of sensors) is denoted as *Nle*. (See Figure 3.5). At this point all nodes to the left of *XQ* and their children have been explored. Property 3 allows us not to explore the next level (*Nle*-1). If there is a better solution, it is in this level or previous ones.
3. If node *XQ* is a leftmost head of a family, identify its parent and move to the next level up (*Nle*-1). If not stay at level *Nle*. Either way go to step 4.
4. Identify all the families of nodes at the current level and on the right hand side of XQ
5. In each family, identifying the node that satisfies the stopping criterion. If that node is not a head of family, continue exploring the next families. If that node is a head of family, disregarding all the nodes that are in the right hand side and share the same root with that node. For example, if that node (which is a head of a family) is "123478910", then all the nodes that are on the right hand side and share the same root ("1234") with that node shall be disregarded; the next node to be explored is "12356789" (assuming there is ascending order in cost from sensor 1 to sensor 10)
6. If all nodes in the current levels are either explored or disregarded, continue exploring the upper levels

7. The tree search terminates once it identifies a level where all nodes are found to be infeasible.

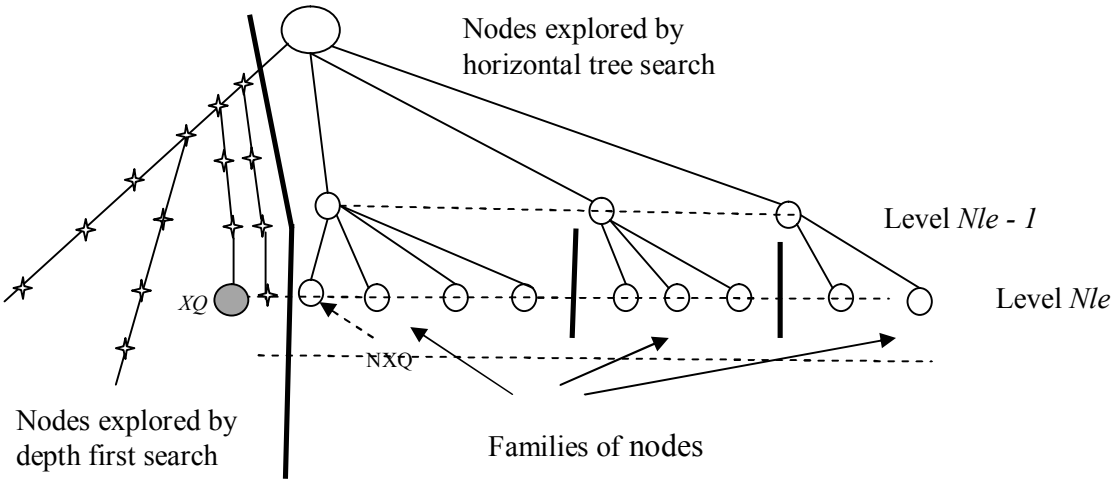


Figure 3.5 - Level-by-level search

3.5. Hybrid vertical and "level-by-level" search

In this method, we combine breadth first and depth first strategies. The method is outlined next:

1. Run the depth first tree search method. Record the current best solutions and the associated depth level (number of sensors) in those solutions
2. Stop tree search if the number of nodes explored reaches the predefined limit. This limit depends on the size of the problem
3. The current best solution found is denoted as XQ and its depth level (the number of sensors) is denoted as *Nle*. The number of sensors in optimal solution is at most equal to *Nle*. Testing results (for medium problems) show

that the number of sensors (depth level) in the optimal solution is generally in the range [*Nle – 2, Nle – 5]*

4. Switch to level traversal search.
5. Choose the depth level to perform horizontal search to be one value in the range [*Nle – 1, Nle – 5],* denoted as *N*
6. Explore horizontally all the *nodes on the right hand side* of the branch that contains XQ that have the same depth level of *N.* These nodes are called root nodes.
7. In each root node, check for its feasibility, then:

- If the current root node (level *N*) is feasible, then the nodes in the upper levels (*N-1*, *N-2*, etc.) can also be feasible. Then
    - Explore the upper levels (*N-1*, *N-2*, etc.) by removing sensors out of the root node (we are exploring the parents only) with the following stopping criterion: stop exploring when the node is found to be infeasible.
    - Do not explore the sister nodes in the same family with the current root node because even if these sister nodes are feasible, they result in the same nodes in the upper levels (*N-1*, *N-2*, etc.) as with the current root node.
    - If that feasible node is head of a family, do not explore the families of nodes that share the same root and on the right hand side of that head of family
- If the current root node (level *N*) is infeasible

- If its cost is lower than current best cost, explore the lower levels (*N+1*, *N+2, etc.*), but do not explore level *Nle*
- If the cost is larger than current best cost, skip this node and the associated sister nodes of this current root node. If that node (which has higher cost than the current best) is head of a family, do not explore the families of nodes that share the same root and on the right hand side of that head of family

The procedure is depicted in figure 3.6

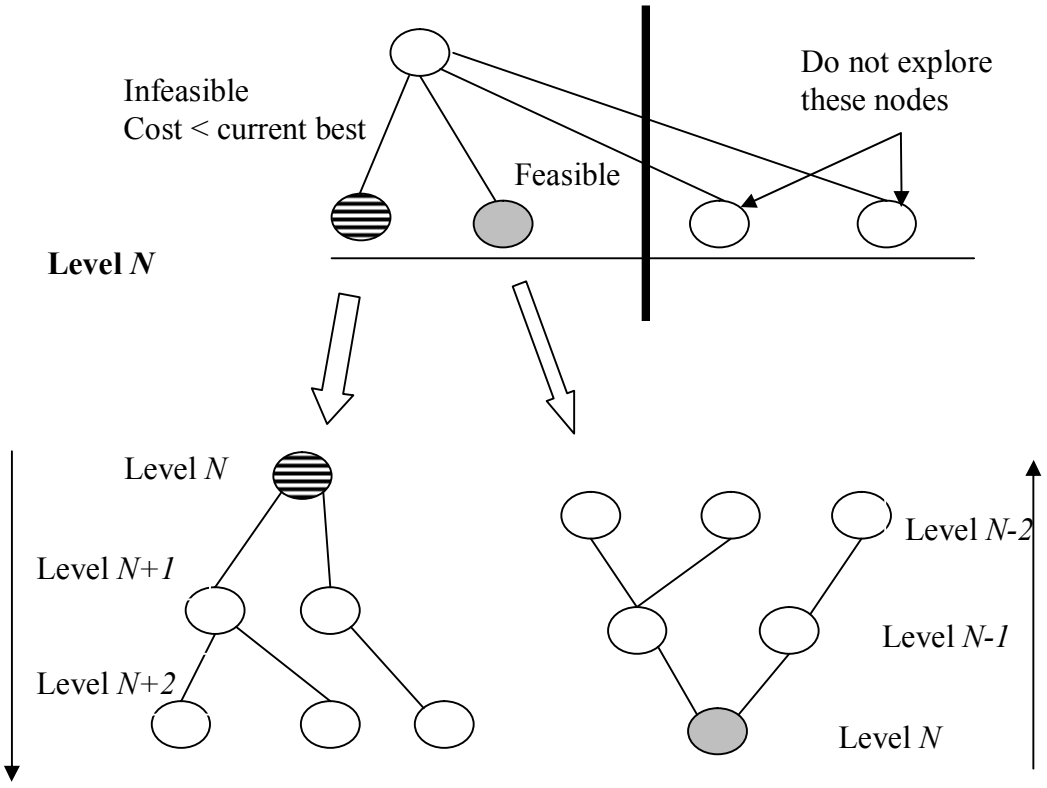


Figure 3.6 - Hybrid Vertical and Level-by-level Search

3.6. Illustrated example – Level traversal methods

The proposed methods are implemented in Fortran running on a 2.8 GHz Intel Pentium CPU 1028 MB RAM PC.

Example 3.1: The mineral flotation process example, introduced in chapter 2 (example 2.2), is used. The same process flowsheet (figure 2.9) and data (tables 2.6 and 2.7) are used. The same design specifications (table 2.8) are used. The level traversal tree search methods, the Level-by-level search and Hybrid search are used to solve the problem. They both identify the optimal solution. The design specifications and the optimal solution are given in Table 3.2

Table 3.2 - Design case studies for the mineral flotation process example

Case Study	MFP1 Low Spec.	MFP2 Moderate Spec.	MFP3 High Spec
No. of key variables	4	4	12
Key variables	F_1, C_{1A}, F_7 and C_{7B}	F_1, C_{1A}, F_7 and C_{7B}	F_1, F_4, F_6, C_{1A}, C_{1B}, F_7, C_{4A}, C_{4B}, C_{6A}, C_{6B}, C_{7A}, C_{7B}
Requirement	Observability	Redundancy	Observability
Precision thresholds	1.5% (F_1, C_{1A}) 2% (F_7 , C_{7B})	1.5% (F_1, C_{1A}) 2% (F_7 , C_{7B})	1.5% (F_1, F_4, F_6, C_{1A}, C_{1B}) 2% (F_7, C_{4A}, C_{4B}, C_{6A}, C_{6B}, C_{7A}, C_{7B})
Residual precision thresholds		5% for F_1, F_7 only	
Measured variables	F_1, F_3, F_5, F_6, F_7, F_8, C_{1A}, C_{2A}, C_{5A}, C_{7B}	F_1, F_3, F_5, F_6, F_7, F_8, C_{1A}, C_{2A}, C_{3B}, C_{4B}, C_{5A} and C_{7B}	F_1, F_3, F_5, F_6, F_7, F_8, C_{1A}, C_{2A}, C_{3B}, C_{4A}, C_{4B}, C_{5A}, C_{6B}, C_{7A} and C_{7B}
Sensors cost	1448	2118	2968

The performance of the two level traversal tree search methods are shown in Table 3.3, and compared to the performance of the depth first tree search. The predefined limit to stop the depth first tree search and switch to level traversal search is 500 (for level-by-level search) and 5000 (for hybrid search). In the hybrid search, the level to be explored horizontally is 2 levels above the level of the current best identified by the depth first tree search (that is, $N = Nle - 2$)

Table 3.3 - Performance of level traversal tree search methods, mineral flotation process example

Case Study		MFP1 (Low Spec.)	MFP2 (Moderate Spec.)	MFP3 (High Spec.)
Hybrid search	Computation time	4 min 24 sec	7 min 22 sec	11 min 5 sec
	Number of nodes explored	205,168	119,188	186,521
Level-by-level search	Computation time	1 min 11 sec	2 min 52 sec	6 min 42 sec
	Number of nodes explored	57,143	117,382	171,975
Depth First tree search	Computation time	23 min, 41 sec	2 hr, 16 min	7h 35 min
	Number of nodes explored	529,130	3,743,327	12,366,120
Equations-based with decomposition	Computation time	13 seconds	40 seconds	1hr 29 min
	Number of nodes explored	5,077	13,622	200,245

It can be seen that the level-by-level search is generally better than the hybrid search. In the two design cases MFP2 and MFP3, the two level traversal search methods explored similar number of nodes but the computational time of the level-by-level search is shorter than the other. The difference is largely due to implementation issue in Fortran of the hybrid search: from a node in the chosen level ($N = Nle - 2$), the tree search either goes up (explore upper levels $N - 1$, N -2) or goes down (explore next level $N + 1$, $N + 2$)

by calling the appropriate subroutines. It is well known that before the commands in a subroutine are executed, a certain amount of time is spent to perform the pre-processing step (known in computer science as "overhead"). The "extra" time spent on "overhead" explains why the computational time of hybrid method is larger than that of the level-by-level search. Although for design cases MFP1 & MFP2, the level-by-level search is not better than the equations-based method with decomposition, but if design case MFP3 is included for comparison, the level-by-level search can be considered to be better than the equations-based method because the level-by-level search solved the design case MFP3 much faster. Another advantage of the level-by-level search over the equations-based method is that it is much simpler to use because it does not require any knowledge to decompose the problem (a poor choice of how to decompose the problem in equations-based method can lead to much longer computational time)

Detail of steps in the level-by-level search for the design case 3 (MFP3) is now illustrated. The list of sensors in ascending order of cost is [5, 3, 6, 1, 7, 2, 8, 4, 13, 17, 15, 20, 24, 18, 10, 22, 9, 11, 21, 14, 23, 16, 12, 19] (vector *SC*). For simplicity, we use the indexes (or locations) of sensors in the vector *SC* to indicate the measurement locations. For example, if the active element in vector $\boldsymbol{q}$ (in Eq. 1 and 2) is [1,2,3] then the actual chosen sensors (measurement location) are 5, 3 and 6 whose indexes in *SC* are 1, 2 and 3 respectively. The solution $\boldsymbol{q}$ = [123] has the smallest cost among all the solutions that have three sensors.

Let the set R be defined as follows: R=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. The depth first tree search after exploring 500 nodes identifies [R, 14, 16, 17, 18, 19, 20, 22] as current best (containing 20 sensors and the current best cost is 3878) at node 406. The

node at which the depth first tree search terminates (node *XQ*) is [R, 14, 16, *18, 21, 23, 24*] (19 sensors). The first node to be explored by level-by-level search is the node that has the same level with the current best (20) and on the right hand side of *XQ*. That node is [R, 14, 16, *19, 20, 21, 22, 23*] (the different part between this node and *XQ* is italicized. The node is also a head of family whose root is [R, 14, 16] and it has higher cost (cost is 3953) than current best. Thus all nodes on the right hand side and share the same root with this node are disregarded. The next node to be explored is [R, 14, *17, 18, 19, 20, 21, 22*]. The same thing is observed (head of family with higher cost than current best), thus the next node to be explored is [R, *15, 16, 17, 18, 19, 20, 21*] (which is again a head of family) (the roots of those heads of families are indicated by normal letters, the other members are indicated by italicized letters). The same thing is observed and this node is disregarded. There is no node in the current level (20) can compete with the current best. After exploring roughly 9000 nodes, the tree search completes exploring the two levels 20 & 19 (found a new current best at level 19, the new current best cost is 3653) and quickly moves to the next level (18). The first node to be explored in this current level (number of sensors = 18) is [R, 14, 16, 18, 22, *23*], this node has lower cost than the current best but it is infeasible, so its sisters node ([R, 14, 16, 18, 22, *24*]) is explored, which does not satisfy the stopping criterion either. The tree search keeps searching horizontally from left to right; in the process it visited nodes that have lower cost than current best but they are infeasible. The first node that is better than the current best is [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 17, 18, *19, 20, 22*]. This new current best is not a head of family, so the next families are explored. The tree search continues in that fashion.

We have shown how the searching process proceeds and how to eliminate non-optimal nodes in the level-by-level search. The hybrid search is performed in the similar fashion. The only difference is that the hybrid search explores only one level (which is a chosen parameter). In each node in the chosen level, it either explores the parent (the root) of that node or the children originated from that node by calling the appropriate subroutines to either "go up" or "go down" the tree.

It can be seen that the level traversal tree search is much more efficient than the depth first tree search because it explores only the "promising" region. However, the fact that the current bests are found in the left side of the tree plays a significant part in reducing computational time because the tight bounds helps eliminate lots of non-optimal solutions. If tight bounds are not obtained (in this context, when current bests are found in the right side of the tree), the level traversal tree search basically has to explore the whole level of tree, which makes it impossible to solve large scale problems efficiently using this method. The large scale problem with medium level of specification shown in chapter 2, the TE example case study 2, exposes such limitation of the level traversal tree search. We attempted to solve the TE example using level traversal tree search but the solutions provided by this method are worse than the equation-based method with decomposition described in chapter 2 even after several days running. For the TE example, a kind of heuristic local search or approximate method is probably the most efficient method. This approximate method is shown in the next section.

3.7. Approximate method

This section presents an approximate (local search) method that improves the equation-based method by complementing it with a local search. The idea is to use the good solutions provided by the equation-based method as input in a local tree search procedure to hopefully arrive close to the global optimum (the term "good" solution is meant to be a feasible solution with objective value / cost near to that of the global optimum)

The core of our methodology, the local search, relies on the following observation:

- The global optimal solution and near-optimal solutions belong to the same region (space of variables), that is, they are different from one another in values of only a few variables (in this context, the measurement locations). This is due to the inherent characteristic of the sensor network: if a measurement is good (because the associated sensor is cheap and this measurement contributes significantly to the observability and redundancy of key variables), then it will show up in global optimum and some other "good" solutions. Compared to the global optimum, a good solution usually misses one or two good measurements and contains some other "extra" measurements
- Therefore, it is reasonable to assume that all the measurements show up in good solutions are good measurements that are very likely to show up in global optimum

Based on the above arguments, the proposed heuristic local search has the following two steps:

Step one:

The purpose of this step is to find a minimum cost (and feasible) solution constituted from those "good" measurements. The calculation procedure to find such a solution is the following:

- Find the union of the last five current best solutions (find all good measurements that show up the last five current best solutions), denoted as vector ***U***
- Employ a tree enumerative strategy to remove measurements out of vector ***U*** to obtain a minimum cost solution (the method is essentially the same as the inverted tree search described in Nguyen & Bagajewicz, 2008). Let us denote that the minimum cost solution as ***MC.***

If all good measurements belonging to global optimum actually show up in the last five current best solutions (which is highly probable), the identified minimum cost solution ***MC*** is indeed global optimum. However, there is a very small chance that one or two good measurements belonging to global optimum do not show up in vector ***U***, thus we go on to step two to account for such situation.

Step two

The purpose of this step is to identify if it is possible to improve the solution by replacing a certain number of measurements in ***MC*** by some other measurements not belonging to ***MC*** (denote all the measurements not belonging to ***MC*** as vector ***A***). This is

done by exploring all the possibilities of replacing a certain number of measurements (denoted as *Nr,* a parameter) in ***MC*** with elements in vector ***A*** trying to obtain a solution better than ***MC***. The calculation is as follows:

- Remove a certain set of *Nr* measurements out of ***MC***, denote the resulting vector as ***B***
- Use a tree enumerative strategy to add elements (measurements) from ***A*** into ***B*** trying to obtain feasible solutions with minimum cost.
- Remove another set of *Nr* measurements out of ***MC***, obtain the new vector ***B*** and repeat the same procedure
- Terminate the process when all the possibilities of removing *Nr* measurements out of ***MC*** are explored

The two steps procedure is illustrated in Figure 3.7

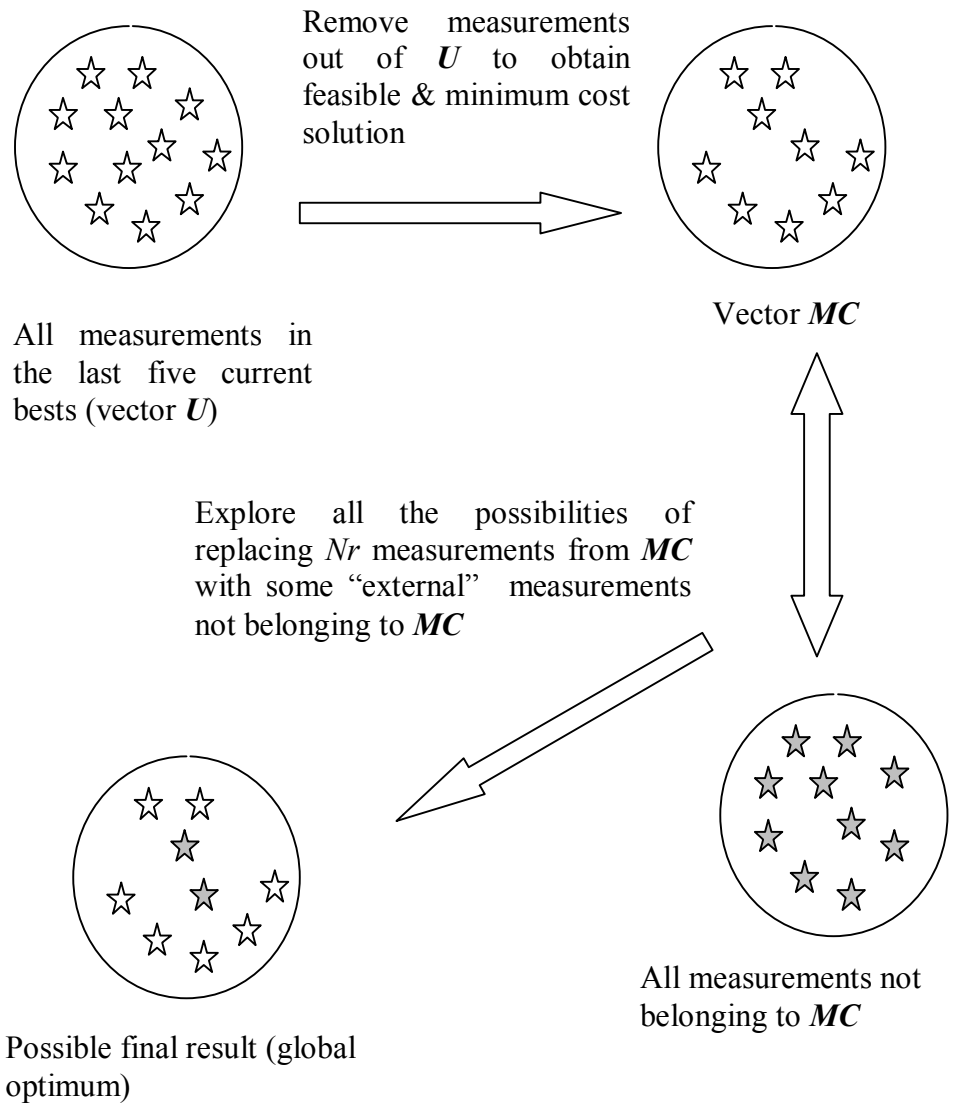


Figure 3.7 - Approximate (heuristic local search) method

3.8. Illustrated example – Approximate method

The proposed approximate method is implemented in Fortran running on a 2.8 GHz Intel Pentium 1028 MB RAM PC. One example is provided

Example 3.2: The TE process example (introduced in chapter 2) is used. The same process flowsheet (shown in figure 2.10) and data (tables 2.10 and 2.11) are used.

Three design case studies are considered, which are described in table 3.4. The first design case is the one with moderate specification described in chapter 2. The other two design cases are ones with high level of specifications.

Table 3.4 - Design case studies for the TE process example

Design case	TE1 Moderate Spec.	TE2 High Spec.	TE3 High Spec.
No. of key variables	17	19	23
Key variables	F_6, y_{A6}, y_{G6}, y_{H6}, F_7, y_{G7}, y_{H7}, y_{A9}, y_{G9}, y_{H9}, F_{11}, y_{G11}, y_{H11}, P_r, T_r, P_s, T_s	F_6, y_{A6}, y_{B6}, y_{G6}, y_{H6}, F_7, y_{A7}, y_{B7}, y_{C7}, y_{A8}, y_{B8}, y_{C8}, y_{D8}, y_{A9}, y_{B9}, y_{C9}, F_{11}, y_{G11}, y_{H11}	F_6, y_{A6}, y_{B6}, y_{G6}, y_{H6}, F_7, y_{A7}, y_{B7}, y_{C7}, y_{G7}, y_{H7}, y_{A8}, y_{B8}, y_{C8}, y_{D8}, y_{A9}, y_{B9}, y_{C9}, y_{G10}, y_{H10}, F_{11}, y_{G11}, y_{H11}
Requirement	Observability	Redundancy	Redundancy
Precision thresholds	2%	1.5%	1.5%
Residual precision thresholds		4%	4%
Measured variables	F_6, y_{A6}, y_{G6}, y_{H6}, F_7, y_{A7}, y_{A9}, y_{G9}, y_{H9}, F_{11}, y_{G11}, P_s (12 sensors)	All variables but $\{y_{C6}$, y_{E6}, y_{F6}, y_{E9}, y_{F9}, y_{E10}, y_{F10}, y_{G10}, y_{H10}, P_r, T_r, P_s, $T_s\}$ (34 sensors)	All variables but $\{y_{C6}$, y_{E6}, y_{F6}, y_{E9}, y_{F9}, F_{10}, y_{E10}, y_{F10}, P_r, T_r, P_s, $T_s\}$ (35 sensors)
Sensors cost	7,070	23,560	24,810

Except for design case studies with low level of specifications (where feasible solutions contain only a small fraction of available candidate sensors), the TE example cannot be solved by using the individual measurement-based tree search (described in Bagajewicz, 1997) in a reasonable computational time. Indeed, it is estimated that, if moderate or high level of specification is required, solving the TE example by individual measurement-based tree search takes as long as several weeks. Equation-based tree search method coupled with decomposition (called Decomposed Equations method, presented in chapter 2) is the only viable option for design cases with moderate level of specifications while design cases with high level of specifications can be solved by using either inverted tree strategy or Decomposed Equations method (as shown in chapter 2).

The Decomposed Equations method was first used to obtain several good solutions as starting point, then use local search to arrive at optimum solution. The specifics of the approximate method as applied to our example are:

- The Decomposed Equations method was first used to solve the problem, which is terminated after 100,000 nodes are explored. All the current best solutions (feasible ones) are recorded.
- The last five current best solutions are used as input ("good" measurements) to the local search procedure described above.
- As stated above, the local search includes two steps:

    i. Removing sensors out of the vector containing all "good" measurements to arrive at the new current best ***MC*** (most likely to be optimal solution),

ii. Exploring all the possibilities of replacing *Nr* measurements in ***MC*** with some other measurements not belonging to ***MC***. We use *Nr*= 2

For the two last design cases (high specification), the inverted tree search method described in Nguyen & Bagajewicz (2008) was used to validate the solutions obtained by the proposed approximate method, which is a combination of the Decomposed Equations method and the local search.

For the first design case, its solution was validated by using the level-by-level "L by L" tree search described in above section. More specifically, the "L by L" tree search was used to explore the level containing 12 sensors (same number of sensors with that of TE1's solution obtained by approximate method); which found no better solution. Thus, the combination of the Decomposed Equations method and the local search is able to find optimal solutions for the TE problem.

The computational performance of the approximate method is shown in table 3.5

Table 3.5 - Performance of the approximate method, TE process example

		TE1 Moderate Spec.	TE2 High Spec.	TE3 High Spec.
Number of nodes explored	Step 1	48,544	484	212
	Step 2	54,097	130,683	166,244
	Total	202,641	231,167	266,456
Total computational time		1hr 12 min	1hr 33 min	1hr 40 min

- For design case studies TE2 & TE3: after exploring 100,000 nodes, the Decomposed Equations method identifies four current best solutions that contain 39, 37, 38 and 36 sensors respectively (costs are 26990, 26290, 26240 and 25540). The union of these solutions (vector ***U***) contains 39 sensors, which is exactly the same as the first current best identified (this means that the first current best contains all "good" measurements). Exploring all the possibilities of removing sensors out of vector ***U*** results in the optimal solutions (containing 34 and 35 sensors with costs being 23,560 and 24,810 for design cases TE2 and TE3 respectively).
- For design case TE1: after exploring 100,000 nodes, the Decomposed Equations method identifies 11 current best solutions; the last five solutions having cost ranging from 11840 to 13370 and number of sensors ranging from 20 to 22. The union of the last five solutions (vector ***U***) contains 23 sensors. Using enumerative tree search strategy to remove sensors out of ***U*** (exploring 48,544 nodes) results in the optimal solution that contains only 12 sensors costing 7070. This solution is much better than the current best solution obtained by using the Decomposed Equations method only (that solution, described in chapter 2, contains 17 sensors whose cost is 9630). Note that, as stated in chapter 2, because the Decomposed Equations tree search was terminated halfway, the obtained current best solution is not guaranteed to be optimal solution. That solution is indeed confirmed to be a sub-optimal solution in this work

In all the testing problems we have tried, the combination of the Decomposed Equations method and step 1 is able to locate optimum solution. The use of step 2, which is a safeguard step to avoid the possibility of missing optimum solution, somehow "guarantees" optimality. We discuss some of this issue next:

Let us denote the measurements contained in the optimal solution as optimal measurements (in the opposite side, the rest are called non-optimal measurements). The optimal solution is missed only if the following two situations occur simultaneously:

i. The optimal solution is missed, that is, the current best ***MC*** is not optimal solution (which means that ***MC*** contains some (=*Nt*) non-optimal measurements)

ii. The number of non-optimal measurements in ***MC*** (*Nt*) is more than the number of measurements we consider removing out of ***MC*** (*Nr*).

Table 3.6 concludes this chapter. This table is the table 3.1 updated with the two new methods presented in this chapter, where the measurement-based tree search is replaced by the level-by-level tree search and the approximate method is used when short computational time is preferential over finding optimal solution

Table 3.6 - Most suitable method for solving sensor network design problem

Level of specifications	Linear systems	Nonlinear systems	
		Focus on finding optimal solution	Focus on computational time
Low	Cutsets-based or level-by-level search	Equations-based or level-by-level search	Equations-based or Approximate method
Medium	Cutsets-based	Equations-based or level-by-level search	Approximate method
High	Cutsets-based or Inverted tree search	Equations-based or Inverted tree search	Approximate method or Inverted tree search

3.9. Conclusions

In this chapter, two efficient methods for solving nonlinear SNDP are presented:

1. The level traversal method helps reduce computation time of the depth first tree search by skipping the non-feasible region and intelligently disregarding the non-optimal solutions via notion of families of nodes. The method is very efficient if feasible solutions are found in the left hand side of the tree.
2. The approximate method is a combination of Decomposed Equations method (presented in chapter 2) and heuristic local search method. This method is very efficient: it is able to solve nonlinear large scale problems within a couple of hours. Although it does not guarantee optimality, the chance of finding global optimal solution is very high. Indeed, the proposed method was able to find optimal solution in all three design case studies shown in this paper.

3.10. References

Bagajewicz, M., Design and Retrofit of Sensors Networks in Process Plants. *AIChe J.* **1997**, 43(9), 2300-2306.

Diwekar U. *Introduction to Applied Optimization* (*Springer Optimization and Its Applications*) 2nd Ed, Springer: NY, USA, **2008**

Gala, M. and Bagajewicz, M. J. Rigorous Methodology for the Design and Upgrade of Sensor Networks Using Cutsets. *Ind. Eng. Chem. Res*. **2006a**, 45(20), 6687-6697.

Gala, M. and Bagajewicz, M. J. Efficient Procedure for the Design and Upgrade of Sensor Networks Using Cutsets and Rigorous Decomposition. *Ind. Eng. Chem. Res*. **2006b**; 45(20), 6679-6686.

Nguyen D.Q. and Bagajewicz M. Design of Nonlinear Sensor Networks for Process Plants. *Industrial and Engineering Chemistry Research*. **2008**, 47(15), 5529-5542

4. VALUE-PARADIGM SENSOR NETWORK DESIGN

Traditional cost-optimal approach to design sensor networks requires expertise knowledge of the users to use appropriate specifications in the model. This chapter presents a new approach to design sensor network. This approach, based on the concept of value of accuracy developed by Bagajewicz (2006), allows the simultaneous optimization of cost and performance of sensor network. Efficient methods to solve the problem are also proposed

4.1. Overview

All the published work on SNDP for *process monitoring* purpose focused on finding more efficient computational methods to solve the problem, where the sensors cost is minimized and the popular specifications on precision, residual precision, error detectability and resilience and estimation reliability are used as performance targets. Recently, Bagajewicz (2005a) introduced the concept of software accuracy that essentially encompasses all the aforementioned performance measures. The economic value of software accuracy was also quantified (Bagajewicz, 2006) and an efficient approximate method was developed to evaluate the economic value of accuracy (Nguyen et al., 2006).

This chapter presents a new approach to design sensor networks that maximizes the economic value of accuracy (named value-optimal SNDP). Relationship between this

new approach and the traditional cost-optimal approach is discussed and efficient methods to solve the problem are presented.

This chapter is organized as follows: firstly the concept of software accuracy and economic value of accuracy is briefly reviewed, followed by description of computational methods to evaluate software accuracy and its associated economic value. The value-optimal SNDP and efficient methods to solve the proposed problems are then presented.

4.2. Software accuracy

Accuracy was conventionally defined as precision plus bias (Miller, 1996). However, the definition is of little practical use because bias size is generally unknown. Recently, Bagajewicz (2005a) introduced the concept of software accuracy in the context of data reconciliation and gross error detection being used to detect biases. In such context, accuracy was defined as sum of precision and induced bias instead of the actual bias. The induced bias and the software accuracy are shown next (Bagajewicz, 2005a):

$$\hat{\delta} = E[\hat{x}] - x = [I - SW]\delta \tag{4-1}$$

$$\hat{a}_i = \hat{\sigma}_i + \delta_i^* \tag{4-2}$$

where $\hat{a}_i, \hat{\sigma}_i, \delta_i^*$ are the accuracy, precision (square root of variance S_{ii}) and the induced bias of the estimator, respectively.

By definition, the accuracy value relies on how one calculates the induced bias. From Eq. (4-1), it is clear that the induced bias is the function of undetected biases whose sizes can be any value below the threshold detection values and their location can be anywhere in the system. Thus, the induced bias is a random number. Bagajewicz

(2005a) proposed to calculate the induced bias as the maximum possible value. Recently, Bagajewicz (2005b) and Bagajewicz and Nguyen (2006) proposed to calculate the induced bias as the expected value of all possible values, which is more realistic, and used a Monte Carlo simulation – based procedure to obtain such expected value.

4.3. Economic value of accuracy

Bagajewicz et al. (2005) presented the theory of economic value of precision and developed formulas for assessing downside financial loss incurred by production loss. They argued that, due to inaccuracy (caused by random errors) of the estimator of a product stream flowrate, there is a finite probability that the estimator is above the target but in fact the real flow is below it. In such situation and under the assumption that the operators did not make any correction to the production throughput set point when the estimator suggested that the targeted production has been met or surpassed, the production output will be below the target and financial loss occurs. The financial loss under simplified assumptions of negligible process variations and normal distributions of the process variation and the measurements was found to be $DEFL = 0.19947*Ks*T*\hat{\sigma}_p$ where Ks is the cost of the product (or the cost of inventory) and T is the time window of analysis (Bagajewicz et al., 2005).

Using the same concept of downside financial loss, Bagajewicz (2006) extended the theory of economic value of precision to include the effect of (induced) bias, namely the economic value of accuracy. The expression for financial loss *DEFL* considering bias is given by (Bagajewicz, 2006):

$$DEFL = \Psi^0 DEFL^0 + \sum_i \Psi_i^1 DEFL^1\Big|_i + \sum_{i1,i2} \Psi_{i1,i2}^2 DEFL^2\Big|_{i1,i2} + .. \sum_{i1,i2,..,iN} \Psi_{i1,i2,..,iN}^n DEFL^N\Big|_{i1,i2,..,iN} \quad (4\text{-}3)$$

In this equation, $\Psi_{i1,i2,..,iN}^n$ and $DEFL^N\Big|_{i1,i2,..,iN}$ are the average fraction of time the system is in the state containing n gross errors $i_1, i_2,..,i_N$ and its associated financial losses, respectively. Detail expression and procedure to calculate the financial loss for system containing n biases $i_1, i_2,..,i_N$ can be found in Nguyen et al. (2006)

Applications of the theory of economic value of precision/accuracy for the determination of economical benefit of instrumentation upgrade were shown by Bagajewicz et al. (2005) and Bagajewicz (2006). The economical benefit of an instrumentation upgrade was calculated as the difference in downside financial loss (*DEFL*) before and after such upgrade. The net present value of instrumentation upgrade (*IU*) was then given by:

$$NPV = d_n \{DEFL(before\ IU) - DEFL(after\ IU)\} - \text{cost of } IU \quad (4\text{-}4)$$

where d_n is sum of discount factor for n years. The cost can be the cost of purchasing of new sensor (when adding new sensors) or the cost of license (when installing data reconciliation software). A large value of the net present value of instrumentation upgrade may justify this type of investment. Case studies on the value of performing data reconciliation as well as savings of adding new sensors at selected locations to the sensor network of a crude distillation unit were provided by Bagajewicz et al. (2005) and Bagajewicz (2006).

It has been also shown that the financial loss without bias $DEFL^0$ is smaller than financial loss in the presence of biases $DEFL^1\big|_i, DEFL^2\big|_{i1,i2}, \ldots$ (Nguyen et al., 2006). Looking at the complete expression for financial loss (Eq. 4-3), it is obvious that if one is to reduce financial loss, one can either directly reduce the individual financial loss (i.e., $DEFL^0, DEFL^1\big|_i, DEFL^2\big|_{i1,i2} \ldots$) by instrumentation upgrade, or one can increase the fraction of time that the system is in the state containing no biases Ψ^0 (as a result, the fractions of time that the system is in the state containing biases $\Psi_i^1, \Psi_{i1,i2}^2, \ldots$ are reduced). This is where maintenance policies come into play because different maintenance schemes of sensor system affect the aforementioned fractions of time.

4.4. Computational methods to evaluate software accuracy and economic value of accuracy

The financial loss $DEFL^N\big|_{i1,i2,..,iN}$ corresponding to the presence of a specific set of gross errors $i_1, i_2,..,i_N$ can be evaluated using two methods: approximate method and Monte Carlo simulation as detailed in Nguyen et al. (2006); upon which the financial loss of a sensor network (*DEFL* in Eq. 4-3) is evaluated. The expected value of accuracy, which is the mean value of all possible values of accuracy, is a more realistic value than the maximum possible value (which is too conservative). Similar to the financial loss (*DEFL*), this expected value of accuracy can be evaluated using two methods: approximate method and Monte Carlo simulation. The Monte Carlo simulation-based procedure to evaluate the expected value of accuracy, termed stochastic accuracy, is detailed in Bagajewicz and Nguyen (2008) (the approximate method to calculate

expected value of accuracy is not described in any published paper, but it is very similar to the approximate method to calculate financial loss, which was described in Nguyen et al., 2006).

The Monte Carlo simulation procedure to calculate the stochastic accuracy and financial loss was described in Bagajewicz & Nguyen (2008) and Nguyen & Bagajewicz (2009), respectively. This method takes longer computational time than the approximate method so it is mainly used to validate the results obtained by the approximate method

The principle of the approximate method is to partition the space of variables into several sub-spaces. In some sub-spaces the expression for financial loss (and accuracy value) can be evaluated analytically while in the others the expression has to be evaluated approximately (Nguyen et al., 2006). The partition of the space of variables is illustrated in Figure 4.1 in the case two biases are present in the system.

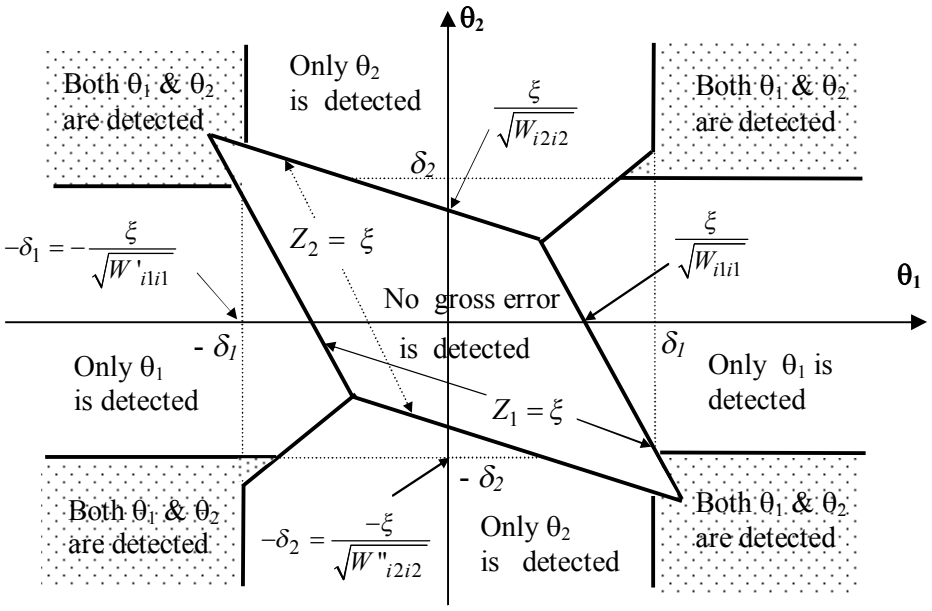


Figure 4.1 - Different regions when two gross errors are present in the system

In the region where both biases are detected, the expression for financial loss can be calculated analytically while in the others, an approximate scheme is used to evaluate the expression (Nguyen et al., 2006).

The better the gross error detection capability of the network (which means the smaller the area of the rhombus shown in figure 4.1), the smaller the expected value of accuracy (and financial loss) is.

4.5. Dependence of software accuracy and the associated economic value on sensor network

Because software accuracy is defined as precision plus induced bias, the requirement on accuracy value encompasses the requirements on precision, gross errors delectability and gross errors resilience. More specifically, a sensor network that renders good (small) software accuracy for variables of interest needs to possess all of the followings:

i. Good precision of estimators of key variables

ii. Good level of redundancy (i.e. enough measured variables) to detect biases so that undetected biases would have small magnitudes; this property is directly related to gross errors delectability

iii. Smearing effect of undetected biases on estimators of key variables is limited (such that estimation accuracy is small even though undetected biases are large); this property is directly related to gross errors resilience.

These needed network's capabilities generally require a good level of hardware redundancy (i.e. more sensors than the number of key variables). To improve estimation accuracy, it is usually needed to use more sensors. The same thing is stated for financial loss, that is, sensor network would have small financial loss if it possesses the three aforementioned properties and it is necessary to use more sensors to reduce financial loss.

The exception to this generalization does exist. Indeed, there exists situation in which the undetected biases are very large, for example, two gross errors cancel out each other such that these two biases are undetected (by using measurement test) no matter how big they are. This phenomenon is known as gross errors equivalency (Bagajewicz and Jiang, 1998). An example for the case of gross errors equivalency is shown next.

Consider the system shown in figure 4.2. The two biases in S_2 and S_3 can not be detected (no matter how big they are) if they are equal but in opposite sign (since the material balance is satisfied in such case). The region of undetected biases for such case is shown in figure 4.3. Note that, practically, gross errors are not unbounded. If a bias in a measurement passes a certain threshold, which is usually a certain percentage of the normal value of the variable, by common sense the operators can tell that there is bias in the measurement.

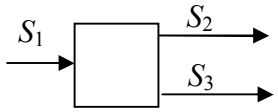


Figure 4.2 – Illustrated example

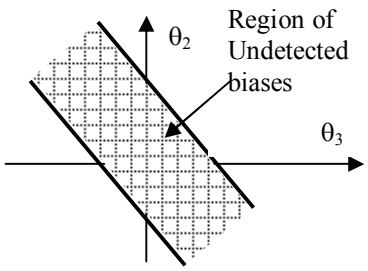
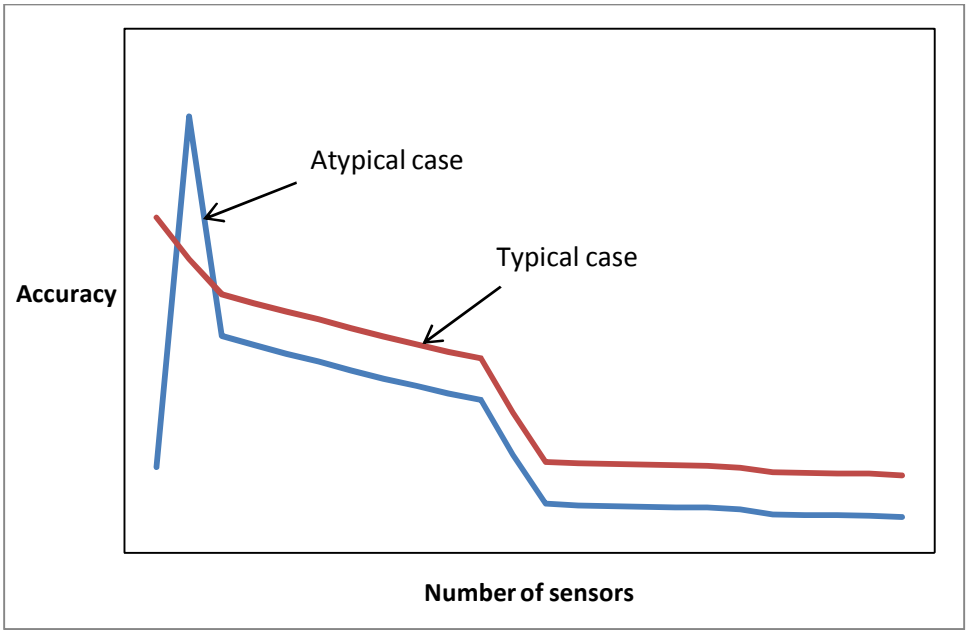


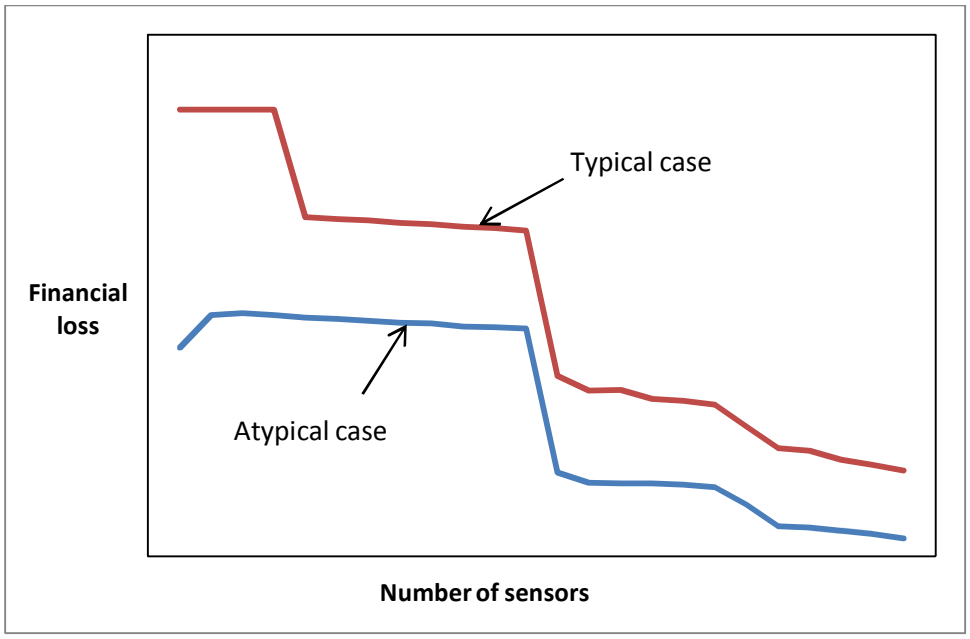
Figure 4.3 – Illustration of biases equivalency

Thus, if the newly added measurement forms such a set of gross errors with existing measurements (while the original network does not have), the software accuracy and financial loss increase when that new measurement is added to the network.

The typical case as well as the irregular case of software accuracy and financial loss as function of the number of measurements is shown in figure 4.4



4a. Accuracy vs. number of sensors



4b. Financial loss vs. number of sensors

Figure 4.4 – Accuracy and financial loss as function of number of sensors

If typical value for sensor precision (2%) is used, typical range for accuracy is from 2% to 20% (of nominal value of measured data). Financial loss does not have typical range because it depends strongly on the economic parameters, which are K_s (cost of the product or cost of inventory) and T (time window of analysis)

As can be seen from figure 4.4:

- The jumps (steep slopes) in figure 4.4 corresponding to the case where the newly added measurement contributes significantly the process monitoring capabilities of the sensor network (e.g. observability and redundancy of key variables). On the other hand, if "meaningless" measurement (that contributes almost nothing to the process monitoring capabilities of the sensor network) is added, the accuracy and financial loss are almost unchanged.

- Generally, adding sensors improves accuracy and financial loss
- If bias in the newly added sensor is very difficult to be detected, accuracy and financial loss would increase when adding that sensor. However, continue adding more sensors would again improve accuracy and financial loss

4.6. Accuracy and value-optimal SNDP

4.6.1. Accuracy-constrained SNDP

Software accuracy can be used as a constraint in the commonly used cost-optimal SNDP (equation 1-6). The problem formulation for accuracy-constrained SNDP is obtained by adding constraint on accuracy to equation (1-6):

$$\begin{aligned} & Min \sum_{\forall i} c_i q_i \\ & s.t. \\ & \sigma_i(\boldsymbol{q}) \le \sigma_i * \qquad \forall i \in M_S \\ & a_i(\boldsymbol{q}) \le a_i * \qquad \forall i \in N_S \\ & q_i = 0,1 \qquad \forall i \end{aligned} \tag{4-5}$$

where $a_i(\boldsymbol{q})$ and a_i^* are accuracy of key variables and the associated threshold values; $\boldsymbol{N_S}$ represents the set of variables where specification on accuracy is required

The problem (4-5) can be readily solved by using any suitable branch and bound method developed in our group (see table 3-6)

4.6.2. Value-optimal SNDP

The proposed sensor network design formulation is as follows:

$$\begin{aligned} &\text{Max}\left\{V(\boldsymbol{q})-c(\boldsymbol{q})\right\} \\ &s.t. \\ &\sum_{i=1}^{n_i} q_i \le \text{m} \\ &c(\boldsymbol{q}) \le \text{b} \end{aligned} \tag{4-6}$$

n_i is the number of candidate sensors (number of process variables under considerations); $V(\boldsymbol{q})$ is value of sensor network (function of measurement locations $\boldsymbol{q}$), $c(\boldsymbol{q})$ is cost of sensors; m is limit on number of sensors and b is limit on budget.

If limit on number of sensors and budget limit are not used, the problem becomes an unconstrained optimization problem:

$$\text{Max}\left\{V(\boldsymbol{q})-c(\boldsymbol{q})\right\} \tag{4-7}$$

The value of a sensor network is given by

$$\boldsymbol{V}(\boldsymbol{q}) = \left\{\boldsymbol{DEFL}(\text{no sensor}) - \boldsymbol{DEFL}(\text{with sensors})\right\} = \boldsymbol{RDEFL} - \boldsymbol{DEFL}(\boldsymbol{q}) \tag{4-8}$$

The financial loss when there is no sensor is a large value, denoted as *RDEFL* (a reference value). Equation (4-8) becomes

$$\text{Max}\left\{RDEFL - \{DEFL(\boldsymbol{q}) + c(\boldsymbol{q})\}\right\} \tag{4-9}$$

Thus, maximizing value minus cost is equivalent to minimizing financial loss plus cost of sensor network

$$\text{Min}\{DEFL(\boldsymbol{q}) + c(\boldsymbol{q})\} \tag{4-10}$$

This is an unconstrained optimization problem with complicated surface of objective function. When the number of sensor increases, cost $c(\boldsymbol{q})$ increases but the financial loss $DEFL(\boldsymbol{q})$ generally decreases as shown in figure 4.5

The best situation is when the objective function exhibits a single global minimum (as shown by line A in figure 4.5). Unfortunately, the objective function (financial loss plus cost) is a complicated function of the sensor network (vector $\boldsymbol{q}$) and usually has many "hills" and "valleys", that is, it usually has many extrema (as shown by line B in figure 4.5)

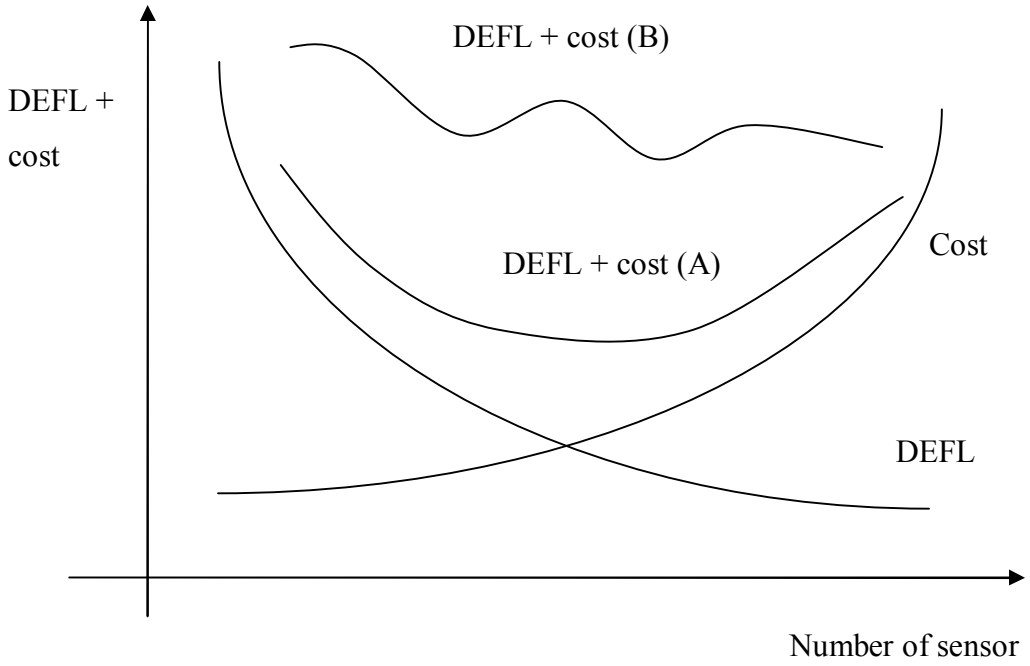


Figure 4.5 – Objective function vs. dependent variables ($\boldsymbol{q}$)

The most popular method to solve unconstrained optimization problem is to use the KKT condition (contact condition). Unfortunately, the objective function is not a

explicit function of dependent variable (***q***); in fact, numerical method in the form of approximate method or Monte Carlo method must be used to calculate the objective function (more specifically, the financial loss). Thus, the only applicable method is the "searching" method. Two "searching" methods are considered: tree enumeration method and Genetic Algorithm, a very popular stochastic approach to solve combinatorial optimization problem.

4.7. Illustrated example of accuracy-constrained SNDP and value-optimal SNDP

4.7.1. Example 4.1

Consider the following process example, which was introduced in chapter 1 (example 1.1). The process flowsheet is shown in figure 4.6, the data for the example is shown in table 4.1 (sensor precision = 2% for all sensors). Sensor failures are assumed to occur with probability 0.2 (third column), the biases associated to these failures are assumed to follow normal distributions with zero means (fourth column) and standard deviations given in fifth column

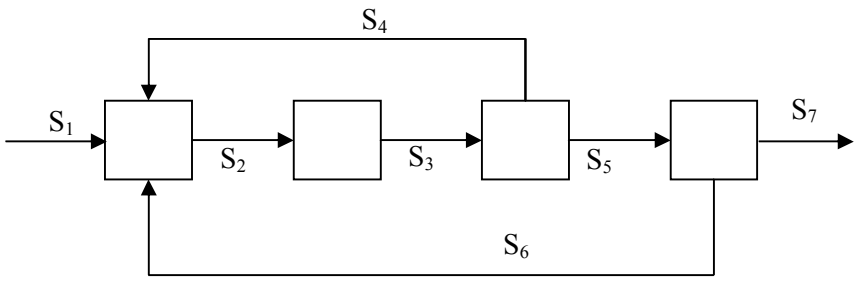


Figure 4.6 – Example process

Table 4.1- Data for example 4.1

Stream	Flow rates	Costs	Prob. of failure of sensor	Mean of pdf of bias	STD of pdf of bias
S_1	100	55	0.2	0	8
S_2	140	40	0.2	0	11.2
S_3	140	60	0.2	0	11.2
S_4	20	50	0.2	0	1.6
S_5	120	45	0.2	0	9.6
S_6	20	55	0.2	0	1.6
S_7	100	60	0.2	0	8

For this small illustrated example, optimal solutions are obtained by using exhaustive tree search without stopping criterion (totally there are 127 candidate solutions).

Accuracy-constrained SNDP

The sensor network design problem requesting a satisfactory accuracy value of key variables (equation 4-5) is illustrated next. The results are shown in table 4.2

Table 4.2- Results for example 4.1, accuracy-constrained SNDP

Case Study	**4.1.1a**	**4.1.1b**	**4.1.1c**	**4.1.1d**	**4.1.1e**
Key variables	S_1 & S_5	S_1 & S_5	S_1 & S_5	S_1 & S_5	S_1 & S_5
Accuracy thresholds	4	3	2	1.8	1.5
Accuracy value	$a_{S1} = 3.36$ $a_{S5} = 2.85$	$a_{S1} = 2.22$ $a_{S5} = 1.99$	$a_{S1} = 1.90$ $a_{S5} = 1.81$	$a_{S1} = 1.65$ $a_{S5} = 1.49$	$a_{S1} = 1.499$ $a_{S5} = 1.27$
Measured variables	S_1, S_6	S_1, S_5, S_6	S_1, S_6, S_7	S_1, S_5, S_6, S_7	All variables
Sensors cost	110	155	170	215	365

The design specifications are shown in rows 2 & 3 of table 4.2, the optimal solutions are shown in row 5 (optimal measurement placement) and row 6 (optimal cost). The accuracy values of key variables corresponding to the optimal solutions are shown in row 4.

From design case 4.1.1a to 4.1.1e, the desired value of accuracy decreases (from 4.0 to 1.5), which requires more sensors to be used. In design case 4.1.1.a, basically only observability is required for the two key variables S_1 & S_5. In design cases 4.1.1b and 4.1.1c, the obtained optimal solutions render redundancy of level one for key variables (the two key variables are still observable if one removes any one sensor out of the three-sensor solutions). When a smaller accuracy threshold (design case 4.1.1d) is required, both redundancy (of key variables) and gross error detection capability of the network are required; hence more sensors need to be used. It can be seen that the optimal solution in this design case is the same as the solution obtained in design case 1.1c (column 4, table 1.2) where both estimation redundancy and gross error detection capability are required. Design case 4.1.1d is the extreme case where the required accuracy threshold is so small such that all sensors need to be used to meet the requirement.

Value-optimal SNDP

The sensor network design problem simultaneously minimizing financial loss and cost of a sensor network (equation 4-10) is illustrated next. This problem does not have any constraint. The economic parameters used in the expressions to evaluate financial loss are as follows: the time window of analysis T is 30 days (this is based on the argument that, by mean of production accounting calculation every month, one can detect

the loss in production that has been covered by biased measurement); the cost of product *Ks* (or cost of inventory) for the two key variables S_1 & S_5 are shown in row 3 of table 4.3. The financial losses of the optimal sensor networks are shown in row 6 of the table

Table 4.3- Results for example 4.1, value-based SNDP

Case Study	**4.1.2a**	**4.1.2b**	**4.1.2c**	**4.1.2d**
Key variables	S_1 & S_5	S_1 & S_5	S_1 & S_5	S_1 & S_5
Ks value	$Ks_1 = 2$ $Ks_5 = 2$	$Ks_1 = 10$ $Ks_5 = 10$	$Ks_1 = 30$ $Ks_5 = 20$	$Ks_1 = 60$ $Ks_5 = 50$
Measured variables	S_1, S_6	S_1, S_6, S_7	S_1, S_5, S_6, S_7	all
Sensors cost	110	170	215	365
Financial loss	78.6	219.9	451.8	824.9

As can be seen from table 4.3:

- When the cost of product K_s increases, financial loss increases
- In the value-optimal SNDP problem, cost and financial loss are simultaneously minimized. If *Ks* is small, the cost factor dominates financial loss factor and the optimal network contains few sensors. In the opposite site, if K_s is large, the financial loss term dominates the cost term and the optimal network contains large fraction of candidate sensors so as to minimize financial loss. This means that if K_s increases, then the number of sensors in optimal network increases as evidenced in table 4.3
- In design case 4.1.2a, K_s is small, cost needs to be minimized and the optimal network contains only enough sensors to guarantee observability of key variables

- In design case 4.1.2d, K_s is large, financial loss needs to be minimized and the optimal network contains all sensors. This is an extreme case
- In design cases 4.1.2b & c, K_s is moderate, the optimal networks contains enough sensors that can guarantee some degree of estimation redundancy and gross error detection capability. The optimal networks in these two design cases are the same as the networks obtained in the two design cases 4.1.1c & d, which have good process monitoring capability (good accuracy value and good gross error detection capability) as shown in columns 4 and 5 of table 4.2 and columns 1 and 3 of table 1.2

The rest of this chapter focuses on the efficient methods to solve the value-optimal SNDP (as mentioned above, the accuracy-constrained SNDP is a constrained optimization problem that can be readily solved by using any appropriate branch and bound method shown in table 3-6)

4.8. Genetic Algorithm

The proposed optimization problem (equation 4-10) is amenable to standard genetic algorithm because:

- The problem is a combinatorial optimization problem involving binary variables.
- There is no constraint.
- The objective function is a complicated function with many extrema.

Genetic Algorithm (GA) is used because this method is well-established and was shown to have good performance (although it does not guarantee optimality). In brief, Genetic Algorithm method is based on the principles of genetics, natural selection and evolution; it "allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the "fitness", i.e. minimizes the cost function" (R.L. Haupt and S.E. Haupt, 2004). The algorithmic procedure and detailed description of the well-known Genetic Algorithm method can be found in various textbooks such as the Haupts' book (2004).

The GA is briefly described as a seven-step procedure as follows:

1. Variable Encoding and Decoding: this step involves the conversion (i.e. encoding) of the values of decision variables into an appropriate representation (a chromosome). If the type of decision variables and the type of GA are the same (e.g. binary variables – binary GA, which is our case) then no conversion is needed: the values of decision variables are copied directly into the chromosomes. Decoding is the reverse process of encoding, which is the conversion from binary representation into real values of variables so that the cost function (i.e. objective function) can be evaluated
2. Initialization of population: this step involves randomly generating a population of N chromosomes. For binary GA, this is done by using uniform distribution to generate random binaries. The size of population, N, is a GA parameter.
3. Natural selection: this step involves three operations: i) evaluating the cost function corresponding to each chromosome / individual in the population, ii)

sorting the population in descending order of "fitness" (e.g. if the cost function / objective value is to be minimized, then lower cost = larger "fitness" value), iii) selecting a portion of population with good fitness value to keep and discarding the rest, usually half of the population (the lower half in the sorted list of chromosomes) will be discarded.

4. Selection: selecting and pairing the retained (survived) chromosomes to produce offspring for the next generation. Usually two chromosomes are paired to produce two offspring. There exist many methods for this operation, one of the most commonly used method is the roulette wheel selection.
5. Mating: offspring of the paired chromosomes (parent) are produced through the crossover process whereby the parent's genetic codes are passed on to the offspring.
6. Mutation: random mutations alter a certain percentage of the bits in the list of chromosomes. Mutation is the second way the GA method explores a cost surface and avoids the trap of local optima. It introduces traits not in the original population and keeps the GA from converging too fast before sampling the entire cost surface. Mutation points are randomly selected from the population; with each mutation point, changing a 1 to a 0 and visa versa. The number of mutation points is defined by mutation rate, which is the fraction of the number of mutation points divided by the total number of bits in the population.
7. Convergence: after the mutation step, a next generation population is generated which contains new chromosomes (i.e. new candidates for optimal solution to

evaluate). The same procedure of evaluating cost functions - selecting - pairing - producing offspring (the steps from natural selection to mutation) is repeated unless convergence criterion is met, which is to terminate the GA procedure if the best objective value obtained in each iteration does not change after a predetermined number of iterations.

The parameters involved in the GA method are the size of population, the portion of population to keep, the mutation rate and the selection and crossover methods. The methods for the GA operators and the values are intuitively chosen in accordance with the scale of the problem using the guidelines provided in the literature (R.L. Haupt and S.E. Haupt, 2004). They are as follows:

- Selection: roulette wheel selection method
- Crossover: two-point crossover method
- Population size = 20
- Fraction of population to keep = 0.5
- Mutation rate = 0.2

4.9. Cutset-based tree search method

The calculation procedure is described below:

1. Find all the cutsets of the process graph.
2. Consider only cutsets that contain at least one key variable, put them to a list of cutsets

3. Remove key variables out of the cutsets in the list and consider them as separate cutsets, e.g. if [1 2 3 4] is a cutset and "1" and "3" are key variables then consider [1], [3] and [2,4] as separate cutsets
4. Sort these cutsets in ascending order of their cost (cost of a cutset is equal to sum of the costs of the sensors placed on the streams of that cutset).
5. Start with the root node with no cutsets being added i.e. $\boldsymbol{t}$ = {0, 0, 0…), trivially infeasible.
6. Use branching criterion to develop branches of the tree (add cutsets to vector $\boldsymbol{t}$).
7. While performing the branching criteria, if any set of streams has already been evaluated in previous nodes, that node is not continued. This occurs frequently because one set of measurements can be a result of the union of different sets of cutsets.
8. Continue adding cutsets until the stopping criterion is met. In such case, the algorithm backs up two levels and develops the next branch.

Branching criterion

Cutset is added in the direction of minimum cost, that is, the newly added cutset is chosen such that the cost obtained by its union with the existing active cutsets is minimum.

An alternative branching has also been investigated, which is choosing cutsets in the direction of minimum objective function. It is found that this branching criterion requires much longer computational time than the other (direction of minimum sensors cost). In fact, for the small scale example given above (figure 4.6), this branching criterion requires roughly 10 times more computational time than the other criterion. For

medium or large scale problems, the difference is much larger. This is because the calculation of financial loss is an intensive computation duty, especially for middle or large scale problems.

The task remaining is to find a proper stopping criterion

Stopping criteria

In the branch-and-bound method, in each node of the search tree, it is necessary to find the lower bound for the best solution obtainable if continuing exploring down the branch of the tree. If that bound is not better than the current best solution (the incumbent) obtained so far, stop exploring down the branch. Unless the bound is obvious, it is found by solving relaxation sub-problems (e.g. LP-relaxation, Lagrangean relaxation) in the subspace of variables. Unfortunately, none of the established techniques to find the bound is applicable to our problem, the main reason is that there is no explicit expression for the objective function.

The proposed stopping criterion is as follows:

In each node, calculating ΔD and ΔC as:

ΔD = DEFL(current node) – DEFL(sensor network with maximum number of sensors)

ΔC = Cost(sensor network with maximum number of sensors) - Cost (current node)

ΔD indicates the maximum gain in financial loss and ΔC indicates the maximum cost incurred if one continues exploring down the tree from the current node. It can also be shown that if $\{\Delta C - \Delta D\}$ of current node > $\{\Delta C - \Delta D\}$ of previous node then the objective value of current node < the objective value of the previous node (see appendix

A1). The reason why ΔD & ΔC are used is illustrated in figures 4.7 and 4.8, where "MNS" is used to denote the network with maximum number of sensors.

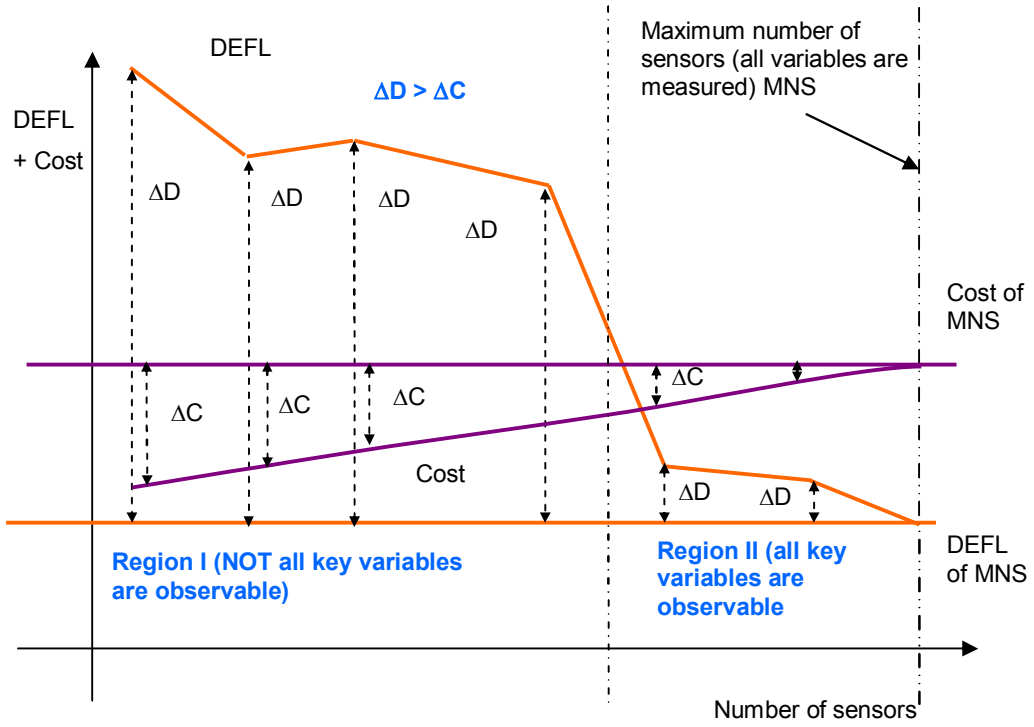


Figure 4.7 – Differentiation of regions using ΔD & ΔC

We always start exploring the branch with nodes that have ΔD > ΔC or {ΔC- ΔD} < 0 (in region I); the relationship ΔD > ΔC implies that one can reduce the objective function if continuing exploring down the tree.

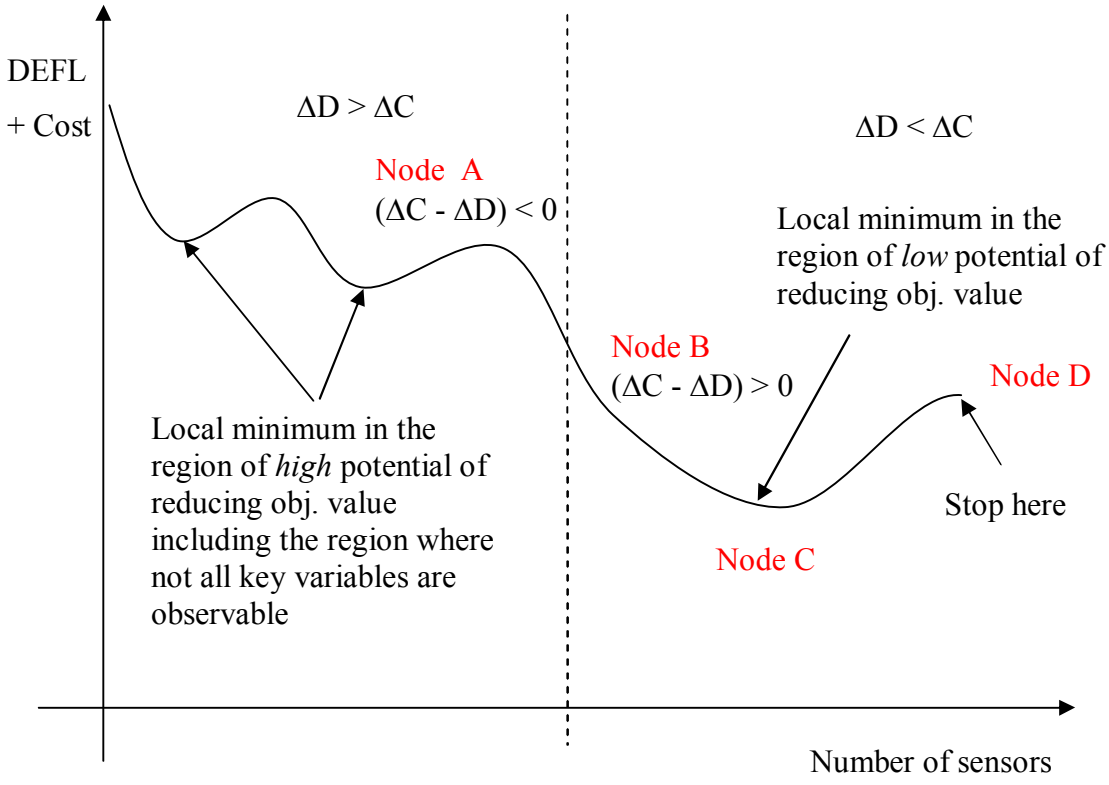


Figure 4.8 – Use of ΔD & ΔC in stopping criterion

Thus, ΔD & ΔC are used because:

i. Optimal solution can NOT be in the region where NOT all key variables are observable (region I, figure 4.7), which *always* has ΔD > ΔC

ii. The relationship ΔD > ΔC implies that there is high potential of reducing the objective function when exploring down the tree; if ΔD < ΔC: less potential

iii. If (ΔC - ΔD) of node 1 > (ΔC - ΔD) of node 2 then objective value of node 1 < objective value of node 2. Using this relationship, with reference to figure 4.8, we would have *objective value of node B < objective values of all the nodes that have* (ΔC - ΔD) < 0 (the region on left hand side).

The proposed *stopping criterion* is:

- Exploring down the branch until $\Delta D < \Delta C$
- When $\Delta D < \Delta C$, explore further down the branch until objective value of current node > objective value of previous node.

The essence of this proposed stopping criterion is, in a branch of the tree, locating a local minimum in the region of less potential of reducing objective function.

We now investigate the possibility that the global optimal solution is missed because the proposed stopping criterion stops the tree search before it reaches global optimal. This is illustrated in figure 4.9

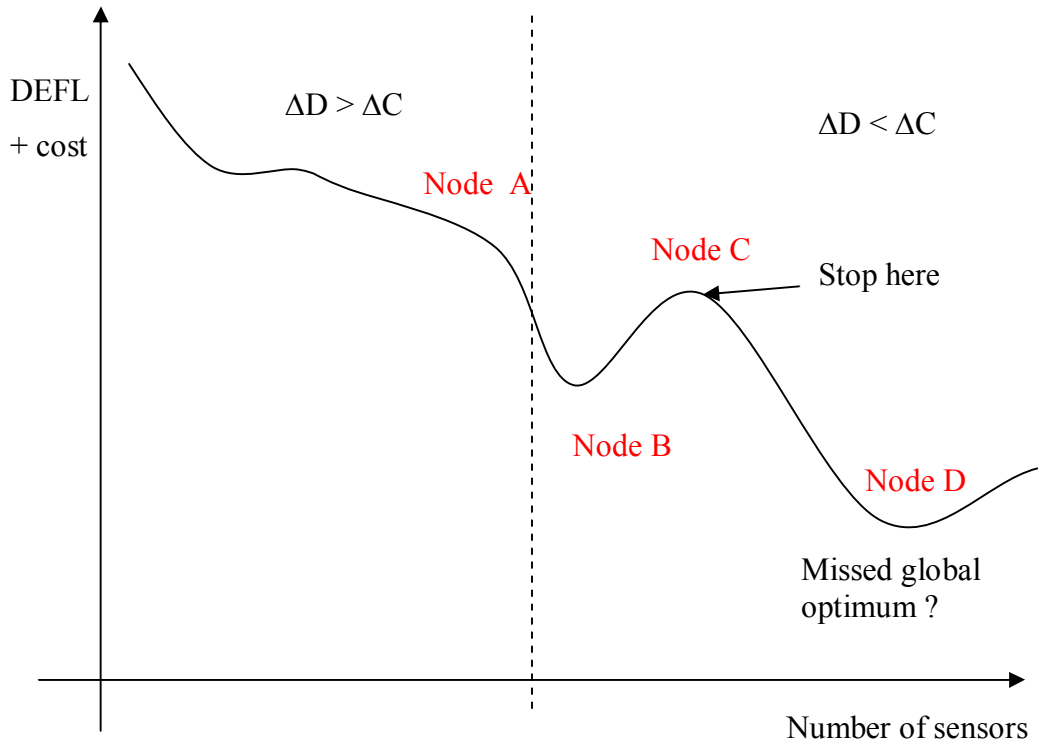


Figure 4.9 – Illustration of missing optimal solution because of stopping criterion

It is found from testing results that there always exists a monotonic pathway to reach global optimum. The reasons are:

- A union of variables (streams) is a result of *many* combinations (unions) of cutsets. This fact implies that, when cutsets are used in the tree search procedure, a specific set of active streams (measurements) can be reached by following many pathways (branches) in the tree. Table 4.4 shows an estimate of how many pathways (using cutsets) to reach a specific set of active streams (i.e. measurements location) for the Madron process problem (shown in next section)
- In most of the cases one can find a pathway in which the objective function is a monotonic decreasing function until it reaches the optimal solution (or at least objective function is monotonic decreasing in the region $\Delta D < \Delta C$ where the stopping criterion is considered). Note that changing the pathway is actually following another branch in the tree. We do not claim that one can always find such a pathway because there is no mathematical proof for this, but we have not found a counter example in which the global solution can NOT be reached by following any branch or pathway using the stated stopping criteria.

The third row of table 4.4 shows the number (N1) of possible combinations (sets) of cutsets from a given number of cutsets while the fourth row shows the number (N2) of candidate solutions (i.e. measurements locations) resulting from the same given list of cutsets. The ratio N1/N2 is an indicator of how frequently the situation that two sets of cutsets result in the same measurements location (by union operation) can occur. For example, if the ratio is 100, then among 100 possible combinations of cutsets, only one

combination leads to a candidate solution, the remaining 99 combinations are disregarded because they result in the same measurements location. This also means that expectedly there are 100 pathways to reach a specific set of active measurements. The information shown in table 4.4 is obtained from the Madron example (containing 24 streams, shown in next section)

Table 4.4- Estimate of pathways (built on cutsets) to reach a specific set of measurements

Case	1	2	3	4
Key variables	$\{S_1, S_9, S_{14}\}$	$\{S_1, S_5, S_{22}\}$	$\{S_1, S_5, S_{24}\}$	$\{S_1, S_7, S_{24}\}$
Number of key variables	3	3	3	3
Number of cutsets containing at least one key variable	99	97	102	108
Number of possible combinations of cutsets (N1)	2^{99}-1	2^{97}-1	2^{102}-1	2^{108}-1
Number of candidate solutions (N2)	46,042	64,781	39,552	38,365
N1/N2	$1.38*10^{25}$	$2.45*10^{24}$	$1.28*10^{26}$	$8.46*10^{27}$

Table 4.4 shows that only roughly 50,000 candidate solutions (each solution is a specific set of measurement locations, for comparison, the total number of such set of measurement locations is 2^{24}-1) resulted from the (2^{100} –1) possible combinations of cutsets. This result reveals that the number of pathways (built on cutsets) to reach a specific set of measurements is very large.

Figure 4.10 illustrated the two different pathways to reach optimal solution, one of them is a monotonic pathway (using actual data from one of the testing problem, example 4.1). If in pathway 1, the calculation procedure stops at the third node, then the global optimal solution (the fourth node) can still be reached by the second pathway where a different set of active cutsets is used. Note that because the calculation procedure

considers all the possible combination of cutsets from the cutsets list, all the pathways that can reach global optimal solution will be automatically considered by the tree search procedure as long as they are different from one another.

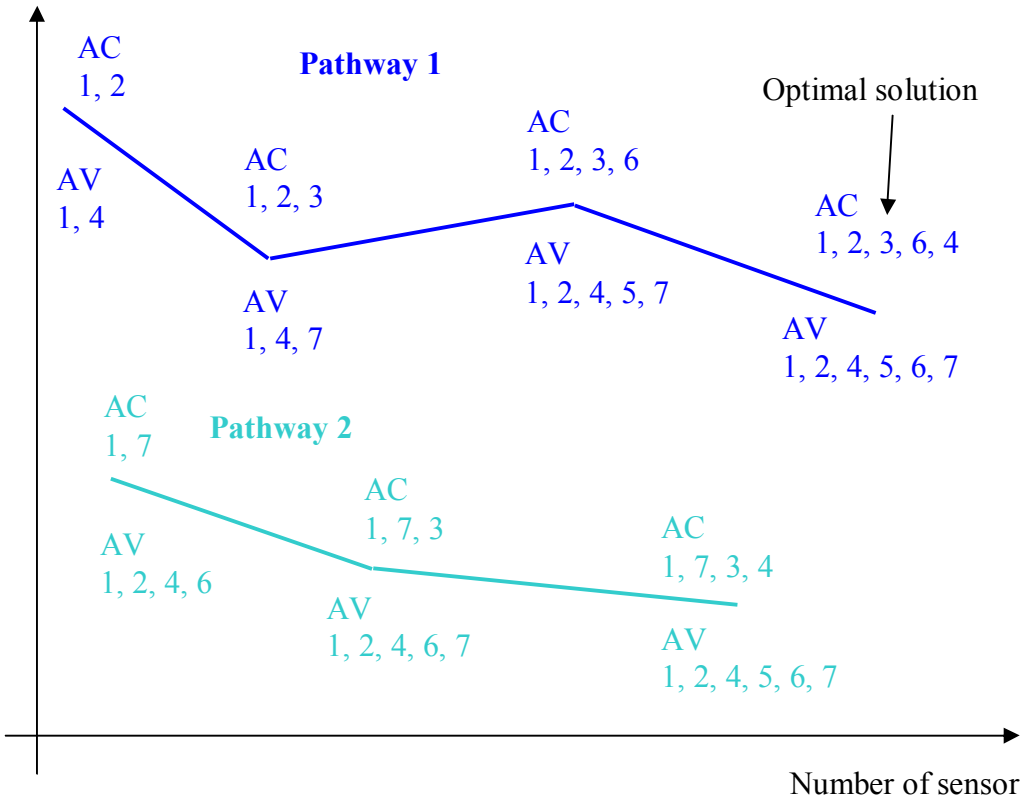


Figure 4.10 – Differeent pathways built on cutsets

In figure 4.10, AC stands for active cutsets in a node and AV stands for the corresponding active variables (stream flowrates) in a node.

The next part shows conceptually how to obtain a monotonic pathway (to reach optimal solution).

Illustration:

Consider the five cutsets C_1,C_2,C_3,C_4,C_5 shown in figure 4.11. A cutset is represented by a line, a stream (variable) is represented by a cross. There are two key variables K_1 and K_2

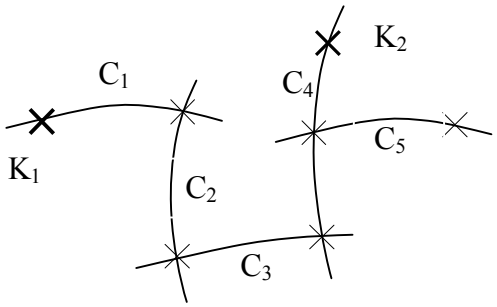


Figure 4.11 – Illustration of cutsets

There are two combinations of cutsets that give the same result, which is measuring all streams shown in figure 4.10: $C_1 \cup C_2 \cup C_4 \cup C_5$ and $C_1 \cup C_3 \cup C_4 \cup C_5$

- Cutsets C_3 and C_5 are only weakly connected to the key variables: using C_3 or C_5 alone does not make any of the key variable observable, using both C_3 and C_5 would make K_2 observable. The improvement (decrease) in financial loss when using such cutset is small
- Cutsets C_1 and C_4 are strongly connected to the key variables: using C_1 makes K_1 redundant while using C_4 makes K_2 redundant. The improvement in financial loss when using such cutset is large

Conceptually, to obtain a monotonic pathway (to reach optimal solution), cutsets can be added in the following order (suppose that there are more one key variable, which is usually the case):

- Put first cutsets that are weakly connected to the key variables like cutsets C_3 and C_5 then put cutsets strongly connected to the key variables like C_1 and C_4. The reason is that if cutsets are added in such order, the improvement (decrease) in financial loss progressively increases with the number of active sensors (active

cutsets), the likely result is that objective function progressively decreases with the number of active sensors (i.e. monotonic pathway)

- Put first expensive cutsets then put cutsets that are less costly: if cutsets are added in such order, the increase in cost becomes progressively smaller with number of active sensors. However, this sequence of adding cutsets is not favored by the branching criterion, which requests using first the cheapest cutsets.

All these discussions point out that:

- Because there are so many pathways to reach a candidate solution, if the global optimal solution is not reachable in a pathway (because that pathway is not monotonic), it would be reachable in another pathway. Thus, the chance of finding global optimal solution is very high.
- The bad side of this fact is that the stopping criterion may not have any effect at all, that is, one candidate solution if not reachable in a pathway can still be reachable in another pathway. The result is that the number of candidate solutions explored is equal for both cases: with stopping criterion and without stopping criterion. The obtained results from the Madron example confirm this speculation.

4.10. Parallelized cutset-based tree search method

4.10.1. Overview of parallel computing

Recently, scientific computing has gradually shifted from serial paradigm to parallel paradigm, especially for large scale problems. Characteristic and benefits of parallel computing (as compared against serial computing) are shown in table 4.5

Table 4.5- Parallel computing vs. Serial computing

Serial computing	Parallel computing
- Run on a single computer having a single CPU - Instructions are executed one after another - Only one instruction may execute at any moment in time	- Run on multiple CPUs - Problem is broken into many parts - A part (a subset of data and / or a part of program instructions) is executed concurrently (on multiple CPUs) together with other parts
	Benefits: - Reduce computational time => solve problem *faster* - Can solve problems with large data set => solve *bigger* problems

The serial computing is illustrated in figure 4.12. As seen in this figure, the four computation tasks (task 1 to 4) are executed sequentially using the whole problem data

(data set 1 to 4). Example of computation task is any kind of arithmetic calculation; example of problem data is space (domain) of variables in optimization / modeling / simulation problems or input data in data mining or data visualization problems

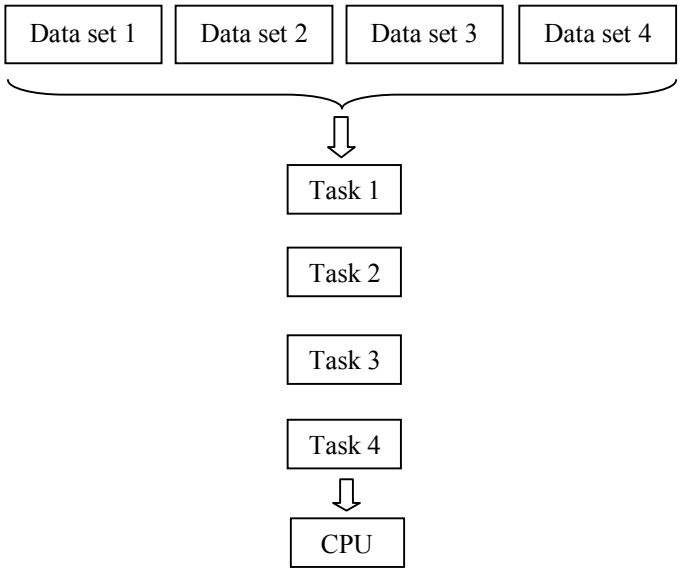


Figure 4.12 – Illustration of serial computing

The most common way to do parallel computing is to process different parts of problem data in different computer nodes (CPUs). This approach is called single instruction multiple data (SIMD) and is illustrated in figure 4.13. It is appropriate to use this approach when problem data can be divided into different parts, each part can be processed independently. This is indeed the case in data mining / visualization problems and optimization / modeling problems, etc.

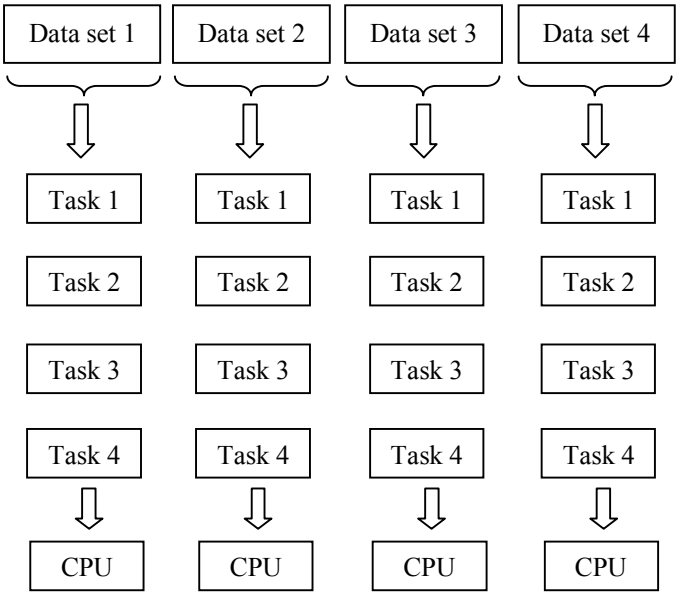


Figure 4.13 – Illustration of single instruction multiple data (SIMD) palallel program

In case the computational tasks can be executed independently (execution of a task does not depend on output from another task), then the program can be parallelized by executing the tasks concurrently as shown in figure 4.14

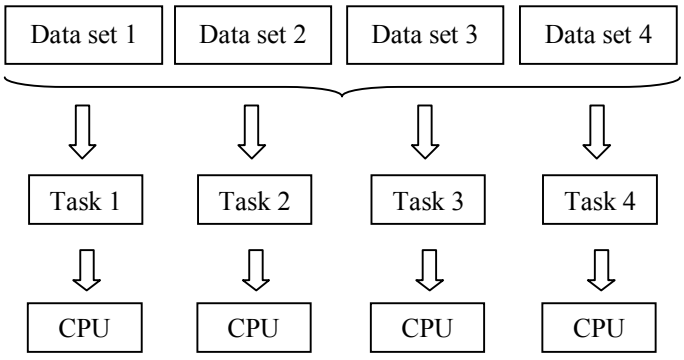


Figure 4.14 – Illustration of multiple instruction single data (MISD) palallel program

If both program instructions and program data can be divided, the parallelization approach is called multiple instruction multiple data (MIMD)

The tree search method for solving SNDP is leaned to SIMD approach because the space of variables can be partitioned into multiple sub-spaces, which are then explored concurrently as shown in figure 4.15

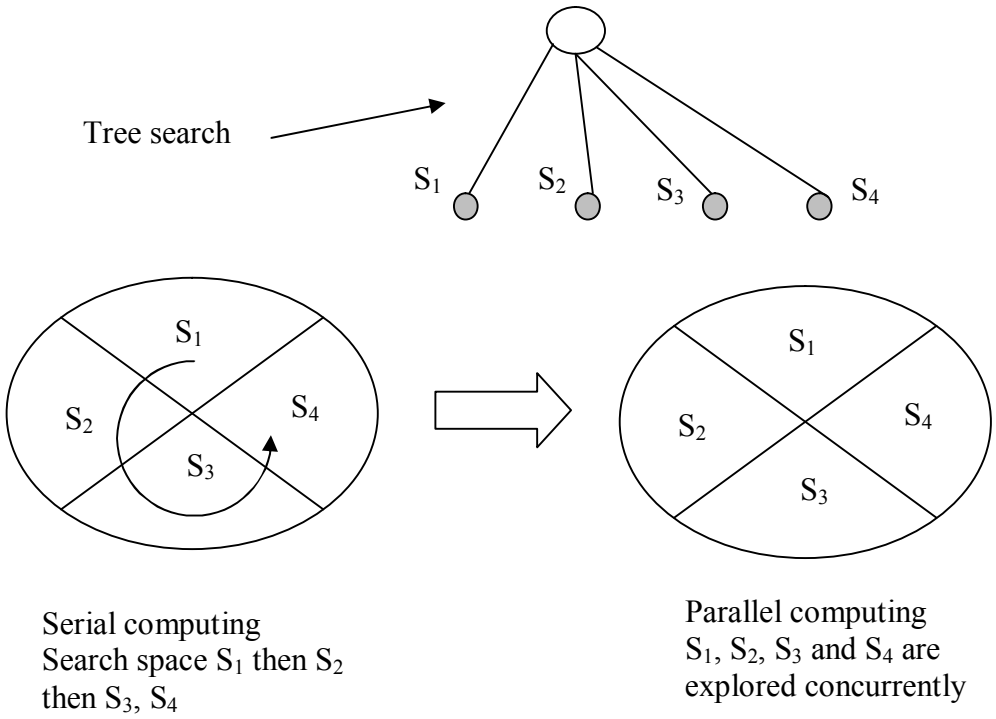


Figure 4.15 – Parallel tree search method

There are many so-called library routines / interface specifications that make it easier for programmer to transfer from serial program to parallel program; the most well-known are openMP (a library of compiler directives and subroutines) and Message Passing Interface (MPI). From a programmer perspective, openMP is very easy to use because of its simplicity; however there is one down side of this advantage: it is difficult to obtain an optimized performance, especially for a big program. The MPI requires

significant effort in programming but it is relatively easy to obtain a satisfactorily good performance, the MPI is very suitable for big programs like the value-optimal SNDP under investigation in this work.

In this work we use all approaches: SIMD (single instruction multiple data) and MISD (multiple instruction single data) and MIMD. The parallel computing is done using an implementation of Message Passing Interface (MPI) called "openMPI". More details on parallel computing and MPI can be found in Pacheco (1997)

4.10.2. Automatic parallelization of loops

The simplest way to do parallel computing is to parallelize the loops (do, for loops). This is illustrated in figure 4.16 where a loop is used to do computation on an array containing 80 elements. The data (80 elements of the array) can be divided into four sub-sets, which are then processed concurrently in four computer nodes as shown in figure 4.16

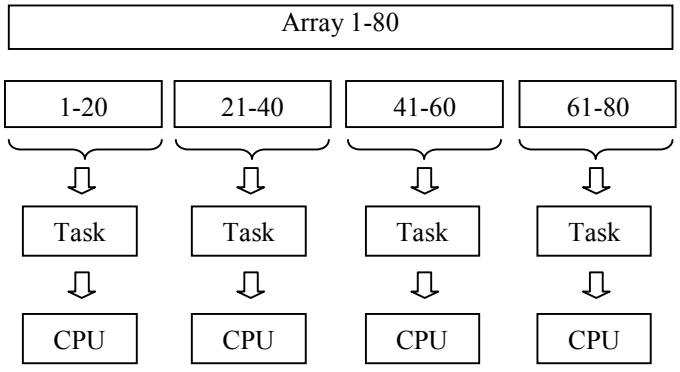


Figure 4.16 – Parallelization of loop

This parallelization of loops can be done easily using openMP. Fortunately, recent Fortran compilers that support parallel computing can accomplish this task automatically without any manual direction from the programmer. The Intel Fortran compiler used in this research work has such kind of feature and as shown in the Madron illustrated example, it greatly reduces computational time

4.10.3. Message Passing Interface (MPI)

The principle of parallel computing is to execute different parts of program on different computer nodes (CPUs). However, it is usually the case that computation in one node still needs to know certain kind of information from other nodes. For example, with reference to figure 4.15, the tree search on sub-space S_1 in one node may need to know the current best solutions obtained in other nodes (where the searched spaces are S_2, S_3, S_4) because a tighter bound in branch and bound (tree search) method would result in improved performance. This can be done by assigning a node (denoted as master node, node 0) that receives and updates the current best solutions obtained in all other nodes on which the tree search procedure is run (denoted as worker nodes, nodes 1 to 4). The most updated current best solution found in any worker node at any moment in time is "communicated" to all other worker nodes in the manner illustrated in figure 4.17

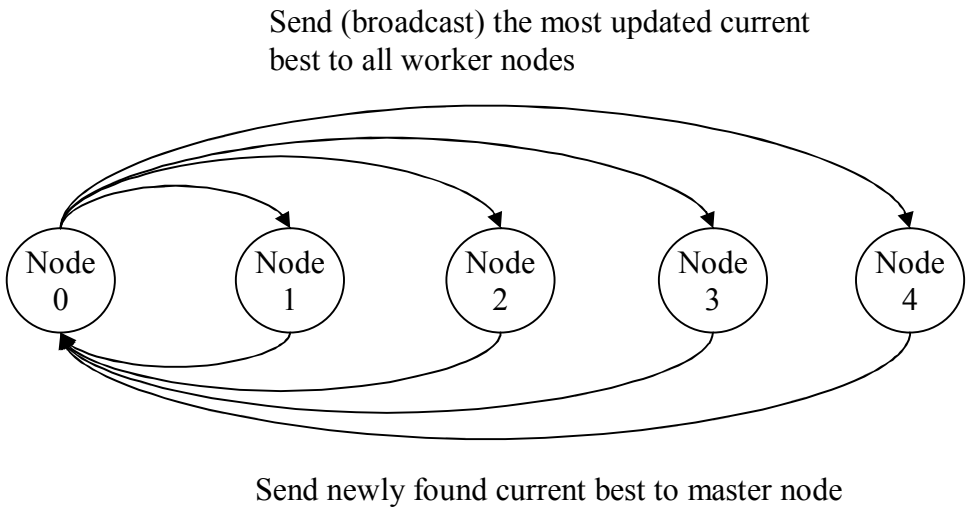


Figure 4.17 – Communication between nodes in parallel tree search method

Thus, there is usually a necessity to communicate between computer nodes. The MPI is developed to provide communication channels between computer nodes (as implied by the name Message Passing Interface). The MPI is a specification / standard for passing message between computer nodes (the most current standard is MPI version 2.2). Openmpi is one of the most popular implementation of MPI; it is a library of message passing subroutines (as well as other supporting subroutines for file handling, debugging, etc…). It is a tool provided for the programmer to do parallel computing; the programmer is responsible for determining all parallelism. More details on MPI can be found in Pacheco (1997) and various documents maintained at (http://www.mpi-forum.org/docs/). Note that, in MPI terminology, there is usually a computer node called master node (or server node), the rest are called worker nodes

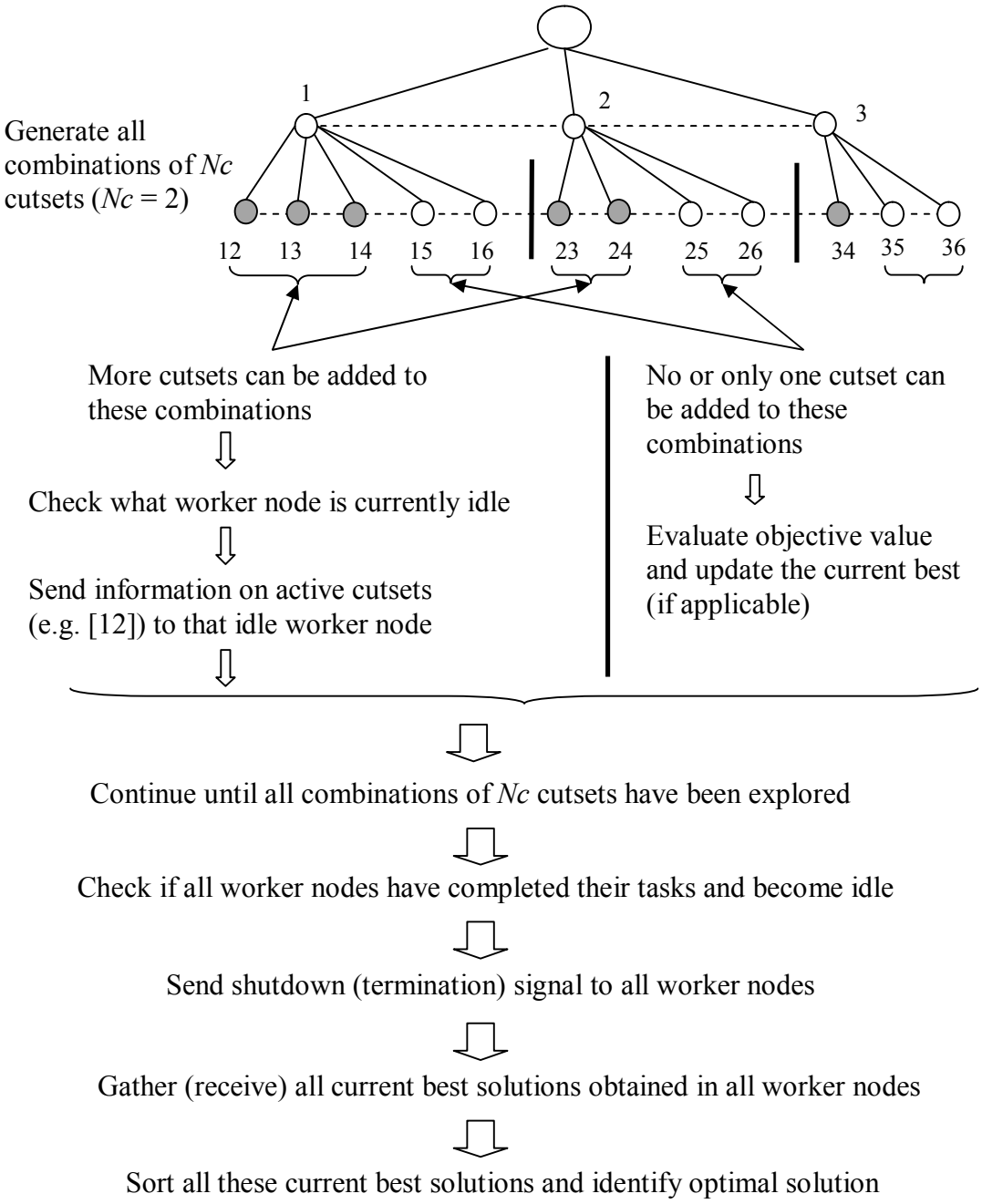
The next section describes three parallelized versions of cutset-based method for solving value-optimal SNDP. The first one follows the SIMD approach, the second one

follows the MISD approach and the third one follows MIMD . The focus is on the second one and third one because their performance is much better than the first one (in this specific problem).

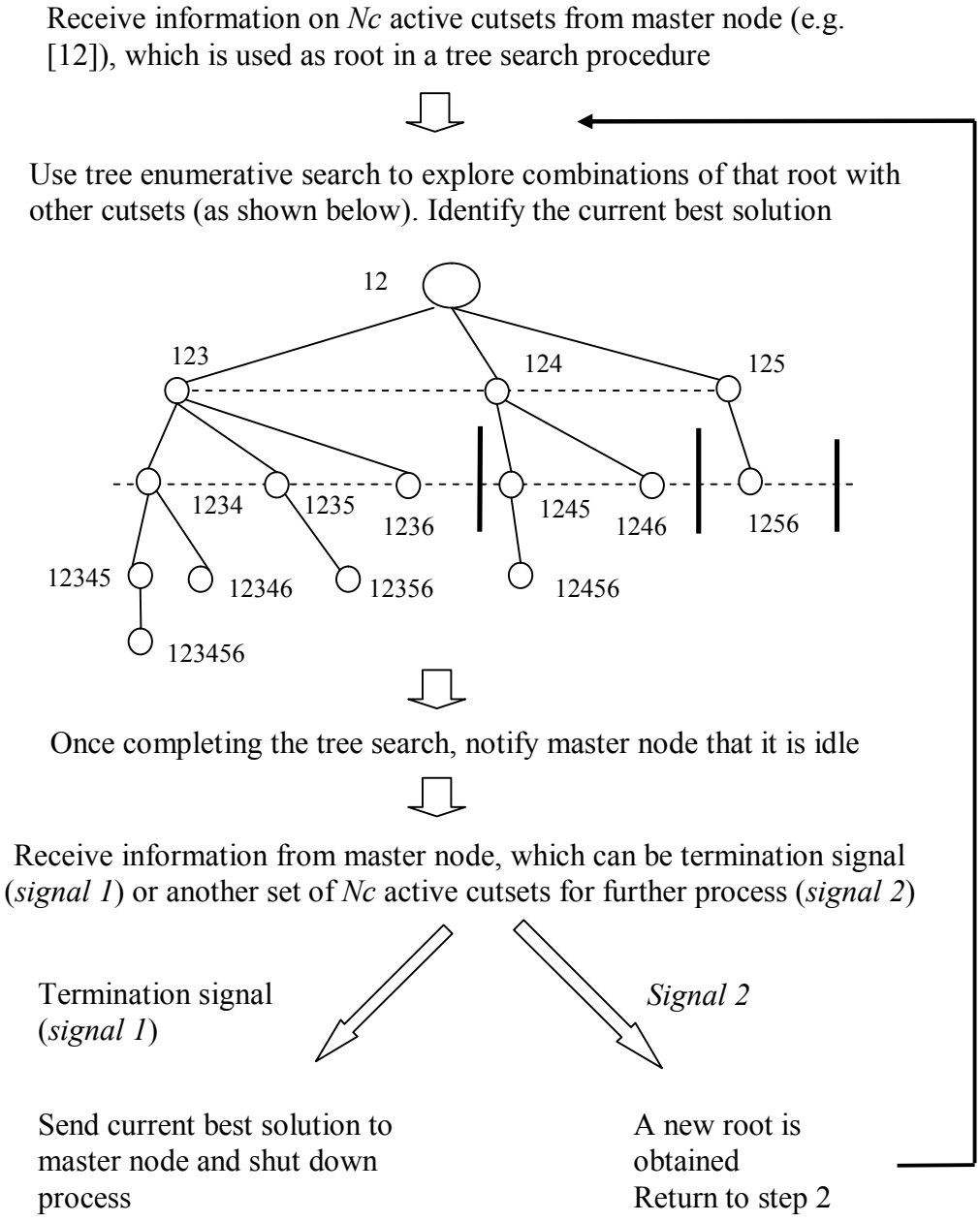
4.10.4. Parallelized cutset-based method – SIMD approach

This parallel program follows the principle illustrated in figure 4.15, which is to partition space of variables into several sub-spaces. The calculation procedure is depicted in figure 4.18 for a system containing 6 cutsets.

- In this parallel program, no branching criterion is used. More specifically, cutsets are added in numbered order, for example, if the current set of active cutsets is [124] then the next cutset to be added is 5, if the new set [1245] is already evaluated (because the measurement locations resulted from union of cutsets [1245] is already evaluated) then consider new set [1246] and so on.
- As discussed in previous section (and proven in the Madron illustrated example), the proposed stopping criterion has little effect in eliminating non-optimal solutions. So it is not used in this parallelized version of cutset-based method. Thus, the best solution obtained by this method is guaranteed to be optimal solution.
- As illustrated in figure 4.18, all combinations of cutsets containing the root [1 2] are evaluated in one worker node (as shown in 4.18b, these combinations are [123], [124], [1234] etc). At the same time all combinations of active cutsets containing another root (e.g. [1 3]) are evaluated in another worker node and so on. This is how the principle of dividing problem data works.



4.18a Calculation procedure in master node



4.18b Calculation procedure in worker nodes

Figure 4.18 – Calculation procedure for parallel computing – SIMD approach

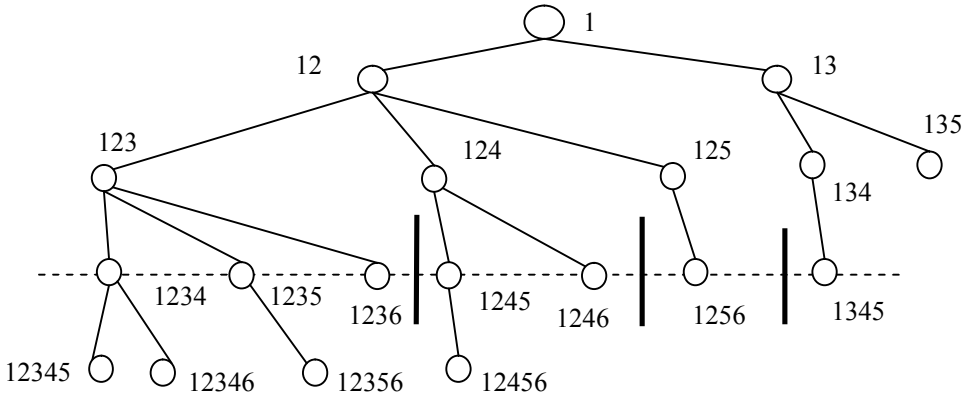
Notes:

- If number of cutsets is m and rank of the last active cutset is n (for example, the current set of active cutsets is [12…n]) then the total number of possible new sets of active cutsets starting from the current set of active cutsets [12…n] is $2^{(m-n)} - 1$ (for example, the new sets are [12…n,$n+1$], [12…n,$n+1$,$n+2$], etc). If $(m-n) \leq 3$ (i.e. the number of possible new sets $\leq 2^3 - 1$), the current set of active cutsets [12…n] is to be processed in master node, otherwise ($(m-n) > 3$), it is processed in worker nodes. The reason is that communication between computer nodes costs time, so it is better to process a root (set of active cutsets) with $(m-n) \leq 3$ directly in master node rather than sending it to worker nodes (in figure 4.18, the condition $(m-n) \leq 1$ is used)
- Initially, all worker nodes are idle (there is no running task at that time) so master node automatically sends roots (combinations of *Nc* active cutsets) to worker nodes. Only in later stage that the master node needs to check if a worker node is idle or not in order to assign new job for that worker node
- The master node acts as a "manger": it assigns jobs and monitors job completion for worker nodes; if master node finds that a worker node is idle (because its job was completed), master node assigns new job for that worker node

4.10.5. Parallelized cutset-based method – MISD approach

This parallel program follows the MISD (multiple instruction single data) approach illustrated in figure 4.15, which is to execute multiple tasks at the same time. The calculation procedure is depicted in figure 4.19

Run tree search procedure using cutsets



Perform union operation of active cutsets to obtain candidate solutions (sets of measurement locations)

Store candidate solutions in a list until 100 candidate solutions have been stored. This list is called "*list100*"

Check what worker node is idle. Send *list100* to that idle worker node. Reset the *list100*

Continue until all candidate solutions have been generated and passed on to worker nodes (tree search procedure was completed)

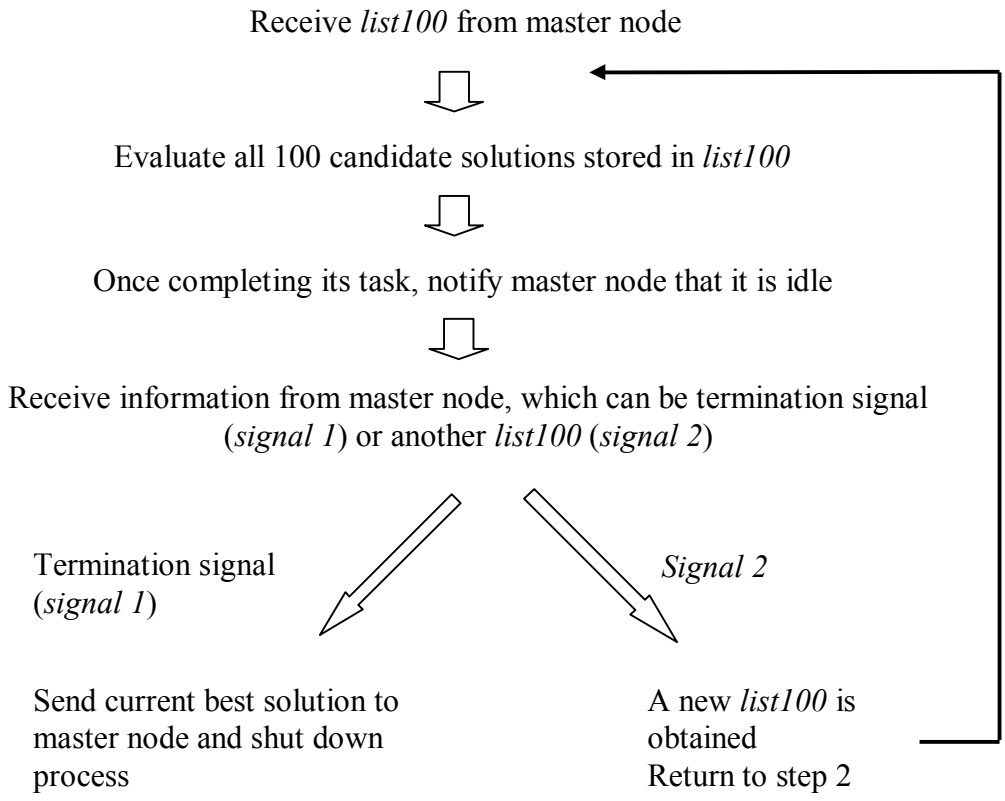
Check if all worker nodes have completed their tasks and become idle

Send shutdown (termination) signal to all worker nodes

Gather (receive) all current best solutions obtained in all worker nodes

Sort all these current best solutions and identify optimal solution

4.19a Calculation procedure in master node



4.19b Calculation procedure in worker node

Figure 4.19 – Calculation procedure for parallel computing – MISD approach

- Similar to the SIMD approach, there is no stopping criterion in this MISD parallel program so the best solution obtained by this method is guaranteed to be optimal solution
- Regarding branching criterion, there are two alternatives: i) branching criterion is used, ii) branching criterion is not used (the tree search illustrated in figure 4.18b and 4.19a does not have a branching criterion). Advantage of the former is that optimal solution is usually identified early (which is very beneficial if

computation process has to be terminated halfway because computational time exceeds limit). Advantage of the latter is that computational time is much shorter than the former because there is no need to determine which cutset (to be added to current active cutsets) results in a minimum cost among all candidate cutsets.

- It can be seen that in this approach there are two computation tasks that are executed simultaneously: generating all candidate solutions (by using tree search procedure with cutsets) in master node and evaluating (i.e. calculating objective value, which is financial loss plus cost) all generated candidate solutions in worker nodes. Although these two tasks are not completely decoupled (candidate solutions need to be generated first before they can be evaluated), the two tasks can still be executed concurrently: the fast job (generating candidate solutions) is done in *one* master node while the slow job (evaluating candidate solutions) is divided across *many* worker nodes. Relative computational times of these two steps are shown in figure 4.20

In figure 4.20, the data is taken from the case study number 2 in the Madron illustrated example shown below (using a 2.8 GHz Pentium CPU, 1028 RAM PC). The straight line "Evaluating 100 solutions" shows the *average* computational time to evaluate 100 candidate solutions (this time ranges from 49 sec to 294 sec). The curve "with branching criterion" shows computational time to generate 6400 candidate solutions when branching criterion is used, the curve "without branching criterion" shows the same computational time but no branching criterion is used. The value corresponding to point *n* in *x*-axis is the elapse time when the number of generated solutions increases from

6400*(n-1) to 6400*n. Thus figure 4.20 shows that computational time to generate 6400 candidate solutions increases progressively with the number of solutions that have been generated. The reason is that a candidate solution (a set of measurement locations) needs to be verified that it is not coincident with any solution that has been evaluated so far. The time spent for this verification step increases with the number of solutions that have been generated; this fact explains the dependence of computational time on number of candidate solutions shown in figure 4.20

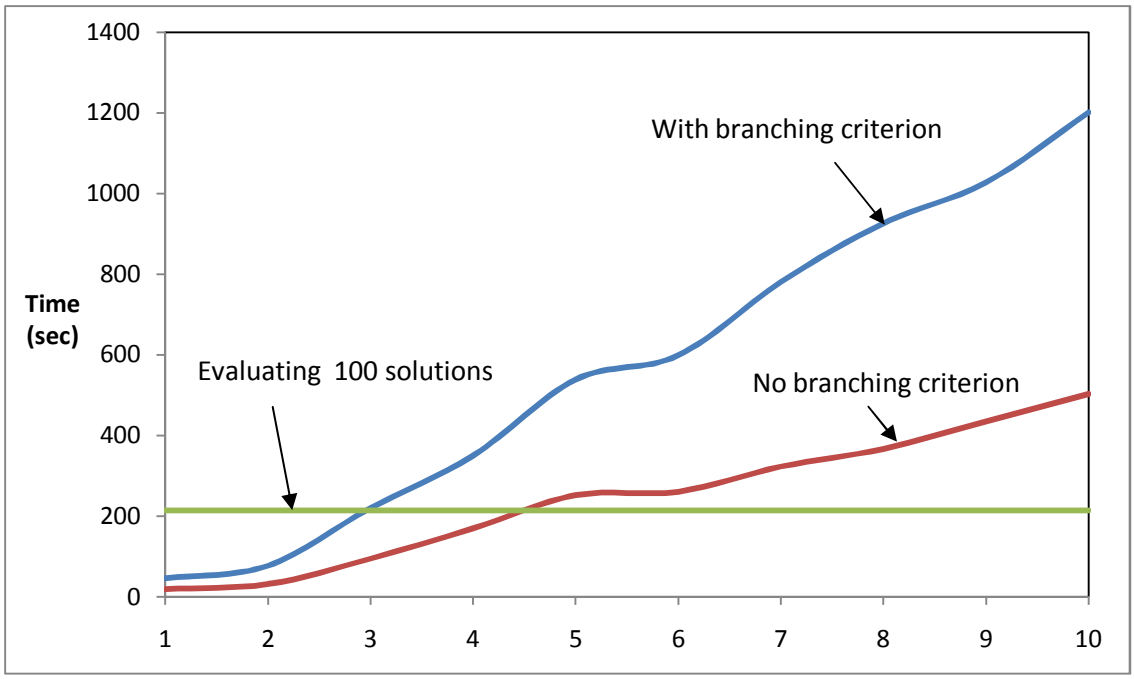


Figure 4.20 – Comparisons of computational times of two steps in the MISD

Thus, on the same basis (e.g. generating 100 solutions and evaluating these 100 solutions), the first task is much faster than the second task.

In the MISD approach, the ideal situation that results in optimum performance is shown in figure 4.21

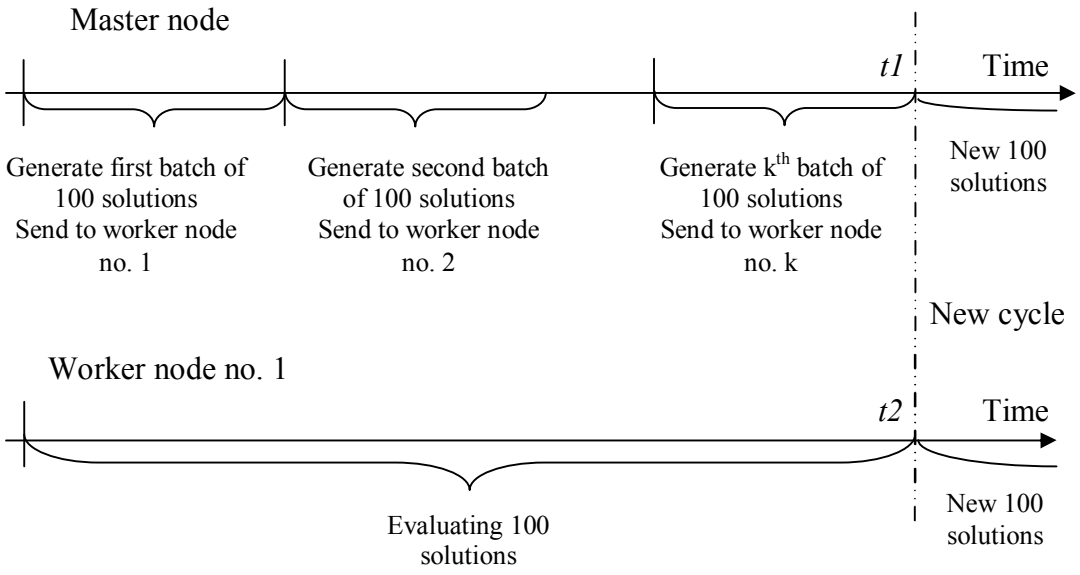


Figure 4.21 – Ideal situation for MISD parallel program

The best performance (achieving short computational time without using too much resource) is obtained when there is no or little idle (dead) time in any computer node. This situation is realized when at the time master node finishes assigning jobs for all worker nodes and starts a new cycle, the first worker node just finishes its job and is ready to take on another job as shown in figure 4.21.

Suppose that the number of worker nodes utilized in the process is k and let the time to generate (100*k) solutions be $t1$ and the average time to evaluate 100 solutions be $t2$, then the best performance is obtained when $t1 \approx t2$. However, this ideal situation will never be achievable because $t1$ increases as computation process progresses (as shown in figure 4.20). In the period where $t1<t2$, when starting a new cycle, the master node can not find any idle worker node to assign new job (so there is delay time). In the other hand, if $t1>t2$, worker nodes are idle for some time before they are assigned a new job (so there is idle time or resource is not fully utilized). The best situation one can get is

that *t1* is close to *t2* in the whole process. The case "No branching criterion" in figure 4.20 is near to this "best" situation.

As can be inferred from the calculation procedure, computational time of the overall parallel computing process can not be reduced lower than the time to generate all candidate solutions. This limit on computational time is achieved when $t2 < t1$ because this condition ($t2 < t1$) implies that the master node can always find an idle worker node to assign job (evaluating a batch of 100 solutions) whenever it needs. This usually means using more computer nodes (CPUs)

Parallelized version (MISD approach) of cutset-based tree search method with decomposition has also been developed. Decomposition technique was described in Gala and Bagajewicz (2006). This parallelized program of cutset-based method with decomposition is similar to the one without decomposition. The only differences are:

i) Differences in cutset-based tree search procedure (section 4.9):

- In step one: decompose the process graph into several sub-graphs. For example a system containing six stream [1 2 3 4 5 6] is partitioned into two sub-systems: one containing three streams [1 2 3] (sub-graph A) and the other containing [3 4 5 6] (sub-graph B). Then find all cutsets in all sub-graphs.
- In step two, consider only cutsets that contain at least one key variable and / or a connecting stream (i.e. the intersection between two adjacent sub-graphs, for example stream [3] connecting [1 2 3] and [3 4 5 6]). The reason why connecting stream is also considered is better explained through an counter

example: suppose that [1] is the key variable, sub-graph A has only one cutset: [1 2 3] (which contains both key variable [1] and connecting stream [3]), sub-graph B has two cutsets containing connecting stream [3], which are [3 4 5] and [3 4 6] (no cutset in sub-graph B contains key variable [1]). If cutsets [3 4 5] and [3 4 6] are not considered, the solutions that contain many measurements like [1 2 3 4 5] or [1 2 3 4 5 6] will never show up (these heavily measured systems are most likely to be optimal solution if the parameter *Ks* is large)

- No stopping criterion is used

ii) Differences in parallelized version (section 4.10.5):

The only difference is in step two of calculation procedure in worker node (figure 4.19a) "Evaluate all 100 candidate solutions stored in *list100*". If no decomposition is used, the task is simply to evaluate objective functions of candidate solutions. When decomposition is used, if a candidate solution (an element of a *list100*) is obtained from union operation of cutsets coming from the same sub-graph, then simply evaluating objective value of that candidate solution. Otherwise (cutsets come from different sub-graphs), suppose that the candidate solution is obtained from union of cutsets A & B (from sub-graph 1) and cutset C (from sub-graph 2) and cutset D (from sub-graph 3). Then, as illustrated in Gala and Bagajewicz (2006), all solutions (that can be resulted from these four cutsets *A, B, C, D*) are found by performing the following operations: $(A \cup B) \cup C \cup D$; $(A \cup B) \cup C \oplus D$ and $(A \cup B) \oplus C \cup D$ and $(A \cup B) \oplus C \oplus D$ (note that $\cup$ and $\oplus$ are union operation and ring sum operation, respectively). The four

solutions (sets of measurement locations) resulted from the above four operations are then evaluated

Now realizing that $(A \cup B) \cup C \oplus D$ is actually the union $(A \cup B) \cup C \cup D$ minus the connecting stream of sub-graph 2 (containing cutset C) and sub-graph 3 (containing cutset D) while $(A \cup B) \oplus C \cup D$ is the union $(A \cup B) \cup C \cup D$ minus the connecting stream of sub-graph 1 (containing cutsets A, B) and sub-graph 2 (containing cutset C), etc. Thus the above three operations $(A \cup B) \cup C \oplus D$ and $(A \cup B) \oplus C \cup D$ and $(A \cup B) \oplus C \oplus D$ are equivalent to exploring all possibilities of removing connecting streams out of the union $(A \cup B) \cup C \cup D$. This approach is used in this work: a tree enumerative procedure is used to explore all possibilities of removing connecting streams out of a candidate solution

Thus the step "Evaluate all 100 candidate solutions stored in *list100*" now comprises of two steps: i) for each candidate solution (an element in the *list100*), use a tree enumerative procedure to explore all possibilities of removing connecting streams out of that candidate solution, ii) then evaluate all the resulting candidate solutions (sets of measurement locations)

4.10.6. Parallelized cutset-based method – MIMD approach

In this MIMD (multiple instructions multiple data) approach, both program data and programs instructions are divided. More specifically, this approach combines both the technique of partitioning space of variables (SIMD approach shown in section 4.10.4) and the technique of dividing and concurrently executing computation tasks (MISD

approach shown in section 4.10.5). The calculation procedure is essentially the same as that of the MISD approach (figure 4.19) except that:

- The task of generating all candidate solutions (task one) is now divided and shared by several computer nodes (call group 1 of computer nodes) instead of only one computer node (the master node) in the MISD approach.
- The group of computer nodes that is responsible for task two, which is evaluating all the generated candidate solutions (called group 2), receive the job assignment (the *list100*) from a computer node in group 1.
- Because task one is fast job while task two is slow job, group two of computer nodes (responsible for task two) is bigger (containing more computer nodes) than group one.
- Group two is divided further into sub-groups. The number of these sub-groups is equal to the number of computer nodes in group one. Each computer node in group one "manages" one sub-group (belonging to group two) as illustrated in figure 4.22.
- The function (duty) and relationship between a computer node in group one (denoted as "managing node") and a sub-group that it manages is similar to the function and relationship between master node and worker nodes in MISD approach (figure 4.19); that is, the managing node generates *list100* and sends it to worker nodes under control of this managing node (e.g. in figure 4.22. CPU1 controls sub-group 1, etc.)
- The function and relationship between master node and computer nodes in group one (the managing nodes) is similar to the function and relationship between master node and worker nodes in SIMD approach (figure 4.18): master node generates a root (a combination of *Nc* active cutsets like [1 2], [1 3] etc) and send it to managing nodes

- The communication and assigning jobs between master node and computer nodes in group one and group two are illustrated in figure 4.22: group one comprises of four "managing" nodes (CPU1 to CPU4), group two comprises of four sub-groups.
- The number of "managing" computer nodes in group one is an important parameter because it strongly affects performance of the method. As shown in illustrated example, usually the best performance is achieved at small number of "managing" computer nodes.
- No branching criterion and no stopping criterion is used

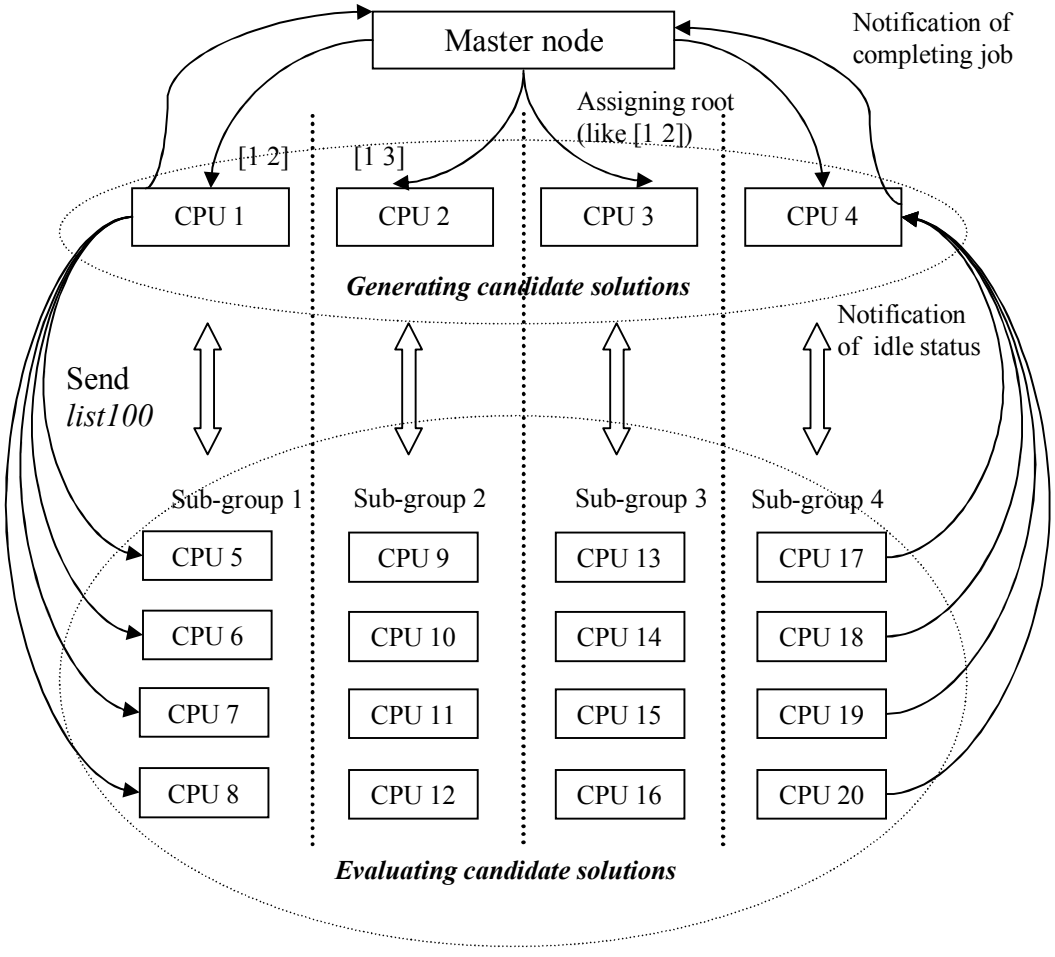


Figure 4.22 – Illustration of MIMD parallelization

4.11. Example 4.2 – Madron problem

All of the proposed methods were implemented in Fortran. The exhaustive tree search, the GA method and the serial version of cutset-based method were run on a 2.8 GHz Intel Pentium CPU, 1028 MB RAM PC. The parallelized programs were run on computer network ("super computer") of OU (University of Oklahoma) Supercomputing Center for Education and Research (abbreviated name is OSCER). The OSCER super computer uses Intel Xeon CPU (speed ranges from 2.0 to 2.4 GHz) and 8,768 GB RAM. More detail on configuration of OSCER super computer can be found in the website www.oscer.ou.edu

Flowsheet of the example is given in Figure 4.23, which was introduced by Madron and Veverka (1992). Madron and Veverka (1992) did not report flow rates, so the flowrate values shown in Table 4.6 were taken from Bagajewicz (1997). The precision and cost of sensors are also given in Table 4.6

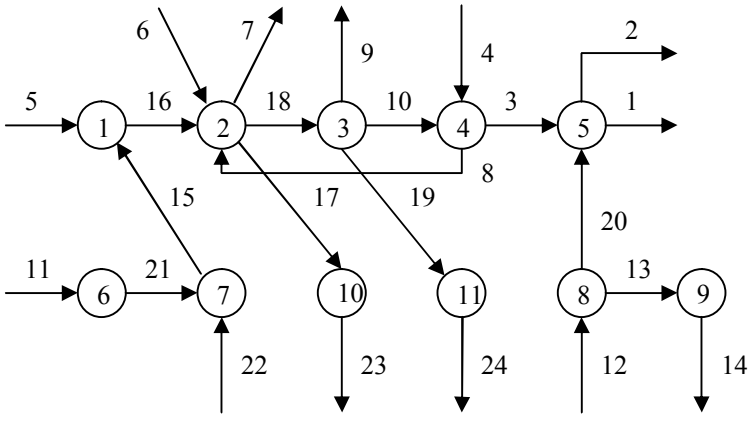


Figure 4.23 – Flowsheet of Madron problem

Table 4.6- Data for Madron problem

Stream	Flow	Sensor cost	Sensor Precision (%)	Stream	Flow	Sensor Cost	Sensor Precision (%)
1	140	19	2.5	13	10	12	2.5
2	20	17	2.5	14	10	12	2.5
3	130	13	2.5	15	90	17	2.5
4	40	12	2.5	16	100	19	2.5
5	10	25	2.5	17	5	17	2.5
6	45	10	2.5	18	135	18	2.5
7	15	7	2.5	19	45	17	2.5
8	10	6	2.5	20	30	15	2.5
9	10	5	2.5	21	80	15	2.5
10	100	13	2.5	22	10	13	2.5
11	80	17	2.5	23	5	13	2.5
12	40	13	2.5	24	45	13	2.5

Information used in the calculation of financial loss is as follows:

- Probability of sensors = 0.1 (for all sensors)
- Biases (in failed sensors) are assumed to follow normal distribution with zero means and standard deviations = 4.0 (for all sensors)
- Windows time of analysis T = 30 days
- The K_s values (cost of product or cost of inventory) vary with design case studies, which are shown in table 4.7

The value-optimal SNDP (equation 4.10) is being studied and performance of the proposed cutset-based methods for solving value-optimal SNDP is tested. Ten design case studies together with the optimal solutions obtained by using cutset-based methods

are shown in table 4.7. The objective is to minimize financial loss plus cost. The last four columns of table 4.7 show details of the obtained optimal solutions, which are the number of sensors, the measurements location, the cost of sensors and the financial loss respectively.

Table 4.7- Results for Madron problem

Case study	Key variables	K_s value	Number of sensors	Measured variables	Sensors cost	Financial loss
4.2.1	1, 9, 14	$Ks_1 = 25$ $Ks_9 = 20$ $Ks_{14} = 20$	11	1, 2, 3, 4, 8, 9, 10, 12, 13, 14, 20	137	415.1
4.2.2	1, 5, 22	$Ks_1 = 25$ $Ks_5 = 20$ $Ks_{22} = 20$	11	1, 2, 3, 4, 5, 8, 10, 12, 13, 20, 22	158	471.4
4.2.3	2, 6, 24	$Ks_2 = 25$ $Ks_6 = 20$ $Ks_{24} = 20$	4	2, 6, 19, 24	57	400.1
4.2.4	4, 9, 23	$Ks_4 = 25$ $Ks_9 = 20$ $Ks_{23} = 20$	4	4, 9, 17, 23	47	283
4.2.5	4, 5, 24	$Ks_4 = 25$ $Ks_5 = 25$ $Ks_{24} = 45$	4	4, 5, 19, 24	67	527.1
4.2.6	1, 5, 24	$Ks_1 = 25$ $Ks_5 = 20$ $Ks_{24} = 20$	12	1, 2, 3, 4, 5, 8, 10, 12, 14, 19, 20, 24	175	498.7
4.2.7	1, 5, 24	$Ks_1 = 45$ $Ks_5 = 36$ $Ks_{24} = 45$	15	1, 2, 3, 4, 5, 8, 9, 10, 12, 13, 14, 18, 19, 20, 24	210	891.2
4.2.8	1, 7, 24	$Ks_1 = 25$ $Ks_7 = 20$ $Ks_{24} = 25$	12	1, 2, 3, 4, 7, 8, 10, 12, 13, 19, 20, 24	157	538.8
4.2.9	1, 7, 24	$Ks_1 = 45$ $Ks_7 = 40$ $Ks_{24} = 45$	19	1, 2, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 20, 23, 24	251	859.3
4.2.10	1, 7, 24	$Ks_1 = 80$ $Ks_7 = 70$ $Ks_{24} = 80$	22	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24	302	1471.8

A few observations can be withdrawn from the above results:

- The locations of key variables can greatly affect the financial loss and the obtained optimal network as evidenced in design cases 4.2.1 to 4.2.6: all of these six design cases have three key variables with similar *Ks* values (only locations of key variables are different) but the number of sensors in optimal network can change significantly.
- As *Ks* values increase, the financial loss term dominates the cost term and optimal network would contain more sensors to reduce financial loss as evidenced in design cases 4.2.6 and 4.2.7 (same key variables, different *Ks* values) and design cases 4.2.8 to 4.2.10
- There is a very high chance that all key variables appear in the optimal solutions: this is the case in all ten design case studies under consideration

The next section shows performance of the proposed methods to solve value-optimal SNDP.

4.11.1. Exhaustive tree search using individual measurements

The simplest method to solve the value-optimal SNDP is the tree search method built on individual measurements (Bagajewicz, 1997; this method is called "All Variables" method in chapter 2). This method is used for the sole purpose of validating the results obtained by cutset-based tree search method; hence no stopping criterion is used. The Madron problem contains 24 streams, hence the total number of candidate solutions is 2^{24} -1 = 16.78 millions. We attempted to solve the design case study 4.2.1

using this method, after one month (30 days) running time, the computational process is terminated. When stopped, the tree search explored only 4.19 millions of candidate solutions (and was able to identify the optimal solution shown in row 2 of table 4.7), hence the estimated computation time of this method is 120 days (4 months!). Computational time in other design case studies should be at the same magnitude with this computational time (120 days). Thus, this method is applicable for small scale problems only.

4.11.2. Genetic Algorithm

Performance of the GA method is shown in table 4.8. In table 4.8, the second and third columns show details (the number and the location of sensors) of the best solutions obtained by GA method. The fourth column shows objective values of these solutions. For comparison, the optimal objective value (summation of sensors cost and financial loss shown in table 4.7) is also shown in column five. The last column shows computational time of the GA method.

Table 4.8- Performance of GA method

Case study	Number of sensors	Measured variables	Objective value	Optimal objective value	Computation Time
4.2.1	13	1, 2, 3, 4, 8, 9, 10, 12, 13, 14, 17, 20, 23	568.1	552.1	54 min
4.2.2	13	1, 2, 3, 4, 5, 6, 8, 10, 12, 13, 19, 20, 22	653.6	629.4	54 min
4.2.3	4	2, 6, 19, 24	457.1	457.1	25 min
4.2.4	6	4, 8, 9, 17, 21, 23	349.9	330	17 min
4.2.5	8	4, 5, 7, 13, 14, 19, 21, 24	614.8	594.1	20 min
4.2.6	15	1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 13, 18, 19, 20, 24	686.9	673.7	32 min
4.2.7	19	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 18, 19, 20, 23, 24	1104.7	1101.2	55 min
4.2.8	12	1, 2, 3, 4, 7, 8, 10, 12, 13, 19, 20, 24	695.8	695.8	38 min
4.2.9	18	1, 2, 3, 4, 6, 7, 8, 9, 10, 12, 13, 14, 16, 17, 19, 20, 23, 24	1111.7	1110.3	39 min
4.2.10	23	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24	1775.5	1773.8	59 min

As shown in table 4.8

- Although the GA method does not guarantee optimality, it is able to locate optimal solution in two design cases 4.2.3 and 4.2.8. Moreover, in the other three design cases (4.2.7, 4.2.9 and 4.2.10), the best solutions obtained by GA are “very good”: they are very near to the optimal solutions
- Computational time of the GA method is acceptable: it solves this problem within an hour.

- In general, performance of the GA method is acceptable. Additionally, the GA method does not exhibit scaling problem (computational time does not increase exponentially with the size of the problem). To increase the chance of locating optimal solution, one can adjust the GA parameters (increase the size of population and / or mutation rate); or simply re-run GA many times (each GA run usually gives a different result).

4.11.3. Cutset-based tree search method

Performance of the cutset-based tree search method is shown in table 4.9

Table 4.9- Performance of cutset-based method

Case study	Number of cutsets	Number of nodes explored		Computational time
		With stopping criterion	No stopping criterion	
4.2.1	99	46,042	46,042	9 hrs 4 min
4.2.2	97	64,773	64,781	11 hrs 44 min
4.2.3	108	38,070	38,070	4 hrs 20 min
4.2.4	105	28,178	28,178	2 hrs 45 min
4.2.5	105	34,134	34,134	3 hrs 31 min
4.2.6	102	39,552	39,552	7 hrs 57 min
4.2.7	102	39,552	39,552	7 hrs 56 min
4.2.8	108	38,365	38,365	8 hrs 2 min
4.2.9	108	38,365	38,365	8 hrs 3 min
4.2.10	108	38,365	38,365	8 hrs 1 min

Column 2 of table 4.9 shows the number of cutsets (containing at least one key variable) in the corresponding design problems, the last column shows computational time of cutset-based method when stopping criterion is used. When stopping criterion is NOT used, the computational time is almost the same (the difference is usually not more than 5 minutes)

As can be seen from table 4.9:

- When the stopping criterion is used, the cutset-based method is able to locate optimal solutions (although optimality is not guaranteed if the stopping criterion is used)
- The stopping criterion has "little" effect: the number of nodes explored and computational time when stopping criterion is used are almost unchanged when compared with the case stopping criterion is NOT used. Only in the design case 4.2.2 that there is a *small* difference in number of nodes explored between the two cases (results of other case studies (not shown here) of Madron example also testify this fact). Thus, it may be not necessary to use stopping criterion in cutset-based tree search method. The parallel versions of cutset-based method will not use stopping criterion
- It may also be not necessary to use branching criterion (just put cutsets in numbered order as illustrated in figure 4.18b). The advantage of using branching criterion is that optimal solution is usually identified earlier than the case where branching criterion is NOT used: among the ten design case studies, there are six design cases where optimal solution is located within the first 20 nodes explored.

The disadvantage is that using branching criterion costs more time as shown in section 4.11.6

- Performance of cutset-based method is acceptable for this medium size Madron problem. However, because this method exhibits scaling problem, it is not efficient enough for large scale problems

4.11.4. Parallelized cutset-based method – parallelization of loops

As mentioned in section 4.10.2, the Intel Fortran compiler used in this research work automatically parallelizes loops in the Fortran program that implements cutset-based method. The results shown in table 4.10 are obtained simply by running the "serial" cutset-based method on the OSCER "super computer" (using 4 CPUs)

Table 4.10- Performance of cutset-based method with parallelization of loops

Case study	Computational time (sec)
4.2.1	2394
4.2.2	2834
4.2.3	1172
4.2.4	825
4.2.5	1005
4.2.6	2008
4.2.7	2007
4.2.8	2107
4.2.9	2106
4.2.10	2107

The results shown in table 4.10 and 4.9 shows that the simple approach to do parallel computing, the parallelization of loops, greatly reduces computational time (computational time reduces by the factor of 14 times, however, a part of this reduction is due to the fact that the configuration of OSCER super computer is much better than the PC used in this work)

4.11.5. Performance of parallelization method – SIMD approach

Performance of the parallelized cutset-based method - SIMD approach (described in section 4.10.4) is shown in tables 4.11 and 4.12. As shown in figure 4.18a, *Nc* is an important parameter that can be varied. If *Nc* = 1, all combinations of *Nc* cutsets are [1], [2], [3],...; all solutions originated from root [1] will be evaluated in worker node 1, solutions originated from root [2] will be evaluated in worker node 2, etc. (the case *Nc* = 2 is illustrated in section 4.10.4). If totally there are n_t cutsets, then the number of possible combinations of *Nc* cutsets is given by

$$C_{Nc}^{nt} = \frac{n_t!}{N_c!(n_t - N_c)!} \tag{4-11}$$

There are roughly 100 cutsets in this Madron problem, thus if *Nc* = 1: $C_1^{100} = 100$; if *Nc* = 2: $C_2^{100} = 4950$; if *Nc* = 3: $C_3^{100} = 161700$. Because C_3^{100} is already greater than the total number of candidate solutions (not more than 70,000) so the case *Nc* = 3 is not considered. Results for the case *Nc* = 1 and *Nc* = 2 are shown in table 4.11 and 4.12 respectively. The results are obtained using 64 computer nodes (64 CPUs), which comprises of one master node and 63 worker nodes

Table 4.11- Performance of parallelized program – SIMD approach (*Nc* = 1)

Case study	Computational time (sec)	Number of candidate solutions explored			
		Total	Max – one node	Min – one node	Total – serial program
4.2.1	2266	82,617	23,539	105	46,042
4.2.2	2623	99,674	33,528	143	64,781
4.2.3	1083	69,235	19,387	252	38,070
4.2.4	771	52,638	14,251	214	28,178
4.2.5	933	60,658	17,154	209	34,134
4.2.6	1333	72,848	19,920	195	39,552
4.2.7	1333	72,848	19,920	195	39,552
4.2.8	2017	69,595	19,566	252	38,365
4.2.9	2016	69,595	19,566	252	38,365
4.2.10	2016	69,595	19,566	252	38,365

Table 4.12- Performance of parallelized program – SIMD approach (*Nc* = 2)

Case study	Computational time (sec)	Number of candidate solutions explored			
		Total	Max – one node	Min – one node	Total – serial program
4.2.1	1609	152,427	12,794	278	46,042
4.2.2	1513	253,796	17,976	3,102	64,781
4.2.3	747	196,069	10,218	2,588	38,070
4.2.4	767	159,078	7,339	2,186	28,178
4.2.5	641	173,509	9,109	2,287	34,134
4.2.6	768	170,223	10,809	1,432	39,552
4.2.7	768	170,223	10,809	1,397	39,552
4.2.8	1391	148,292	10,321	245	38,365
4.2.9	1392	148,292	10,321	245	38,365
4.2.10	1392	148,292	10,321	245	38,365

In tables 4.11 and 4.12:

- Column 3 shows the total number (i.e. the summation) of all candidate solutions that have been evaluated in all worker nodes plus the master node. For comparison, the total number of candidate solutions explored in serial program (table 4.9) is also re-shown in the last column
- Column 4 shows the maximum value of the numbers of candidate solutions evaluated in one worker node, while the corresponding minimum value is shown in column 5. The maximum value is always realized in worker node number 1 (that explores solutions starting from root [1] (Nc = 1) or [1 2] (Nc = 2)).

The results shown in tables 4.11 and 4.12 show that

- Because the worker nodes do not communicate with one another, there is a high chance of repetition of job (that is, the same candidate solution is evaluated in at least two worker nodes). This repetition of task explains why the total number of candidate solutions explored when SIMD parallelization is used (column 3 of the tables) is more than the corresponding value when SIMD parallelization is NOT used (column 6). The chance of repetition of job (and the total number of candidate solutions explored, shown in column 3) increases when the parameter Nc increases
- The SIMD approach offers only a small improvement in computational time: when compared with the case when SIMD parallelization is NOT used (that is, the results shown in table 4.10): the option (Nc = 1) offers roughly 5% improvement in computational time while the option (Nc = 2) offers roughly 40% improvement.

- The poor performance of SIMD parallelization is due to the poor load balancing of this approach in this specific problem. The SIMD parallelization achieves the best performance if all the worker nodes process the same amount of job so that all worker nodes finish their jobs at the same time. If this situation is realized, the parallelization is said to have good balancing of jobs. If this is not the case (i.e. poor balancing), the worker node with the heaviest amount of job will finish last (and worker nodes with small amount of job will finish early and become idle until the overall computation process completes, which means that the resource is not fully utilized). In such case (poor balancing), the computational time of the overall process is determined by the worker node with the heaviest amount of job.
- The load balancing property of the SIMD parallelization is indicated by the difference in the numbers of candidate solutions evaluated in worker nodes. One can see that there is a large difference between the maximum and minimum value of the numbers of candidate solutions evaluated in a worker node (shown in columns 4 and 5 of the tables), hence the balancing of jobs is poor
- The option $Nc = 2$ has a better load balancing property than the option $Nc = 1$, so option $Nc = 2$ has better performance
- The performance of SIMD parallelization can be improved by improving the load balancing of the parallelized program, which is left for future work.

4.11.6. Performance of parallelization method – MISD approach

Results of the case where branching criterion is used are shown first. The results are obtained using 64 computer nodes (one master and 63 worker nodes). Table 4.13 shows results when decomposition technique is NOT used.

Table 4.13- Performance of MISD approach – With branching criterion

Case study	Computational time (sec)	Number of candidate solutions evaluated	
		Min - one node	Max - one node
4.2.1	126	600	800
4.2.2	192	800	1300
4.2.3	90	400	700
4.2.4	53	300	1200
4.2.5	74	400	1000
4.2.6	89	400	900
4.2.7	90	400	900
4.2.8	101	500	700
4.2.9	102	500	700
4.2.10	101	500	700

The results shown in tables 4.11, 4.12 and 4.13 show that:

- The MISD approach is much better than the SIMD approach: computational time of the MISD approach is 12 times smaller than that of the SIMD approach. If compared against the base case where MPI parallelization is not used (table 4.10), the MISD approach reduces computational time by the factor of about 20 times (this result is obtained using 64 computer nodes)

- The load balancing property of MISD approach is much better than the SIMD approach, one can see that the difference between the maximum and minimum value of solutions evaluated in a computer node is small

Let us now consider the case where decomposition technique is used. Three decomposition options are considered for this case. They are described in table 4.14

Table 4.14- Decomposition option

Option	Name	Number of sub-graphs	Locations of connecting streams
1	Single decomposition	2	$\{S_8, S_{18}\}$
2	Double decomposition	3	$\{S_3\}$ and $\{S_{16}\}$
3	Multiple decomposition	7	$\{S_3\}$, $\{S_{15}\}$, $\{S_{16}\}$, $\{S_{17}\}$, $\{S_{19}\}$, $\{S_{20}\}$,

For example, in option 1, the process flowsheet is "cut" at position between unit 2 and 3 (the connecting streams are S_8 and S_{18}): the process graph is decomposed into two sub-graphs, one contains five units 1, 2, 6, 7, 10 and the other contains six units: 3, 4, 5, 8, 9, 11.

Table 4.14 shows results when decomposition technique is used. In this table, columns 4 and 7 shows the number of candidate solutions generated in master node, which are then sent to worker nodes for evaluating (denoted as "number of solutions generated"). When decomposition technique is used, because of the extra step of "explore all possibilities of removing connecting streams out of that candidate solution" (described in section 4.10), the number of candidate solutions evaluated will be greater than the solutions sent to worker node from master node. For example, if an element in

the *list100* (a candidate solution) is [1 2 3 4 5] and [4 5] are connecting streams, then for this specific candidate solution the following four solutions need to be evaluated: [1 2 3 4 5], [1 2 3 4], [1 2 3 5] and [1 2 3]. Thus the total number (obtained by summation) of solutions that have been evaluated in all worker nodes (shown in columns 3 and 6) must be greater than the number of solutions generated in master node (columns 4 and 7).

Table 4.15- Performance of MISD approach with decomposition and branching criterion

Case study	Option 1			Option 3		
	Comput. time (sec)	Number of solutions evaluated	Number of solutions generated	Comput. time (sec)	Number of solutions evaluated	Number of solutions generated
4.2.1	99	87,451	22,337	833	713,861	19,272
4.2.2	111	140,528	36,142	821	598,321	15,889
4.2.3	66	72,112	18,243	813	580,569	15,086
4.2.4	62	50,513	12,739	737	484,968	13,836
4.2.5	67	63,455	16,182	737	430,397	12,447
4.2.6	89	81,132	20,997	817	447,922	12,517
4.2.7	89	81,132	20,997	815	447,922	12,517
4.2.8	94	72,508	18,325	790	577,835	14,929
4.2.9	95	72,508	18,325	790	577,835	14,929
4.2.10	94	72,508	18,325	791	577,835	14,929

The large difference between the number of solutions evaluated in this case (columns 3 and 6 of table 4.15) and the total number of candidate solutions (column 4 of table 4.9) is due to the fact that there is repetition of job when decomposition is used. Take for example the above illustration (where [4 5] are connecting streams), if the mentioned candidate solution [1 2 3 4 5] is processed in one worker node while another

candidate solution [1 2 3 4] is processed in another worker node, then these two worker nodes evaluate the same two solutions: [1 2 3 4] and [1 2 3]. The chance of job repetition and the total number of solutions evaluated increase when the number of decomposition (how many times the process graph is "cut") increases as clearly shown in table 4.15.

Using decomposition has two opposite effects: the good side is that it reduces the time to generate candidate solutions in the master node, the bad side is that it increases the time to evaluate solutions in worker nodes because of the problem of job repetition. Table 4.15 shows that when branching criterion is used, a small number of decomposition is beneficial: the option 1, single decomposition reduces computational time by 10% (when compared with the base case where no decomposition is used, table 4.13). However, a large number of decomposition (option 3) has adverse effect: it increases computational time because of the problem of job repetition

The results when branching criterion is not used are shown in table 4.16. The second column of this table shows computational time for the case when no decomposition is used while the last three columns show computational times for the three decomposition options described in table 4.14

Table 4.16- Performance of MISD approach with decomposition, no branching criterion

Case study	Computational time (sec)			
	Number of decomposition			
	None	Single	Double	Multiple
4.2.1	63	97	78	824
4.2.2	97	110	74	832
4.2.3	47	66	67	830
4.2.4	22	62	61	738
4.2.5	37	60	65	858
4.2.6	47	86	66	858
4.2.7	47	86	65	858
4.2.8	57	93	67	805
4.2.9	57	93	68	804
4.2.10	57	93	68	804

Table 4.16 shows that, when no branching criterion is used:

- The computational time improves significantly (compared with the base case when branching criterion is used, computational time improves 50%).
- The bad effect of using decomposition (the problem of job repetition) overshadows the benefit (less time to generate candidate solutions) and computational time generally increases when decomposition technique is used

Dependence of computational time of this method (when no branching criterion and no decomposition is used) on the number of computer nodes utilized is shown in table 4.17

Table 4.17- Performance of MISD vs. number of CPUs

Case study	Computational time (sec)		
	$N_{CPU} = 32$	$N_{CPU} = 64$	$N_{CPU} = 96$
4.2.1	100	63	52
4.2.2	139	97	84
4.2.3	60	47	39
4.2.4	36	22	22
4.2.5	50	37	30
4.2.6	73	47	41
4.2.7	74	47	41
4.2.8	88	57	48
4.2.9	89	57	48
4.2.10	88	57	48

Table 4.17 shows that if more computer nodes are used (i.e. more "workers" to share the tasks), computational time is reduced. However, it is well known that the dependence of performance on number of computer nodes is not linear: the performance improvement becomes smaller as more computer nodes are used: when number of computer nodes increases from 32 to 64 and from 64 to 96 (32 nodes added), performance (computational time) improves 36% and 15% respectively.

The best performance (column 4 of table 4.17) of the MISD parallelization is achieved when: i) no branching criterion and no decomposition is used, ii) using as many computer nodes as possible. The MISD parallelization is a great improvement over the simple exhaustive tree search method and the serial cutset-based method.

4.11.7. Performance of parallelization method – MIMD approach

Performance of the MIMD parallelization method is shown in table 4.18 using the following parameters: i) *Nc* (section 4.10.6) = 2, ii) the number of "managing" computer nodes (in group one) = 4; thus the number of computer nodes in a sub-group (belonging to group two) is 15, that is, one "managing" computer node generates candidate solutions and then sends them to 15 computer nodes to evaluate them.

Table 4.18- Performance of MIMD parallelization method

Case study	64 CPUs				96 CPUs			
	Candidate solutions explored			Time (sec)	Candidate solutions explored			Time (sec)
	Max - node	Min - node	Total		Max - node	Min - node	Total	
4.2.1	21,037	9,958	71,213	111	19,948	9,958	69,761	74
4.2.2	28,360	9,731	91,529	107	28,355	9,731	91,541	73
4.2.3	21,812	6,634	55,588	57	20,196	6,236	56,958	40
4.2.4	12,624	7,336	52,055	57	12,580	7,336	51,962	40
4.2.5	19,695	4,754	47,429	48	18,392	4,754	51,570	30
4.2.6	19,721	12,611	69,970	59	18,493	13,202	71,782	47
4.2.7	19,808	12,677	69,710	59	18,412	13,191	71,655	48
4.2.8	19,233	5,382	61,278	99	19,272	5,382	61,262	68
4.2.9	19,151	5,382	61,195	101	19,186	5,382	61,227	69
4.2.10	19,279	5,382	61,339	101	19,009	5,382	60,761	69

Columns 2, 3, 6, 7 ("Max-node", "Min-node") of table 4.18 show the maximum and minimum value of candidate solutions explored in one "*managing*" node while columns 4 and 8 show the total number of candidate solutions evaluated in the process

(summation of all candidate solutions explored in all "*managing*" nodes plus master node)

The results shown in table 4.18 show that:

- As shown in section 4.11.5 (SIMD approach), dividing problem data suffers from the problem of job repetition (a same candidate solution is evaluated in at least two computer nodes) such that the total number of candidate solution is more than that when no parallelization is used (table 4.9). This approach also divides problem data so it also suffers from the problem of job repetition.
- Because this approach (MIMD) divides both the problem data and problem instruction so reasonably it should be better than the SIMD and MISD approach. However, because of the problem of job repetition this approach is not necessary better than the MISD (multiple instruction single data) approach. In comparison with the MISD approach:
  i. The MIMD approach uses more resources to generate candidate solutions: four CPUs in group one (in MIMD) vs. one CPU (master node) in MISD, so the MIMD approach should spend less time to generate candidate solutions.
  ii. Because of the problem of job repetition, the number of candidate solutions that need to be evaluated in MIMD parallelization is more than that in the MISD approach. Moreover, available resource (CPUS) for this purpose in MIMD approach is less than that in MISD approach: 59 (64 – 1 master node – 4 CPUs in group one) vs. 63 (64 – 1 master node). Thus, the MIMD approach should spend more time for this purpose.

iii. Comparison of performance of these two approaches can not be concluded unless a large number of tests have been conducted. For this specific Madron problem with this specific configuration (i.e. when 4 "managing" nodes are used), the MISD parallelization is better than the MIMD approach.

- The load balance of this approach is pretty good (the difference between the maximum and minimum value of candidate solutions evaluated in one node is small) and when using more computer nodes, computational time decreases.
- Because of the dynamic nature of the process (real time checking of worker nodes' status and assigning jobs), the number of candidate solution explored is usually different if the computer system on which the process runs is different (e.g. changing the number of CPUs) as shown in table 4.18

The effect of varying number of computer nodes ("managing" nodes) in group one (M) is shown in table 4.19 using totally 96 CPUs. Three cases are considered ($M = 2$, $M = 4$, $M = 8$), which are shown in table 4.19. Take for example the second case (M = 4): each of the first thee "managing" CPU controls 23 worker nodes while the last "managing" CPU controls 22 worker nodes (so the total number of CPUs is 1 master node + 4 "managing" CPUs + 3*23 + 22 = 96)

Table 4.19- Performance of MIMD parallelization at varied number of managing nodes

Case study	Total candidate solutions explored			Computational time (sec)		
	M = 2	M = 4	M = 8	M = 2	M = 4	M = 8
4.2.1	65,373	69,761	102,555	54	74	148
4.2.2	81,505	91,541	137,974	62	73	142
4.2.3	50,594	56,958	89,230	25	40	73
4.2.4	41,652	51,962	79,258	25	40	74
4.2.5	48,975	51,570	81,744	18	30	64
4.2.6	58,920	71,782	83,669	46	47	79
4.2.7	59,410	71,655	83,916	47	48	78
4.2.8	51,973	61,262	89,713	39	68	133
4.2.9	51,864	61,227	89,873	39	69	134
4.2.10	51,973	60,761	89,613	39	69	134

It can be seen from table 4.19 that computational time increases when the number of managing nodes (*M*) increases. This is because when *M* increases:

i) The problem of job repetition become more severe (the total number of candidate solutions explored, shown in columns 2,3 and 4 of table 4.19, increases when *M* increases)

ii) Less worker nodes to evaluate the candidate solutions (at the same total number of CPUs)

It is recommended to use 2 managing nodes only; for large scale problems (e.g. CDU example shown below), one can use up to 4 managing nodes or use decomposition technique.

If decomposition technique is used, the calculation procedure (without decomposition) is modified at the same way as described in section 4.10.5, that is, the

step "Evaluate all 100 candidate solutions stored in *list100*" (in worker nodes in group two) now comprises of two steps: removing connecting streams out of a candidate solution and then evaluate all the new candidate solutions resulted from that operation. The results when decomposition technique is used are shown in table 4.20; only one decomposition option (double decomposition, shown in column 3 of table 4.14) is considered (using 96 CPUs, among which two are managing nodes). For comparison, the results when decomposition is NOT used (and at the same configuration, 96 CPUs - two managing nodes) are also shown in table 4.20

Table 4.20- Performance of MIMD parallelization with decomposition technique

Case study	Total candidate solutions explored		Computational time (sec)	
	No decomposition	Double decomposition	No decomposition	Double decomposition
4.2.1	65,373	103,295	54	80
4.2.2	81,505	106,344	62	75
4.2.3	50,594	73,579	25	67
4.2.4	41,652	59,793	25	60
4.2.5	48,975	60,069	18	64
4.2.6	58,920	73,169	46	66
4.2.7	59,410	75,557	47	67
4.2.8	51,973	70,778	39	67
4.2.9	51,864	68,433	39	67
4.2.10	51,973	68,433	39	67

As can be seen from table 4.20, using decomposition technique costs more time (this observation is also realized for the other decomposition options, option 1 and 3). As

explained above, using decomposition has two opposite effects: i) reduce the time to generate candidate solutions (in master node and "managing" nodes) and ii) increase the time to evaluate candidate solutions (in worker nodes) due to the problem of job repetition. For this specific Madron problem, when decomposition is used, the number of cutset in the problem reduces about 3.3 to 6 times (from about 100 cutsets to 30 cutsets in decomposition option 1 and about 16 cutset in decomposition option 3). For this specific Madron problem, it is not beneficial to use decomposition because using decomposition increases computational time. However, for the large scale CDU example shown below, it is advisable to use decomposition because decomposition leads to a great reduction in number of cutsets.

Of all the parallelization methods and all options that have been considered, the following option gives the best performance (the shortest computational time): MIMD approach with no branching criterion and no decomposition, using 96 computer nodes in total with 2 managing nodes

It can be concluded from the above results that the MIMD and MISD approach are much better than the SIMD approach and it is always better to use more computer nodes (more resource). Additionally, it is better not to use branching criterion. There is no final conclusion regarding the following two issues:

i. Which parallelization method is better, MIMD or MISD
ii. Whether it is beneficial to use decomposition technique

As mentioned above, compared with the MISD (multiple instruction single data) approach as base case, there are two opposite effects of also dividing the problem data

(that is, the MIMD approach): the time to generate candidate solutions decreases while the time to evaluate candidate solutions increases. The same thing is said about using decomposition (when compared against the counterpart where decomposition is NOT used). The trade-off (final result) of these two opposite sides depends on the specific problem under consideration. The following section gives an intuitive guideline on which option is the best choice. Assuming that there are *Nw* worker nodes available, the criterion to determine the best option is based on the analysis of relative computational speed of the following two tasks: generating (*Nw**100) candidate solutions and evaluating *list100* (that was sent from master node to worker nodes). Figure 4.24 shows the two extreme cases that can occur for the base case where MISD parallelization method and no decomposition are used.

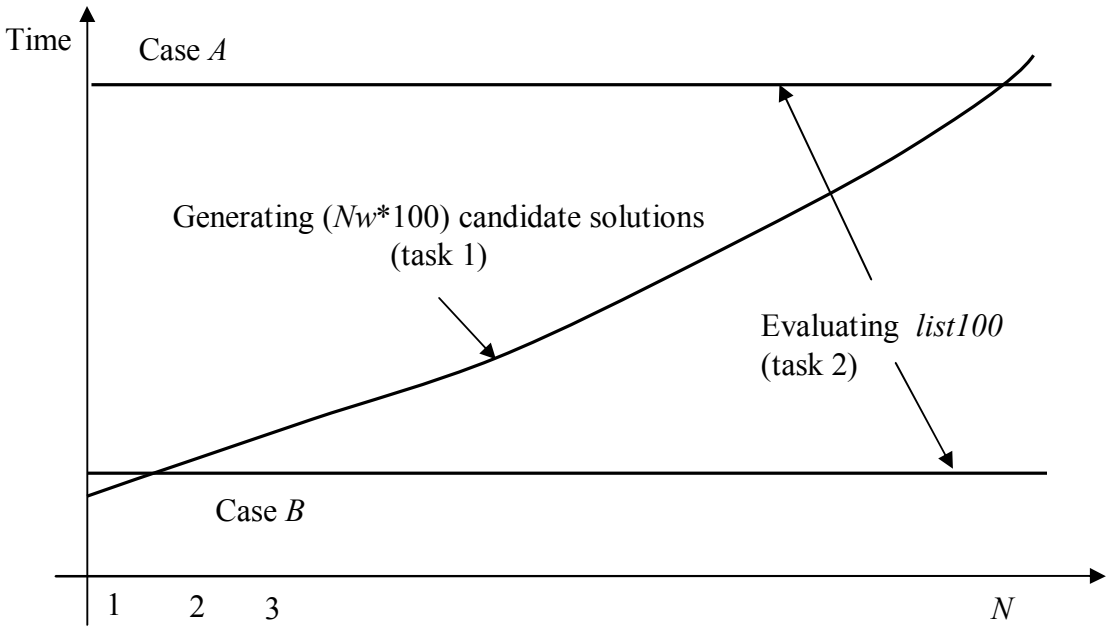


Figure 4.24 – Analysis of performance of parallelization method

Notes:

- The points 1, 2, 3,…, *N* in x-axis indicate the total number of candidate solutions generated in the process: at point *N* the total number of candidate solutions generated is *N**(*Nw**100). As discussed above, the time to generate (*Nw**100) candidate solutions increases progressively along with the computational process.
- "Evaluating *list100*" refers to the task of evaluating the 100 elements of the list *list100*. If no decomposition is used, this task is simply evaluating the 100 candidate solutions contained in the list. If decomposition is used, the actual number of candidate solutions that need to be evaluated is more than 100 (because "evaluating *list100*" now comprises of two steps as described above) and the time for this task increases (compared against the base case where decomposition is NOT used) as illustrated in figure 4.25

We now discuss possible options to reduce computational time for the two extreme cases:

- Case A: "Evaluating *list100*" takes significant time so any option that suffers from the problem of job repetition (MIMD parallelization method and decomposition technique) would increase computational time. The best option for this case is MISD approach without decomposition.
- Case B: The task "Evaluating *list100*" (task 2) is a lot faster than the other task (generating candidate solutions, task 1) so the task 1 is the dominating (limiting) factor. Thus the options that reduce the time for task 1 would reduce overall computational time of the process; which are MIMD parallelization method and

decomposition technique (use either of them or use both). The effect of using these two options is illustrated in figure 4.25

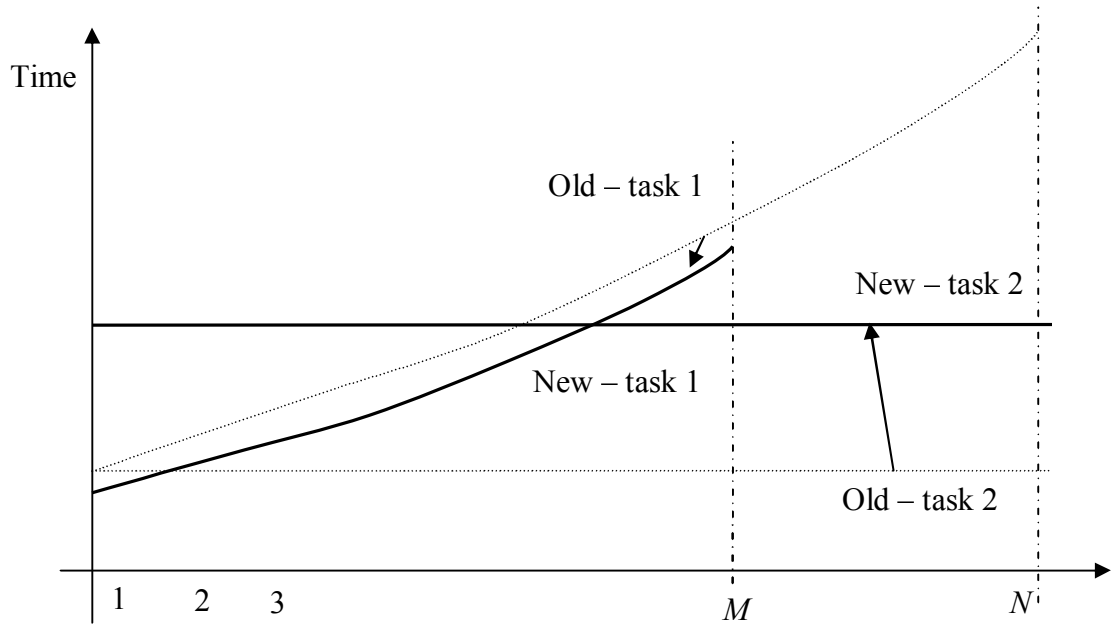


Figure 4.25 –Performance of parallelization method - with MIMD and/or decomposition

The MIMD parallelization method and decomposition technique reduce time for task one (generating candidate solutions) because

- This task is shared among several "managing" nodes (MIMD approach) instead of only one computer node (master node) in MISD approach
- Or decomposition technique helps reduce the total number of candidate solutions generated (as illustrated in figure 4.25: the "new" value of total number of candidate solutions generated is $M^*(Nw^*100) <$ the old value $= N^*(Nw^*100)$).

For the cases in between these two extreme cases (i.e. computational times for the two tasks are at the same magnitude), which option is the best choice can only be determined from actual testing.

4.12. Example 4.3 – CDU example

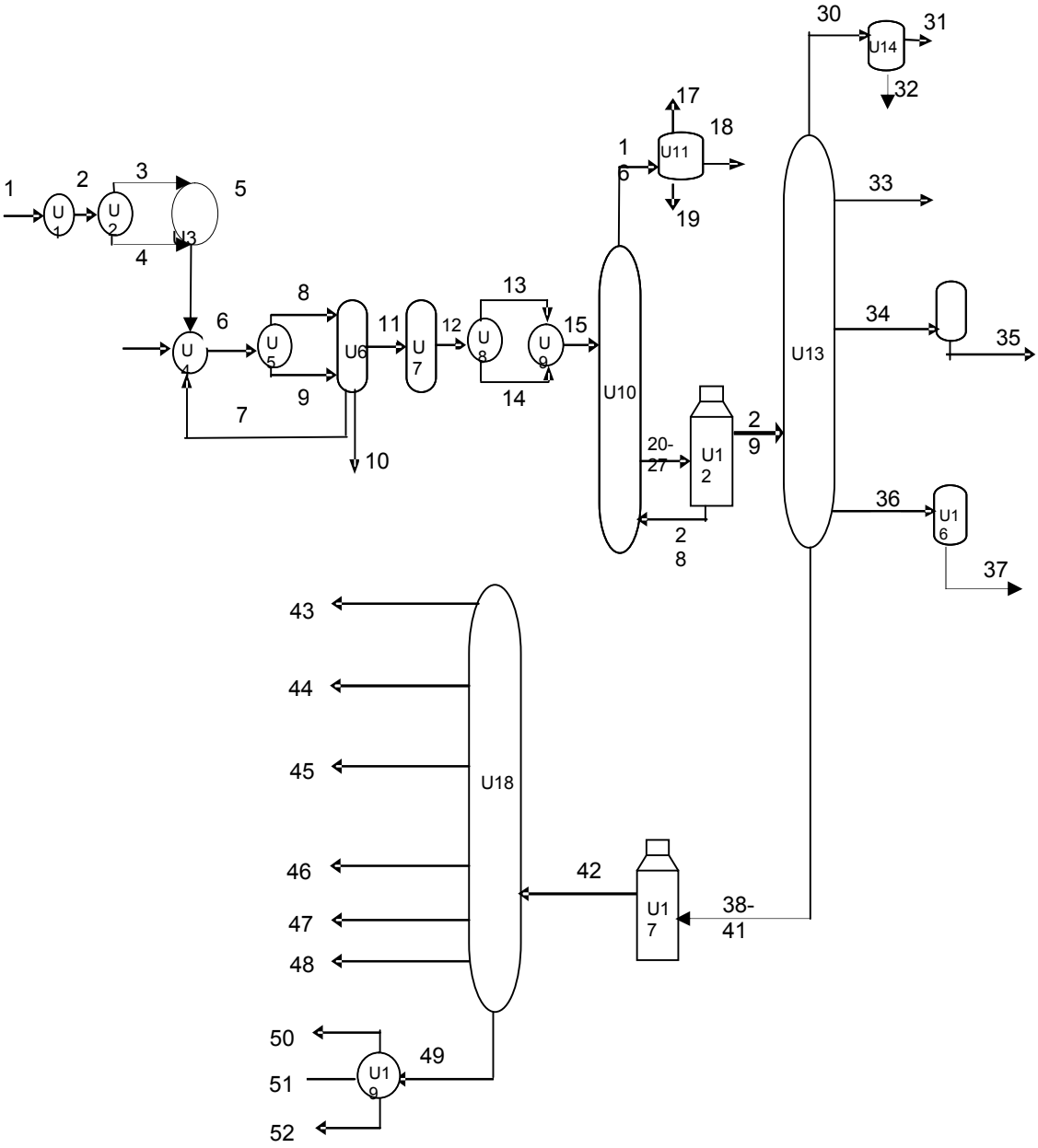


Figure 4.26 –Process flowsheet – CDU example

The CDU example introduced in Gala and Bagajewicz (2006) is considered next. The process flowshheet is shown in figure 4.26 and the problem data is shown in table 4.21

Table 4.21- Data for CDU Example

Streams	Flow	Cost	Streams	Flow	Cost
S_1	413349	2000	S_{27}	60413	2000
S_2	419579	2000	S_{28}	103939	1800
S_3	209316	1800	S_{29}	386580	1500
S_4	210262	1800	S_{30}	57169	2300
S_5	419579	2200	S_{31}	45829	2100
S_6	460520	2100	S_{32}	4202	1800
S_7	26510	2100	S_{33}	26133	2200
S_8	230650	1700	S_{34}	73900	2200
S_9	229870	1700	S_{35}	73704	2000
S_{10}	26243	2400	S_{36}	50851	2200
S_{11}	413650	2000	S_{37}	50715	2200
S_{12}	413650	2000	S_{38}	45902	2000
S_{13}	206932	1800	S_{39}	45878	2000
S_{14}	206717	1800	S_{40}	45928	2000
S_{15}	413650	1500	S_{41}	45851	2000
S_{16}	27068	2300	S_{42}	185593	2300
S_{17}	5124	2200	S_{43}	38557	1800
S_{18}	21467	2200	S_{44}	18932	1800
S_{19}	478	1800	S_{45}	19846	1800
S_{20}	61562	2000	S_{46}	23880	2100
S_{21}	60985	2000	S_{47}	18196	2100
S_{22}	61253	2000	S_{48}	18106	2100
S_{23}	61490	2000	S_{49}	48081	2300
S_{24}	61109	2000	S_{50}	15154	2000
S_{25}	60796	2000	S_{51}	20268	2000
S_{26}	62012	2000	S_{52}	12659	2000

Information used in the calculation of financial loss is as follows:

- Probability of sensors = 0.1 (for all sensors)
- Biases (in failed sensors) are assumed to follow normal distribution with zero means and standard deviations = four times the standard deviations of measurements (for all sensors)
- Windows time of analysis T = 30 days

Only one design case study is considered; the design case study and the obtained solution are described in table 4.22

Table 4.22- Results for CDU Example

Key variables	S_{31}, S_{33}, S_{35}, S_{37}, S_{43}, S_{44}
Ks values	Ks_{31} = 400, Ks_{33} = 360, Ks_{35} = 350, Ks_{37} = 340, Ks_{43} = 250, Ks_{44} = 240
Measured variables	S_{31}, S_{33}, S_{34}, S_{35}, S_{36}, S_{43}, S_{44}
Cost	14300
Financial loss	11566.3
Total number of candidate solutions explored	3.1 millions
Computational time	10 hrs 5 min

For this problem, if decomposition is not used, the number of cutsets (containing key variables) is 973 while if decomposition is used, the number of cutsets is reduced to 158 (single decomposition, connecting stream =S_{15}) and 69 (double decomposition, connecting streams = S_{11} & S_{29}). Thus, simple decomposition strategies like those

described greatly reduces the number of cutsets (6.2 times for single decomposition and 14.1 times for double decomposition).

The following options are used to solve the problem:

- MIMD parallelization method using 200 CPUs in total, among which two are managing nodes
- Double decomposition, connecting streams = S_{11} & S_{29}

It takes a lot more time to solve this 52-stream CDU example than the 24-stream Madron example (10 hours vs. 1 minute). However, the computational time is still acceptable

4.13. Conclusions

In this chapter, two new approaches to design sensor networks for process monitoring purpose are presented. These two new approaches are based on software accuracy and its associated economic value. Efficient methods to solve the proposed problems are presented, among which the parallelized cutset-based method is proven to be a very efficient method to solve the value-optimal sensor network design problem.

4.14. References

Bagajewicz, M., Design and Retrofit of Sensors Networks in Process Plants. *AIChe J.* **1997**, 43(9), 2300-2306.

Bagajewicz M. On the Definition of Software Accuracy in Redundant Measurement Systems. *AIChE Journal*. **2005a**; 51(4), 1201-1206.

Bagajewicz, M. On a New Definition of a Stochastic-based Accuracy Concept of Data Reconciliation-Based Estimators. 15th *ESCAPE proceeding (European Symposium on Computer-Aided Process Engineering)*, Spain, **2005b**, pp. 1135-1141

Bagajewicz M. and Q. Jiang. Gross Error Modeling and Detection in Plant Linear Dynamic Reconciliation. *Comp. Chem. Eng*., **1998**, 22(12), 1789-1810

Bagajewicz M., M. Markowski and A. Budek. Economic Value of Precision in the Monitoring of Linear Systems. *AIChE Journal*. **2005**, 51(4), 1304-1309.

Bagajewicz, M. Value of Accuracy in Linear Systems. *AIChE Journal*. **2006**, 52(2), pp. 638-650.

Bagajewicz M. and DuyQuang Nguyen. Stochastic-Based Accuracy of Data Reconciliation Estimators for Linear Systems. *Computers and Chemical Engineering.* **2008***,* 32(6), 1257-1269.

Gala, M. and Bagajewicz, M. J. Efficient Procedure for the Design and Upgrade of Sensor Networks Using Cutsets and Rigorous Decomposition. *Ind. Eng. Chem. Res*. **2006**; 45(20), 6679-6686.

Haupt, R.L. and S.E. Haupt. *Practical Genetic Algorithms*, 2nd Edition, Wiley-Interscience: New Jersey, USA, **2004**

Madron, F., and V. Veverka, Optimal Selection of Measuring Points in Complex Plants by Linear Models, *AIChe J.* **1992**, 38(2)**,** 227.

Nguyen Thanh D.Q., Siemanond K., Bagajewicz M.J. Downside Financial Loss of Sensor Networks in the Presence of Gross Errors. *AIChE J.* **2006**, 52(11), pp. 3825-3841.

Nguyen D.Q and Bagajewicz M. On The Impact of Sensor Maintenance Policies on Stochastic-Based Accuracy. *Comp. Chem. Eng*. **2009**, 33(9), 1491-1498

Pacheco P.S. *Parallel Programming with MPI*. Morgan Kaufmann Publishers: San Francisco, USA, **1997**.

5. CONCLUSIONS AND FUTURE WORKS

The research work aims to achieve the following two objectives:

i. Developing efficient computational methods to solve realistic large scale nonlinear sensor network design problems

ii. Studying and proposing efficient methods to design sensor networks that simultaneously optimize performance (using economic value of accuracy as performance measure) and cost of sensor network.

For the first objective, although a perfect solution (i.e. an efficient method that guarantees optimality) can not be found, a variety of "good" solutions are presented. The equation-based method guarantees optimality but it is not efficient enough for realistic large scale problems. The same thing is said for the level traversal tree search. The approximate method is very efficient. Although the approximate method does not guarantee optimality, the chance of finding optimal solution is very high. Additionally, the inverted tree search strategy tailored for problems with high level of specifications is also presented (it also guarantees optimality).

For the second objective, two methods that can run on a personal computer are proposed. The genetic algorithm is satisfactorily efficient but it does not guarantee optimality. In the opposite side, the cutset-based method guarantees optimality but it is not efficient enough for large scale problems. The last proposed method, the parallelized cutset-based

method is very efficient and it guarantees optimality (although it has one small disadvantage: it has to be run on a super computer).

As can be noted throughout this dissertation, this work studies the problem of designing sensor network for process monitoring purpose only. The same approach (maximizing value of sensor network) can be applied to design sensor network for other purposes like process fault diagnosis. This is left for future work

APPENDIX A1

ΔD = DEFL(current node) – DEFL(sensor network with maximum number of sensors)

ΔC = Cost(sensor network with maximum number of sensors) - Cost (current node)

It is now shown that if *{ΔC- ΔD} of current node > {ΔC- ΔD} of previous node then the objective value of current node < the objective value of the previous node*

Using "c.node" and "pre.node" as abbreviated names for current node and previous node respectively, then

OBJ value (c.node) = Cost (c.node) + DEFL (c.node)

= Cost (pre.node) + DEFL (pre.node)

+ {Cost (c.node) – Cost (pre.node)}

+ {DEFL (c.node) – DEFL (pre.node)}

= OBJ value (pre.node)

+ {Cost (c.node) – Cost (MSN)} - {Cost (pre.node) – Cost (MSN)}

+ {DEFL(c.node) – DEFL(MSN)} - {DEFL(pre.node) – DEFL(MSN)}

Suppose that OBJ value (c.node) $\leq$ OBJ value (pre.node)

$\Rightarrow$ {Cost (c.node) – Cost (MSN)} - {Cost (pre.node) – Cost (MSN)}

+ {DEFL(c.node) – DEFL(MSN)} - {DEFL(pre.node) – DEFL(MSN)} $\leq$ 0

{DEFL(c.node) – DEFL(MSN)} - {DEFL(pre.node) – DEFL(MSN)} $\leq$

{Cost (MSN) - Cost (c.node)} – { Cost (MSN) - Cost (pre.node)}

$\Rightarrow \Delta D(c.node) - \Delta D(pre.node) \leq \Delta C(c.node) - \Delta C(pre.node)$

$\Rightarrow \{\Delta C(c.node) - \Delta D(c.node)\} \geq \{\Delta C(pre.node) - \Delta D(pre.node)\}$

Q.E.D

APPENDIX A2

List of publications

1) Bagajewicz M. and DuyQuang Nguyen. Stochastic-Based Accuracy of Data Reconciliation Estimators for Linear Systems. *Computers and Chemical Engineering*. **2008**, 32(6), 1257-1269

2) DuyQuang Nguyen and Bagajewicz M. On The Impact of Sensor Maintenance Policies on Stochastic-Based Accuracy. *Computers and Chemical Engineering*. **2009**, 33(9), 1491-1498.

3) DuyQuang Nguyen and Bagajewicz M. Design of Nonlinear Sensor Networks for Process Plants. *Industrial and Engineering Chemistry Research*. **2008**, 47(15), 5529-5542

4) DuyQuang Nguyen, C. Brammer and M. Bagajewicz. New Tool for the Evaluation of the Scheduling of Preventive Maintenance for Chemical Process Plants. *Industrial and Engineering Chemistry Research*. **2008**, 47(6); 1910-1924

5) DuyQuang Nguyen and M. Bagajewicz. Optimization of Preventive Maintenance in Chemical Process Plants. *Industrial and Engineering Chemistry Research*. **2009**, submitted

6) DuyQuang Nguyen and M. Bagajewicz. New Efficient Level Traversal Tree Search Methodology For The Design And Upgrade Of Sensor Networks. *AiChe journal*. **2009**, submitted

7) Barbaro A., DuyQuang Nguyen, N. Vipanurat and M. J. Bagajewicz. All-At-Once And Step-Wise Detailed Retrofit Of Heat Exchanger Networks Using an MILP Model, *Industrial and Engineering Chemistry Research*. **2009**, submitted

8) Bagajewicz M., Thang Cao, R. Crosier, S. Mullin, J. Tarver, and D.Q Nguyen. Method for Evaluation of Thermochemical and Hybrid Water-Splitting Cycles. *Industrial and Engineering Chemistry Research*. **2009**, 48 (19), pp 8985–8998

In preparation

DuyQuang Nguyen and M. Bagajewicz. Efficient Approximate Methods For The Design And Upgrade Of Sensor Networks

DuyQuang Nguyen and M. Bagajewicz. Value-optimal Sensor Network Design