Computers and Chemical Engineering 35 (2011) 1119–1134

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/compchemeng

Parallel computing approaches to sensor network design using the value paradigm

DuyQuang Nguyen, Miguel J. Bagajewicz*

University of Oklahoma, 100 E. Boyd St., T-335, Norman, OK 73019, United States

ARTICLE INFO

Article history:
Received 18 December 2009
Received in revised form 2 June 2010
Accepted 12 July 2010
Available online 18 July 2010

Keywords:
Instrumentation design
Sensor network design
Parallel computing

ABSTRACT

One previous paradigm was based on sensor positioning selection to allow key variable observability. Later, the problem was posed as a minimum cost problem subject to sensor network performance (precision and gross error robustness first and accuracy as a replacement later). In a previous paper (Nguyen and Bagajewicz, submitted for publication), we presented a new paradigm in sensor network design, one that maximizes value, defined as economic value of information minus its cost. We presented two methods to solve the problem and we concluded that their performance was acceptable for small and medium size problems, but not for large problems. In this paper we use parallel computing approaches for large problems.

1. Introduction

In a modern chemical plant, a large number of measurements are available as a result of digital data acquisition systems and reliable low-cost sensors. Such measured data are important for process monitoring, model identification, online optimization, and process control. Thus, the process plant performance depends on the available measurements and the problem of optimally selecting measurements location arises naturally. This problem is known as the sensor network design problem (SNDP). Research on sensor network design in the past three decades has been extensive.

The most common SNDP in literature is designing sensor network for process monitoring purposes, that is, 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. desired level of precision of estimators, observability and redundancy of key variables). Another common type of SNDP is to design sensor networks for process fault detection and isolation purpose. This type of SNDP has gained increasing attention from research community, the most recent works belong to this type are shown next. Bhushan, Narasimhan, and Rengaswamy (2008) presented a framework for designing robust sensor network for reliable process fault diagnosis; the problem was then solved by using constraint programming (Kotecha, Bhushan, & Gudi, 2007, 2008). Chen and Chang (2008) used graph-theoretic method to design sensor network for process fault identification. Finally, a sensor positioning for control purposes based on backed-off optimal regions has been proposed (Peng & Chmielewski, 2006).

Kretsovalis and Mah (1987) minimized a weighted sum of estimation error and measurement cost using a combinatorial search algorithm. Madron and Veverka (1992) used Gauss Jordan elimination; Meyer, Le Lann, Koehret, and Enjalbert (1994) and Luong, Maquin, Huynh, and Ragot (1994) used graph-theoretic methods. Bagajewicz (1997) was the first to formulate the sensor network problem as a mixed-integer programming (MINLP) model; Chmielewski, Palmer, and Manousiouthakis (2002) used a branch and bound method with linear matrix inequalities (LMI) transformation to obtain a solution. Carnero, Hernandez, Sanchez, and Bandoni (2001), Carnero, Hernandez, Sanchez, and Bandoni (2005) used genetic algorithms to solve multiobjective SNDP.

Several articles covered different angles of the SND problem using the tree search method (Bagajewicz & Sánchez, 1999a, 1999b, 2000a, 2000b, 2000c; Sánchez & Bagajewicz, 2000; Bagajewicz & Cabrera, 2001, 2003; Bagajewicz, Fuxman, & Uribe, 2004; Nguyen & Bagajewicz, 2008, in press) In particular, 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. The method is later generalized to solve nonlinear SNDP (Nguyen & Bagajewicz, 2008). Most recently, Kelly and Zyngier (2008) presented a MILP model based on the Schur complements.

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

* Corresponding author. Tel.: +1 405 325 5458; fax: +1 405 325 5813.
E-mail address: bagajewicz@ou.edu (M.J. Bagajewicz).

0098-1354/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compchemeng.2010.07.014

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). When there is a trade-off, the right strategy 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-optimal approach. This new approach has been recently tackled. Narasimhan and Rengaswamy (2007) discussed the value (as a performance measure) of a sensor network (from fault diagnosis perspective). Nguyen and Bagajewicz (submitted for publication) presented a value-optimal SNDP that simultaneously maximizes performance and cost of sensor network for process monitoring purpose. They presented two computational methods to solve the proposed problem: genetic algorithm (GA) and cutset-based tree search method. These two methods can run on a PC and solve medium size problems within acceptable time. However, they are not efficient methods that guarantee optimality for large scale problems. This shortcoming will be addressed in this work. More specifically, in this work we attempt to reduce the computational time of the cutset-based tree search method (Nguyen & Bagajewicz, submitted for publication) by using parallel computing.

Parallel computing has been shown to be able to remarkably reduce the solving time of computationally intensive problems. This, together with the availability of cheap/affordable computers, has led to the increasing popularity of parallel computing in scientific research community. An introduction to parallel computing can be found in Heroux, Raghavan, and Simon (2006). In the chemical industry, parallel computing has been mainly used to solve heat and mass transfer problems in fluid flow systems (the computational fluid dynamics—CFD problems). This is because the CFD problems are inherently amenable to parallel computing: they are computationally intensive and the domain as well as the data of the problems can be easily partitioned. One such use of parallel computing in the CFD problem can be found in Kerdouss et al. (2008). Some other problems of interest in chemical industry have also been solved by using parallel computing: process simulation and design (e.g. Smith & Dimenna, 2004; Hua & Wang, 2004; Ling, Biegler, & Grant Fox, 2005), dynamic process optimization (Leineweber, Schäfer, Bock, & Schlöder, 2003), supply chain optimization (Lee, Adhitya, Srinivasan, & Karimi, 2008; Shimizu, Miura, & Ikeda, 2009). The "Message Passing Interface (MPI)" tool or other type of distributed memory parallelization is commonly used (as is the case in the aforementioned articles) while the "OpenMP" tool or other type of shared memory parallelization is also occasionally used. However, to this date, parallel computing has not been used on sensor network design.

In this work, we seek to obtain the best parallelization method applicable to our problem by trying out all of the three basic approaches to parallelize computation and we do not propose any new parallelization strategy/method. The results (comparison of the three approaches) could be of interest to readers who wish to apply parallel computing to solve similar problems.

This paper is organized as follows: firstly the model that designs sensor network using the value paradigm is summarized, then the cutset-based tree search method for solving the problem is briefly described. Next, the different approaches we use to parallelize the cutset-based method are presented. Finally, two illustrated examples are provided.

2. Value-optimal sensor network design problem

Bagajewicz (2006) introduced the concept and developed formula to quantify the economic value of accuracy, which is expressed through the notion of downside expected financial loss (DEFL). DEFL is the entity that encompasses the economical effect of common performance measures of a sensor network, namely precision, gross errors delectability and gross errors resilience. More specifically, a sensor network that renders good (small) financial loss 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; this property is directly related to gross errors detectability.
iii. Small smearing effect of undetected biases on estimators of key variables; this property is directly related to gross errors resilience.

Thus, the performance of a sensor network, commonly used as constraints or required specifications in the popular cost-optimal approach, was converted into an economic term, namely the financial loss. Then, the performance and the cost of sensor network can be simultaneously optimized by maximizing value minus cost (or equivalently, minimizing financial loss plus cost).

The problem formulation is as follows (Nguyen & Bagajewicz, submitted for publication):

$$Min\{DEFL(\boldsymbol{q}) + c(\boldsymbol{q})\} \quad (1)$$

where $\boldsymbol{q}$ is binary vector indicating location of sensors, $DEFL(\boldsymbol{q})$ and $c(\boldsymbol{q})$ are financial loss and cost of the sensor network, respectively. The optimization problem (Eq. (1)) is an unconstrained integer programming problem with an objective function with many "valleys" and "hills". In turn, the financial loss $DEFL(\boldsymbol{q})$ is evaluated numerically by using Monte Carlo simulation or the approximate method presented in Nguyen Thanh, Siemanond, and Bagajewicz (2006).

Nguyen and Bagajewicz (submitted for publication) presented a genetic algorithm and a cutset-based tree search method to solve the above problem. They have shown that the cutset-based method is efficient enough for medium size problems, but not large scale problems, and that the proposed stopping criterion has "little" effect: it does not help reduce computational time.

This work aims to improve the efficiency of cutset-based tree search method. Because a stopping criterion cannot be used, we improve its efficiency by using parallel computing. The cutset-based method is briefly described next.

3. Cutset-based tree search methods

This method is the branch and bound (tree search) method using cutsets. The calculation procedure is briefly described next (Nguyen & Bagajewicz, submitted for publication):

1. Find all the cutsets of the process graph.
2. Consider only cutsets that contain at least one key variable (these are the variables of interest for monitoring), and put them to a list of cutsets.
3. Consider all key variables as separate pseudo-cutsets.
4. Sort all 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).

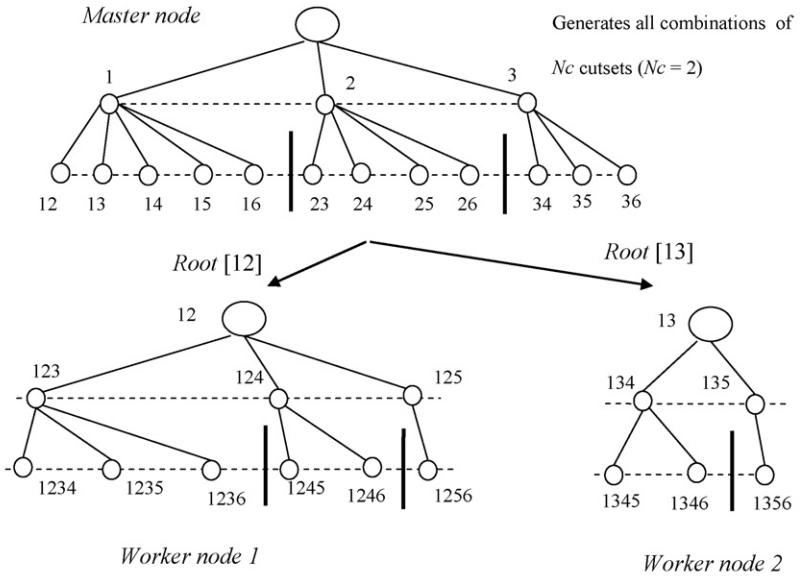
5. Start with the root node with no cutsets being added i.e. $\boldsymbol{t} = \{0, 0, 0 \ldots\}$, trivially non-optimal.
6. Use the branching criterion (see below) to develop branches of the tree (add cutsets to vector $\boldsymbol{t}$).
7. While performing the branching criteria, if any set of measurements 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.

The branching criterion is such that a 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.

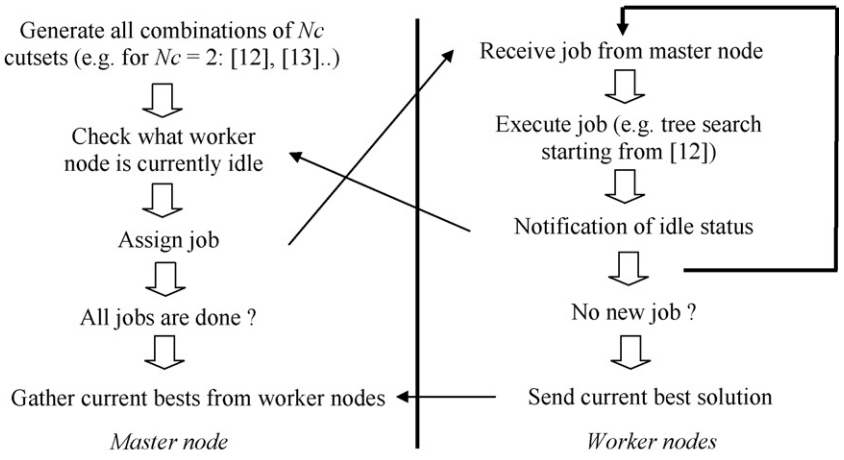
Nguyen and Bagajewicz (submitted for publication) showed that stopping criterion has "little" effect because it does not help reduce computational time. In this work, the *stopping criterion is not used*, that is, all possible combinations of cutsets are explored. Thus, the best solution identified by the tree search procedure is guaranteed to be global optimum. In turn, the advantage of using a branching criterion is that the optimal solution is usually identified earlier, which is very beneficial if the computational process has to be terminated halfway because the computational time becomes too long. The disadvantage is that using a branching criterion costs more time because a sorting of candidates need to be performed. If a branching criterion is not used, cutsets are added in the order they are generated. We investigate both cases: when branching criterion is used and NOT used.

4. Parallelized cutset-based methods

Nguyen and Bagajewicz (submitted for publication) showed that the cutset-based method was efficient for small and medium size problems, but not large scale problems, and that is why we chose to increase efficiency of the cutset-based method by using parallel computing. We discuss next the parallelization strategies and the calculation procedure. The parallel computing is implemented using the Message Passing Interface (MPI) tool. A brief introduction to parallel computing and the MPI can be found in the Appendix A.



(a) - Principle of SIMD parallelization method



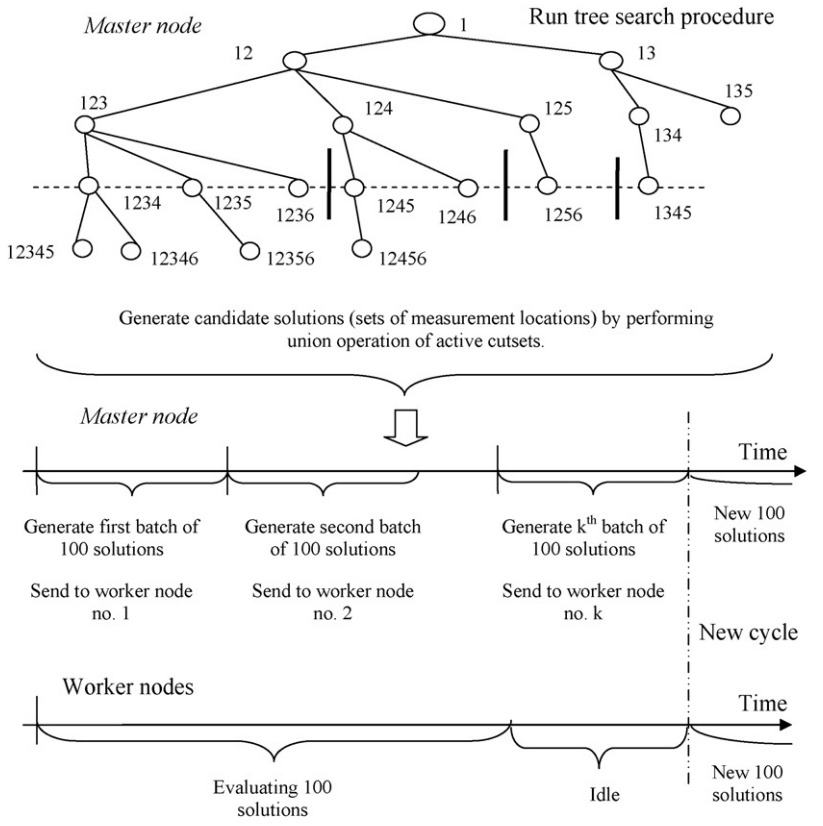
(b) – Calculation procedure of SIMD parallelization method

Fig. 1. Principle and calculation procedure of SIMD parallelization method. (a) Principle of SIMD parallelization method. (b) Calculation procedure of SIMD parallelization method.

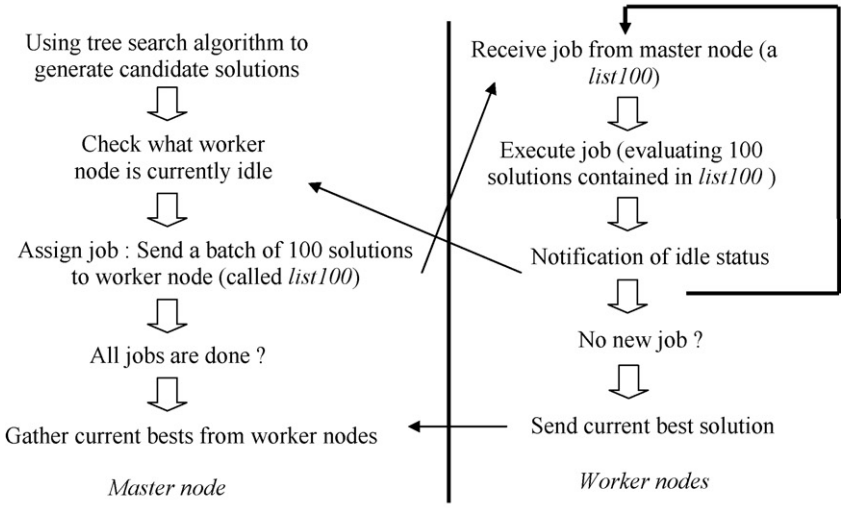
4.1. Single instruction multiple data (SIMD) parallelization method

The principle and calculation procedure of SIMD parallelization method are illustrated in Fig. 1 where we use an example of 6 cutsets [1,2,3,4,5,6].

- As illustrated all combinations of cutsets containing the root node [1 2] are evaluated in one worker node (these combinations are [123], [124], [1234], etc.). At the same time all combinations of cutsets containing another root node (e.g. [1 3]) are evaluated in another worker node and so on. To run this option the parameter *Nc* (number of active cutsets in "roots") needs to be chosen.
- Initially, all worker nodes are idle (there is no running task at that time) so the master node automatically assign jobs (combinations of *Nc* active cutsets) to worker nodes. At a later stage, the master node checks if a worker node is idle in order to assign a new job.
- Nguyen and Bagajewicz (submitted for publication) pointed out that a candidate solution (measurements location) can be a result of a union operation of different sets of cutsets (e.g. set [123] on



(a) - Principle of MISD parallelization method



(b)– Calculation procedure of MISD parallelization method

Fig. 2. Principle and calculation procedure of MISD parallelization method. (a) Principle of MISD parallelization method. (b) Calculation procedure of MISD parallelization method.

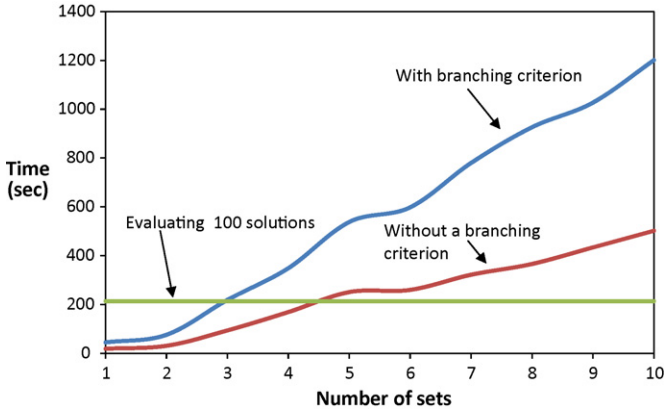


Fig. 3. Comparisons of computational times of two steps in the MISD.

worker node 1 and set [135] on worker node 2 can lead to the same candidate solution). Because the worker nodes do not communicate with each another, the SIMD parallelization method suffers from the problem of job repetition (that is, the same candidate solution is evaluated in at least two worker nodes).

4.2. Multiple instruction single data (MISD) parallelization method

The principles and calculation procedure of the MISD parallelization method are illustrated in Fig. 2.

- There are two alternatives regarding the branching criterion:
  i. The branching criterion is used.
  ii. The branching criterion is not used (the tree search illustrated in Fig. 2a does not use a branching criterion).
- In this approach there are two computation tasks that are executed simultaneously: generating all candidate solutions (by using tree search procedure with cutsets) in the 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, that is, 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.

The relative computational times of these two steps are shown in Fig. 3. In this figure, the data is taken from the case study number 2 in the Madron and Veverka's example shown below (using a 2.8 GHz Pentium CPU, 1028 RAM PC). The curve labeled "with branching criterion" shows the computational time to generate sets of 6400 candidate solutions when branching criterion is used and the curve labeled "without branching criterion" shows the corresponding computational time when no branching criterion is used. The abscissa indicates the number of sets used. For example, if the abscissa is n then the number of candidates generated are $6400^{*}n$. Thus, this computational time 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 the number of candidate solutions shown in Fig. 3. The straight line "Evaluating 100 solutions" shows the *average* computational time to evaluate 100 candidate solutions. This computational time ranges from 49 to 294 s depending on the number of sensors the solutions have (the greater the number of sensors in a sensor network, the longer the time needed to evaluate financial loss of this network).

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 Fig. 4

The best performance (the one achieving the shortest 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 the 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 Fig. 4.

Suppose that the number of worker nodes utilized in the process is k and let the time to generate $(100 \times k)$ 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 achieved because $t1$ increases as the computation process progresses (as shown in Fig. 3). In the period where $t1 < t2$, when starting a new cycle, the master node cannot 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 Fig. 3 is near to this "best" situation.

As can be inferred from the calculation procedure, the computational time of the MISD parallelization method cannot 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 and/or CPUs.

The MISD parallelization of the cutset-based tree search method with decomposition (Gala and Bagajewicz, 2006a, 2006b) has also been implemented. The parallelization of the cutset-based method with decomposition is similar to the one without decomposition. The only differences are

(1) The tree search algorithm using the decomposition technique is executed in the master node to generate candidate solutions. The decomposition technique helps reduce computational time for this task (generating candidate solutions)
(2) In step two of the calculation procedure where a worker node evaluates a batch of 100 candidate solutions:
   - Because decomposition is used, if a candidate solution (an element in the list of 100 solutions given to the worker) is obtained from the union operation of cutsets coming from the same sub-graph, then the task is to simply evaluate the objective value of that candidate solution. Otherwise, when the candidate solution contains cutsets coming from different sub-graphs, then all solutions resulted from unions and ring sums of these cutsets need to be additionally generated and evaluated as described in Gala and Bagajewicz (2006a, 2006b).
   - We introduce a new technique where the combination of unions and ring sums of cutsets can be replaced by the union of cutsets minus the connecting streams between subgraphs. For example, suppose that the candidate solution is obtained from union of cutsets A and B (from sub-graph 1), cutset C (from sub-graph 2) and cutset D (from sub-graph 3). We can verify that $(A \cup B) \cup C \oplus D$ is actually equal to the union $(A \cup B) \cup C \cup D$ minus the connecting stream of subgraph 2 (containing cutset C) and sub-graph 3 (containing cutset D). Similarly $(A \cup B) \oplus C \cup D$ is the union $(A \cup B) \cup C \cup D$

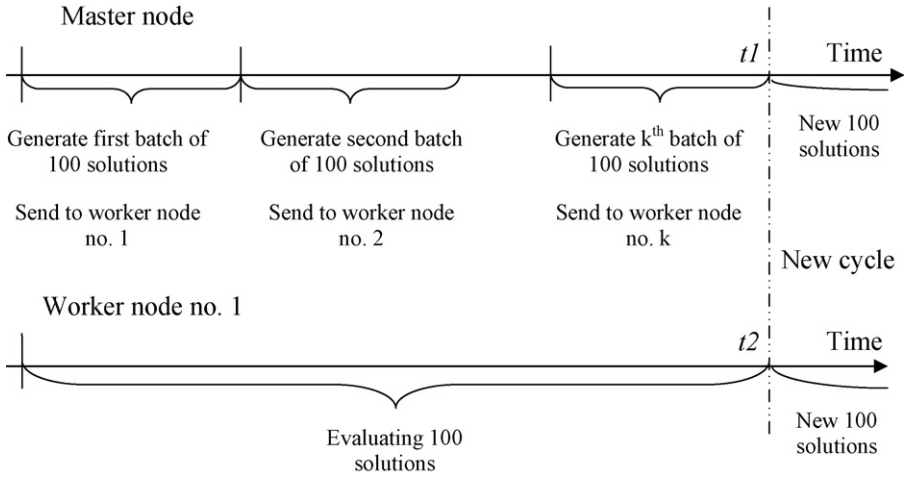


Fig. 4. Ideal scenario for MISD parallel program.

minus the connecting stream of sub-graph 1 (containing cutsets A, B) and sub-graph 2 (containing cutset C), etc. Thus all needed 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.

4.3. Multiple instruction multiple data (MIMD) parallelization method

This approach combines both the technique of partitioning the space of variables (SIMD approach) and the technique of dividing and concurrently executing computation tasks (MISD approach). The calculation procedure is essentially the same as that of the MISD approach (Fig. 2) 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 list of candidates to evaluate) from a computer node in group 1.
- The communication and assignment of jobs between master node and computer nodes in group one and group two are illustrated in Fig. 5: group one comprises of four "managing" nodes (CPU1 to CPU4), group two comprises of four sub-groups under control of the four "managing" nodes.
- Because task one is a fast job while task two is a slow job, the second group 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 Fig. 5.
- The function and the 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 the relationship between the master node and worker nodes in the MISD approach (Fig. 2); that is, the managing node generates batches of 100 candidates and sends them to worker nodes under control of this managing node (e.g. in Fig. 5 CPU1 controls sub-group 1, etc.)
- The function and the relationship between the master node and the computer nodes in group one (the managing nodes) is similar to the function and the relationship between master node and worker nodes in SIMD approach (Fig. 1): the master node generates a root node (a combination of *Nc* active cutsets like [1 2], [1 3], etc.) and send it to managing nodes.
- 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 a small number of "managing" computer nodes.
- No branching criterion and no stopping criterion is used.

All three parallelization methods use dynamic allocation of jobs: the master node and the "managing" nodes (in MIMD method) monitor the status of worker nodes and assign new jobs for idle worker nodes (except in the early stage where master node and the "managing" nodes deterministically assign jobs for worker nodes because in early stage all worker nodes are idle). We chose *dynamic load balancing* because it works well for a wide range (scale) of problems and its performance is satisfactory.

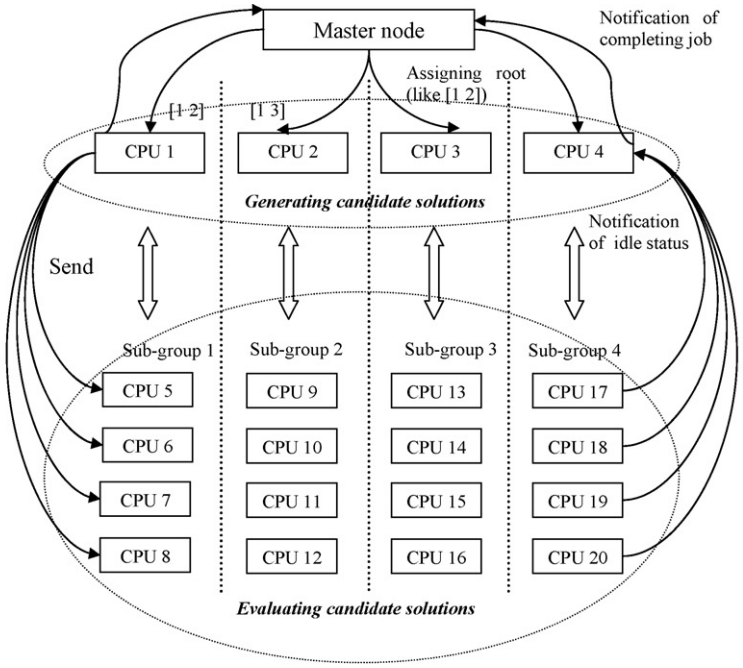
5. Example 1

The parallelized cutset-based tree search methods were implemented in Fortran. The parallelized programs were run on computer network ("super computer") of 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 8768 GB RAM. More detail on configuration of OSCER super computer can be found in the website http://www.oscer.ou.edu.

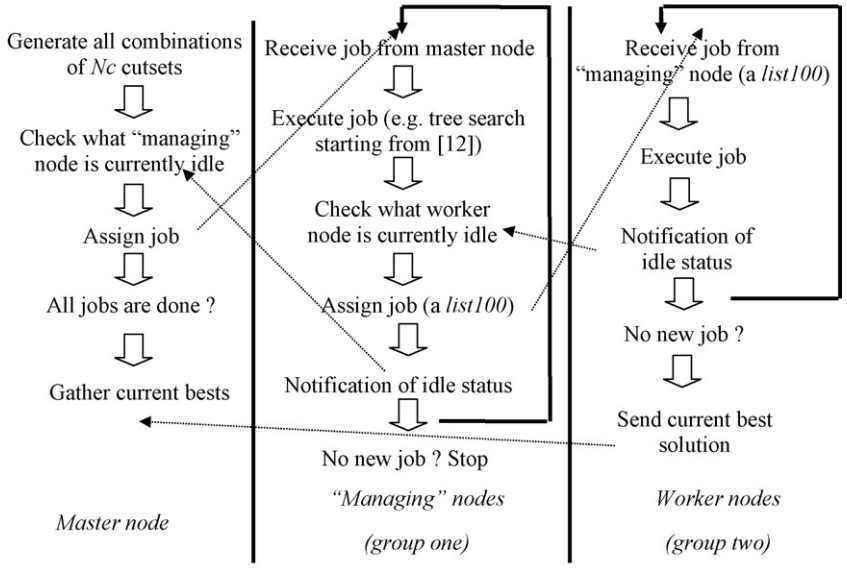
The Madron and Veverka's example presented in Nguyen and Bagajewicz (2009) is considered. The process flowsheet is given in Fig. 6 and the process data is given in Table 1.

The following data was used.

- 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 2.



(a) – Principle of MIMD parallelization method



(b) – calculation procedure of MIMD parallelization method

Fig. 5. Principle and calculation procedure of MIMD parallelization method. (a) Principle of MIMD parallelization method. (b) Calculation procedure of MIMD parallelization method.

Ten design case studies, introduced in Nguyen and Bagajewicz (submitted for publication), are considered. Their solutions obtained using the cutset-based methods are shown in Table 2. The last four columns of Table 2 show the number of sensors, the measurements locations, the total cost of sensors and the financial loss.

The performance of the three parallelization methods is shown next. The reported computational times shown in the following tables are the average wallclock time (elapsed time) determined from 2 measurements (while more measurements may be needed, we found that the deviations between measured values are small so we decided to perform only two runs to obtain two measurements for each reported computational time).

5.1. Performance of the serial program

The performance of the serial cutset-based method running on the OSCER super computer is shown in columns 3 and 4 of Table 3. No stopping criterion is used. The computational time is obtained by running the serial cutset-based method on OSCER "super computer" without any kind of parallelization (that is, using only 1 CPU) and considering both cases: with and without branching criterion. For comparison, performance of the same serial cutset-based method (with branching criterion) running on a typical PC (2.8 GHz Pentium CPU, 1028 MB RAM) is also shown in column 5 of Table 3.

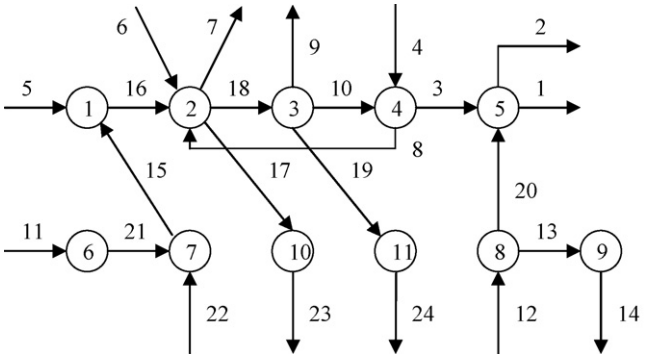


Fig. 6. Flowsheet of Madron problem.

Thus, running the program on the OSCER "super computer" saves computational time by the factor of 12 (when compared against running it on a PC).

We show next the results for the three MPI-enabled parallelization methods. The computational times are reported for all cases and occasionally the following two terms are also reported.

The speedup rate $S(P)$ is defined by:

$$S(P) = \frac{T(1)}{T(P)} \tag{2}$$

where $T(P)$ is the average wallclock time (elapsed time) to solve the problem using P computer nodes/CPU. The case $P = 1$ corresponds to the case where no parallelization is used (shown in columns 3 and 4 of Table 3). In turn, the efficiency $\varepsilon(P)$ is defined by:

$$\varepsilon(P) = \frac{S(P)}{P} \tag{3}$$

Table 1
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

Table 2
Results for Madron problem.

Case study	Key variables	K_s value	Number of sensors	Measured variables	Sensors cost	Financial loss
1.1	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
1.2	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
1.3	2, 6, 24	$Ks_2 = 25$ $Ks_6 = 20$ $Ks_{24} = 20$	4	2, 6, 19, 24	57	400.1
1.4	4, 9, 23	$Ks_4 = 25$ $Ks_9 = 20$ $Ks_{23} = 20$	4	4, 9, 17, 23	47	283
1.5	4, 5, 24	$Ks_4 = 25$ $Ks_5 = 25$ $Ks_{24} = 45$	4	4, 5, 19, 24	67	527.1
1.6	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
1.7	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
1.8	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
1.9	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
1.10	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

Table 3
Performance of serial cutset-based method.

Case study	Number of candidate solutions explored	Computational time		
		OSCER—with branching criterion	OSCER—No branching criterion	PC—with branching criterion
1.1	46,042	46 min 31 s	43 min 49 s	9 h 4 min
1.2	64,781	60 min 40 s	55 min 40 s	11 h 44 min
1.3	38,070	22 min 5 s	20 min 30 s	4 h 20 min
1.4	28,178	15 min	14 min 6 s	2 h 45 min
1.5	34,134	18 min 49 s	17 min 48 s	3 h 31 min
1.6	39,552	40 min 23 s	38 min 7 s	7 h 57 min
1.7	39,552	40 min 25 s	38 min 6 s	7 h 56 min
1.8	38,365	39 min 24 s	37 min 51 s	8 h 2 min
1.9	38,365	39 min 27 s	37 min 51 s	8 h 3 min
1.10	38,365	39 min 26 s	37 min 51 s	8 h 1 min

For concise presentation of the paper, we present only the average values of these two common performance measures of parallel computing (each term is calculated as the average value of the ten values corresponding to the ten design cases).

5.2. Performance of the SIMD parallelization method

The performance of the parallelized cutset-based method-SIMD approach is shown in Table 4. In this method the important parameter is *Nc*, the number of cutsets of a node that is transferred to a worker. If *Nc* = 1, all combinations of *Nc* cutsets are [1], [2], [3], . . ., etc.; all candidate 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 Fig. 1). If there are n_t cutsets in total, then the number of possible combinations of *Nc* cutsets is given by

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

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* = 2 are shown in Table 4. The results are obtained using 64 computer nodes (one master CPUs and 63 worker nodes).

Column 3 of Table 4 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 3) is also shown in the last column. In turn, column 4 of Table 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 (the one that explores solutions starting from root [1] (*Nc* = 1) or [1 2] (*Nc* = 2)).

The results shown in Table 4 show that

- The SIMD's job repetition problem is illustrated by the fact that the total number of candidate solutions explored (column 3) is larger than the corresponding value when SIMD parallelization is NOT used (column 6).
- The SIMD approach offers only a small improvement in computational time. When *Nc* = 2 there is roughly 46% improvement in computational time. The (average) speedup rate $S(64)$ is 2.02 and the (average) efficiency $\varepsilon(64)$ is 0.032.
- The computational time for *Nc* = 1, not shown here, is only about 5% shorter.
- The poor performance of the SIMD parallelization is also due to the poor load balancing of this approach in this specific problem. 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 table), hence the load balancing is bad. Load balancing can be improved, but this is left for future work.

5.3. Performance of the MISD parallelization method

The results are obtained using 64 computer nodes (one master and 63 worker nodes). Table 5 shows results when branching is activated and no decomposition technique is used. We conclude that:

- The computational time of the MISD approach is 12 times smaller than that of the SIMD approach.
- The (average) speedup rate $S(64)$ is 21.2 and the efficiency $\varepsilon(64)$ is 0.33.
- The load balancing of the MISD approach is much better than the SIMD approach as it is revealed by the small difference between the maximum and minimum value of solutions evaluated in a computer node.

Table 4
Performance of SIMD parallelization method (*Nc* = 2).

Case study	Computational time (s)	Number of candidate solutions explored			
		Total	Maximum number of candidates explored in one node	Minimum number of candidates explored in one node	Total—serial program
1.1	1609	152,427	12,794	278	46,042
1.2	1513	253,796	17,976	3102	64,781
1.3	747	196,069	10,218	2588	38,070
1.4	767	159,078	7,339	2186	28,178
1.5	641	173,509	9,109	2287	34,134
1.6	768	170,223	10,809	1432	39,552
1.7	768	170,223	10,809	1397	39,552
1.8	1391	148,292	10,321	245	38,365
1.9	1392	148,292	10,321	245	38,365
1.10	1392	148,292	10,321	245	38,365

Table 5
Performance of MISD parallelization with branching criterion (No decomposition).

Case study	Computational time (s)	Number of candidate solutions evaluated	
		Minimum number of candidates explored in one node	Maximum number of candidates explored in one node
1.1	126	600	800
1.2	192	800	1300
1.3	90	400	700
1.4	53	300	1200
1.5	74	400	1000
1.6	89	400	900
1.7	90	400	900
1.8	101	500	700
1.9	102	500	700
1.10	101	500	700

Table 6
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	$\{S_3\}$, $\{S_{15}\}$, $\{S_{16}\}$, $\{S_{17}\}$, $\{S_{19}\}$, $\{S_{20}\}$

Table 6 describes three different decompositions that were used. Table 7 depicts the results obtained using the branching criterion and these three different decompositions.

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.

In Table 7, columns 4 and 7 show 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 the decomposition technique is used, because of the extra step of exploring all possibilities of removing connecting streams out of that candidate solution (described above), the number of candidate solutions evaluated will be larger than the solutions sent to a worker node from master node. For example, if an element in the list (a candidate solution) is [1 2 3 4 5] and [4 5] are the 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 (the summation) of solutions that have been evaluated in all worker nodes (shown in columns 3 and 6) must be larger than the number of solutions generated in master node (columns 4 and 7).

The large difference between the total number of solutions evaluated when the MISD parallelization method with decomposition is used (columns 3 and 6 of Table 7) and the total number of candidate solutions when it is not used (column 2 of Table 3) is due to the fact that there is job repetition 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 7.

Using decomposition has two opposite effects: the good effect is that it reduces the time to generate candidate solutions in the master node, the bad effect is that it increases the time to evaluate solutions in worker nodes because of the problem of job repetition. Table 7 shows that when the branching criterion is used, a small number of decompositions is beneficial: option 1, single decomposition, reduces the computational time by 10% (when compared with the base case where no decomposition is used, Table 5). However, a large number of decompositions (option 3) has an adverse effect: it increases computational time because of the problem of job repetition.

The results when the branching criterion is not used are shown in Table 8. The second column of this table shows the 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 6.

Table 7
Performance of MISD approach with decomposition and branching criterion.

Case study	Single decomposition			Multiple decomposition		
	Comput. time (s)	Number of solutions evaluated	Number of solutions generated	Comput. time (s)	Number of solutions evaluated	Number of solutions generated
1.1	99	87,451	22,337	833	713,861	19,272
1.2	111	140,528	36,142	821	598,321	15,889
1.3	66	72,112	18,243	813	580,569	15,086
1.4	62	50,513	12,739	737	484,968	13,836
1.5	67	63,455	16,182	737	430,397	12,447
1.6	89	81,132	20,997	817	447,922	12,517
1.7	89	81,132	20,997	815	447,922	12,517
1.8	94	72,508	18,325	790	577,835	14,929
1.9	95	72,508	18,325	790	577,835	14,929
1.10	94	72,508	18,325	791	577,835	14,929

Table 8
Performance of MISD approach with decomposition, no branching criterion.

Case study	Computational time (s)			
	Number of decomposition			
	None	Single	Double	Multiple
1.1	63	97	78	824
1.2	97	110	74	832
1.3	47	66	67	830
1.4	22	62	61	738
1.5	37	60	65	858
1.6	47	86	66	858
1.7	47	86	65	858
1.8	57	93	67	805
1.9	57	93	68	804
1.10	57	93	68	804

Table 9
Performance of MISD method vs. number of CPUs, no branching, no decomposition.

	$N_{CPU}=32$	$N_{CPU}=64$	$N_{CPU}=96$
Average speedup rate	25.5	38.65	44.94
Average efficiency	0.77	0.604	0.47
Case study	Computational time (s)		
1.1	100	63	52
1.2	139	97	84
1.3	60	47	39
1.4	36	22	22
1.5	50	37	30
1.6	73	47	41
1.7	74	47	41
1.8	88	57	48
1.9	89	57	48
1.10	88	57	48

- Column 2 of Table 8 reveals that the computational time improves by 50%, which is significant, when compared with the base case when the branching criterion is used.
- 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 a decomposition technique is used.

The 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 9. The two rows 2 and 3 of Table 9 show the average speedup rate and efficiency.

Table 9 shows that if more computer nodes are used (i.e. more "workers" to share the tasks), the computational time is reduced, as expected. However, it is well known that the dependence of performance of MISD method on the number of computer nodes is not linear: the performance improvement becomes smaller as more computer nodes are used: when the number of computer nodes increases from 32 to 96, the efficiency decreases from 0.77 to 0.47 (as usually the case). The efficiency for this case is also much better than the case where branching criterion is used: the efficiencies $\varepsilon(64)$ for the two cases are 0.604 (no branching) and 0.33 (with branching).

Thus, the MISD parallelization is a great improvement over the serial cutset-based method. The best performance (column 4 of Table 9) of the MISD parallelization is achieved when no branching criterion and no decomposition is used, and when one uses as many computer nodes as possible.

5.4. Performance of the MIMD parallelization method

Results are shown in Table 10. They were obtained using the following set of parameters:

i. *Nc* (as described in Fig. 5) = 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 (for the case 64 CPUs are used).
iii. No branching criterion.

Columns 2, 3, 6, 7 ("Max-node", "Min-node") of Table 10 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).

- This approach suffers from the problem of job repetition, similarly to the SIMD approach because both divide the problem data.
- Because this approach (MIMD) divides both the problem data and problem instructions 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 the MIMD

Table 10
Performance of MIMD parallelization method.

Case study	64 CPUs				96 CPUs			
	Candidate solutions explored			Time (s)	Candidate solutions explored			Time (s)
	Maximum number of candidates explored in one node	Minimum number of candidates explored in one node	Total		Maximum number of candidates explored in one node	Minimum number of candidates explored in one node	Total	
1.1	21,037	9,958	71,213	111	19,948	9,958	69,761	74
1.2	28,360	9,731	91,529	107	28,355	9,731	91,541	73
1.3	21,812	6,634	55,588	57	20,196	6,236	56,958	40
1.4	12,624	7,336	52,055	57	12,580	7,336	51,962	40
1.5	19,695	4,754	47,429	48	18,392	4,754	51,570	30
1.6	19,721	12,611	69,970	59	18,493	13,202	71,782	47
1.7	19,808	12,677	69,710	59	18,412	13,191	71,655	48
1.8	19,233	5,382	61,278	99	19,272	5,382	61,262	68
1.9	19,151	5,382	61,195	101	19,186	5,382	61,227	69
1.10	19,279	5,382	61,339	101	19,009	5,382	60,761	69

Table 11
Performance of MIMD parallelization at varied number of managing nodes.

	$M=2$	$M=4$	$M=8$
Average speedup rate	51.8	36.4	19.55
Average efficiency	0.54	0.38	0.203

Case study	Total candidate solutions explored			Computational time (s)		
	$M=2$	$M=4$	$M=8$	$M=2$	$M=4$	$M=8$
1.1	65,373	69,761	102,555	54	74	148
1.2	81,505	91,541	137,974	62	73	142
1.3	50,594	56,958	89,230	25	40	73
1.4	41,652	51,962	79,258	25	40	74
1.5	48,975	51,570	81,744	18	30	64
1.6	58,920	71,782	83,669	46	47	79
1.7	59,410	71,655	83,916	47	48	78
1.8	51,973	61,262	89,713	39	68	133
1.9	51,864	61,227	89,873	39	69	134
1.10	51,973	60,761	89,613	39	69	134

parallelization is larger than that in the MISD approach. Moreover, the available resource (CPUS) for this purpose in the 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). This explains why the MIMD approach spends more time for this goal.

iii. A comparison of performance of these two approaches cannot be made unless a large number of tests have been conducted. For this specific case of the Madron and Veverka's 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, the computational time decreases.
- Because of the dynamic nature of the process (real time monitoring of worker nodes' status and assigning jobs), the number of candidate solutions explored is usually different at different runs or if the computer system on which the process runs is different (e.g. changing the number of CPUs).
- The average efficiencies are $\varepsilon(64) = 0.405$, $\varepsilon(96) = 0.38$.
- For this specific configuration (no branching, no decomposition, using four "managing" nodes in MIMD), the MIMD parallelization is worse than the MISD approach: the efficiencies $\varepsilon(64)$ for the two cases are 0.604 (MISD) and 0.405 (MIMD); the efficiencies $\varepsilon(96) = 0.47$ (MISD) and 0.38 (MIMD).

The effect of varying number of computer nodes ("managing" nodes) in group one (M) is shown in Table 11 using a total of 96 CPUs. Three cases are considered ($M=2$, $M=4$, $M=8$), which are shown in Table 11. 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).

It can be seen from Table 11 that the computational time increases (efficiency decreases) 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, increases when M increases).
(ii) There are fewer worker nodes to evaluate the candidate solutions (for the same total number of CPUs).

For this specific configuration (no branching, no decomposition, using two "managing" nodes in MIMD), the MIMD parallelization is better than the MISD approach: the efficiencies $\varepsilon(96)$ for the two cases are 0.54 (MIMD) and 0.47 (MISD).

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

The results when a decomposition technique is used (with the modifications outlined above) are shown in Table 12; only one decomposition option (double decomposition, shown in row 3 of Table 6) is considered (we used 96 CPUs, among which two CPUs 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 12.

As can be seen from Table 12, using a 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 problem, when decomposition is used, the number of cutsets in the problem reduces about 3.3–6 times (from about 100 cutsets to 30 cutsets in decomposition option 1 and 16 cutset in decomposition option 3). Such a small decrease in the number of cutsets does not help reduce 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 best performance (the shortest computational time) is given by the 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, using the MISD (multiple instruction single data) approach as base case, there are two opposite effects when 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 can be said about using decomposition (when compared against the counter-

Table 12
Performance of MIMD parallelization with decomposition technique.

Case study	Total candidate solutions explored		Computational time (s)	
	No decomposition	Double decomposition	No decomposition	Double decomposition
1.1	65,373	103,295	54	80
1.2	81,505	106,344	62	75
1.3	50,594	73,579	25	67
1.4	41,652	59,793	25	60
1.5	48,975	60,069	18	64
1.6	58,920	73,169	46	66
1.7	59,410	75,557	47	67
1.8	51,973	70,778	39	67
1.9	51,864	68,433	39	67
1.10	51,973	68,433	39	67

part where decomposition is NOT used). The trade-off (final result) of these two opposite effects depends on the specific problem under consideration.

We now give an ad hoc guideline regarding what option is recommended. 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 ($Nw \times 100$) candidate solutions and evaluating a batch of 100 candidates sent from the master node to worker nodes. Fig. 7 shows the two extreme cases that can occur for the base case where the MISD parallelization method and no decomposition are used.

- The points 1, 2, 3, ..., N in the x-axis indicate the total number of candidate solutions generated in the process: at point N the total number of candidate solutions generated is $N \times (Nw \times 100)$. As discussed above, the time to generate ($Nw \times 100$) candidate solutions increases progressively along with the computational process.
- If no decomposition is used, the task 2 (evaluating a batch of 100 candidates) involves computing the objective function for each of the 100 candidate solutions. If decomposition is used, the actual number of candidate solutions that need to be evaluated is more than 100 (because the step now has the additional task of generating additional candidates as described above) and the time for this task increases (compared against the base case where decomposition is NOT used) as illustrated in Fig. 8.

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

- Evaluation step is time consuming (Case A in Fig. 7): to avoid job repetition, the best option is the MISD approach without decomposition.
- Evaluation step (task 2) is a lot faster than the task of generating candidate solutions (Case B): The best option is to use the MIMD parallelization method and the decomposition technique (use either of them or use both). The effect of using these two options is illustrated in Fig. 8.

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
- The decomposition technique helps reduce the total number of candidate solutions generated (as illustrated in Fig. 8: the "new" value of total number of candidate solutions generated is $M \times (Nw \times 100) <$ the old value $= N \times (Nw \times 100)$).

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.

6. Example 2—CDU example

The CDU example introduced in Gala and Bagajewicz (2006a, 2006b) is considered next. The process flowshheet is shown in Fig. 9 and the problem data is shown in Table 13.

Financial loss is calculated using the following parameters:

- 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.

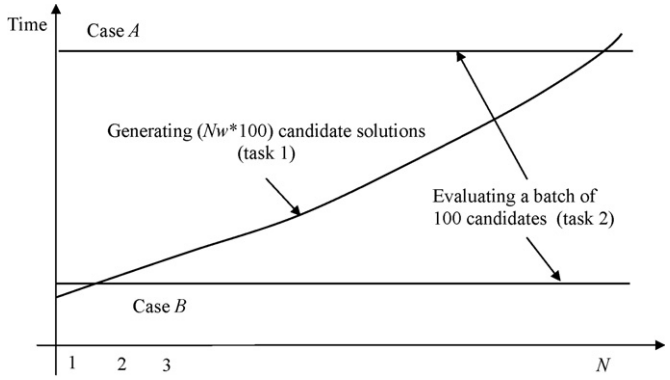


Fig. 7. Analysis of performance of the MISD parallelization method.

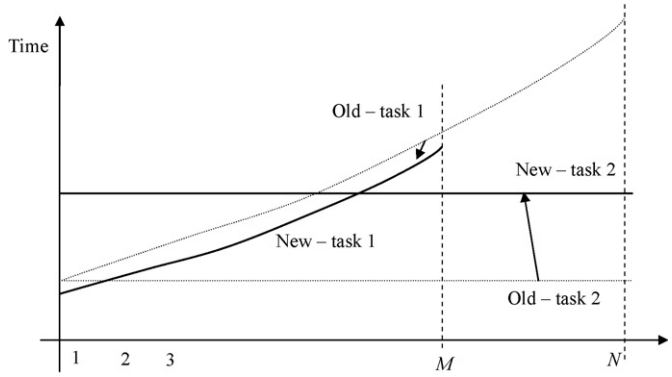


Fig. 8. Performance of parallelization method—with MIMD and/or decomposition.

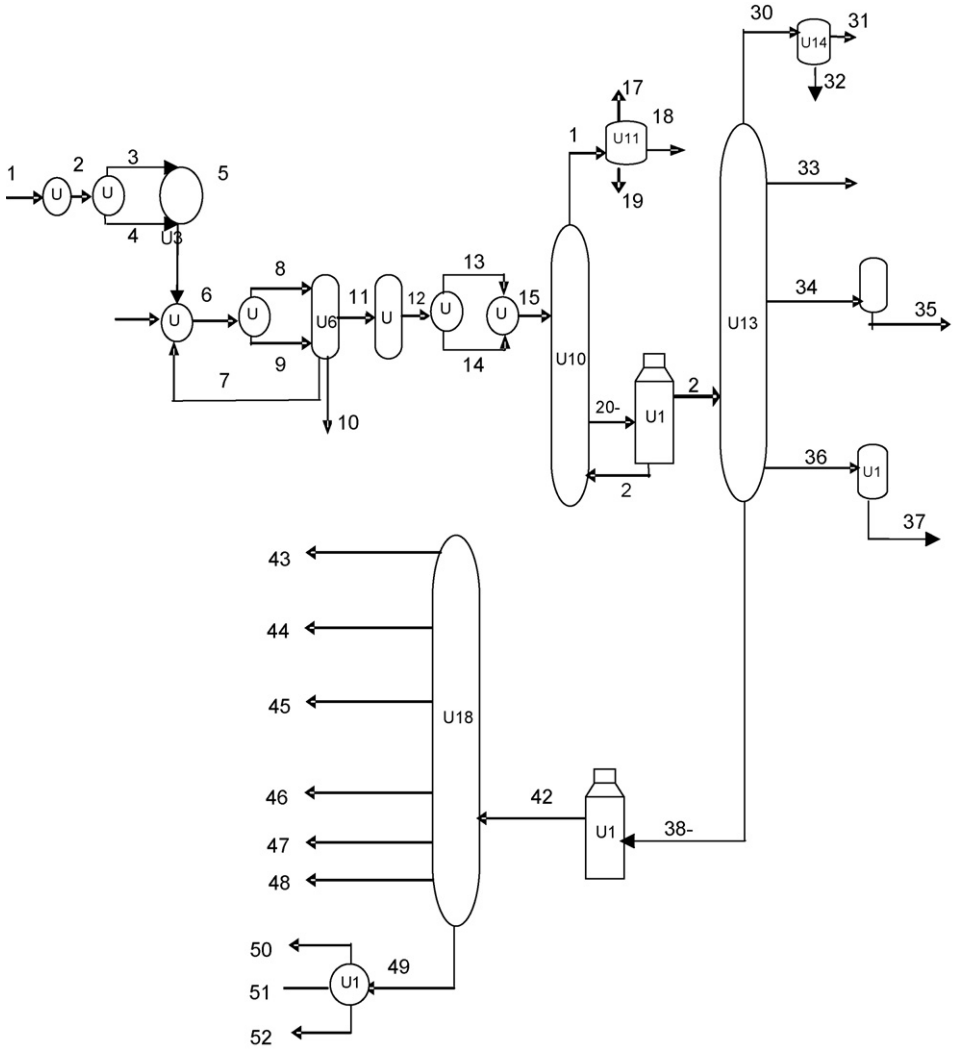


Fig. 9. Process flowsheet—CDU example.

Table 13
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

The design case study and the obtained solution are described in Table 14.

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} and S_{29}.

For this problem, if decomposition is not used, the number of cutsets (containing at least one key variable) is 973 while if decomposition is used, the number of cutsets is reduced to 158 (single decomposition, connecting stream = S_{15}) and 69 (double

Table 14
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 h 5 min

decomposition, connecting streams = S_{11} and S_{29}). Thus, simple decomposition strategies like those described greatly reduces the number of cutsets (the number of cutsets reduces 6.2 times for single decomposition and 14.1 times for double decomposition).

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

7. Conclusions

In this paper, the efficiency of the cutset-based tree search method (Nguyen & Bagajewicz, submitted for publication) is improved by using parallel computing. Three parallelization methods are investigated. The methods that divide program instruction (MISD and MIMD) are found to be much better than the method that divides program data only (SIMD). The parallelized cutset-based method (using MISD and MIMD parallelization methods) is found to be a very efficient method to solve the value-optimal sensor network design problem. It guarantees optimality and it is able to solve large scale problems within an acceptable time.

Appendix A.

A.1. Overview of parallel computing

The characteristic and benefits of parallel computing compared against serial computing are shown in Table A1.

There are many so-called library routines/interface specifications that make it easier for a programmer to transfer from serial program to parallel program; the most well-known are the openMP (a library of compiler directives and subroutines) and the Message Passing Interface (MPI). From a programmer perspective, the 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.

A.2. Message passing interface

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. 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). The 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 (CPU) called master node (or server node), the rest are called worker nodes.

Different approaches to do parallel computing are shown next.

Table A1
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

A.3. Parallelization methods

(i) Single instruction multiple data (SIMD): 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). It is appropriate to use this approach when *problem data* can be divided into different parts, each part can be processed independently.

(ii) Multiple instruction single data (MISD): in case the computational tasks can be executed independently (execution of a task does not depend on output from another task) or if computational tasks can be decoupled, then the program can be parallelized by executing the tasks concurrently

(iii) Multiple instruction multiple data (MIMD): this approach combines both the SIMD approach and MISD approach. It is applicable if both program instructions and program data can be divided.

A.3.1. Factors affecting performance of parallelized programs

- The parallelization method achieves the best performance (short computational time without using too much resources) 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 parallelization method achieves the best performance if there is no repetition of jobs assigned/executed in worker nodes.

References

Bagajewicz, M. (1997). Design and retrofit of sensors networks in process plants. *AIChe Journal*, *43*(9), 2300–2306.
Bagajewicz, M. (2006). Value of accuracy in linear systems. *AIChE Journal*, *52*(2), 638–650.
Bagajewicz, M., & Sánchez, M. (1999a). Duality of sensor network design models for parameter estimation. *AIChE Journal*, *45*(3), 661–664.
Bagajewicz, M., & Sánchez, M. (1999b). Design and upgrade of non-redundant and redundant linear sensor networks. *AIChE Journal*, *45*(9), 1927–1939.
Bagajewicz, M., & Sánchez, M. (2000a). Reallocation and upgrade of instrumentation in process plants. *Computers and Chemical Engineering*, *24*, 1945–1959.
Bagajewicz, M., & Sánchez, M. (2000b). Cost-optimal design of reliable sensor networks. *Computers and Chemical Engineering*, *23*(11/12), 1757–1762.
Bagajewicz, M., & Sánchez, M. (2000c). Reallocation and upgrade of instrumentation in process plants. *Computers and Chemical Engineering*, *24*(8), 1961–1980.
Bagajewicz, M., & Cabrera, E. (2001). A new MILP formulation for instrumentation network design and upgrade. *AIChe Journal*, *48*(10), 2271–2282.
Bagajewicz, M., & Cabrera, E. (2003). Pareto optimal solutions visualization techniques for multiobjective design and upgrade of instrumentation networks. *Industrial and Engineering Chemistry Research*, *42*, 5195–5203.
Bagajewicz, M., Fuxman, A., & Uribe, A. (2004). Instrumentation network design and upgrade for process monitoring and fault detection. *AIChE Journal*, *50*(8), 1870–1880.

Bhushan, M., Narasimhan, S., & Rengaswamy, R. (2008). Robust sensor network design for fault diagnosis. *Computers & Chemical Engineering*, *32*, 1067–1084.
Carnero, M., Hernandez, J., Sanchez, M., & Bandoni, A. (2001). An evolutionary approach for the design of nonredundant sensor networks. *Industrial and Engineering Chemistry Research*, *40*, 5578–5584.
Carnero, M., Hernandez, J., Sanchez, M., & Bandoni, A. (2005). On the solution of the instrumentation selection problem. *Industrial and Engineering Chemistry Research*, *44*, 358–367.
Chen, J. Y., & Chang, C. T. (2008). Development of an optimal sensor placement procedure based on fault evolution sequences. *Industrial and Engineering Chemistry Research*, *47*, 7335–7346.
Chmielewski, D., Palmer, T., & Manousiouthakis, V. (2002). On the theory of optimal sensor placement. *AIChe Journal*, *48*(5), 1001–1012.
Gala, M, & Bagajewicz, M. J. (2006). Rigorous methodology for the design and upgrade of sensor networks using cutsets. *Industrial and Engineering Chemistry Research*, *45*(20), 6687–6697.
Gala, M., & Bagajewicz, M. J. (2006). Efficient procedure for the design and upgrade of sensor networks using cutsets and rigorous decomposition. *Industrial and Engineering Chemistry Research*, *45*(20), 6679–6686.
Heroux, M. A., Raghavan, P., & Simon, H. D. (2006). *Parallel processing for scientific computing*. Philadelphia, USA: Society for Industrial and Applied Mathematics.
Meng, Hua, & Wang, Chao-Yang. (2004). Large-scale simulation of polymer electrolyte fuel cells by parallel computing. *Chemical Engineering Science*, *59*(16), 3331–3343.
Kelly, J. D., & Zyngier, D. (2008). A new and improved MILP formulation to optimize observability redundancy and precision for sensor network problems. *AIChe Journal*, *54*(5), 1282–1291.
Kerdouss, F., Bannari, A., Proulx, P., Bannari, R., Skrga, M., & Labrecque, Y. (2008). Two-phase mass transfer coefficient prediction in stirred vessel with a CFD model. *Computers & Chemical Engineering*, *32*, 1943–1955.
Kotecha, P. R., Bhushan, M., & Gudi, R. D. (2007). Constraint programming based robust sensor network design. *Industrial and Engineering Chemistry Research*, *46*, 5985–5999.
Kotecha, P. R., Bhushan, M., & Gudi, R. D. (2008). Design of robust, reliable sensor networks using constraint programming. *Computers & Chemical Engineering*, *32*, 2030–2049.
Kretsovalis, A., & Mah, R. S. H. (1987). Effect of redundancy on estimation accuracy in process data reconciliation. *Chemical Engineering Science*, *42*, 2115.
Lee, Y. K., Adhitya, A., Srinivasan, R., & Karimi, I. A. (2008). Decision support for integrated refinery supply chains: Part 2 Design and operation. *Computers & Chemical Engineering*, *32*(11), 2787–2800.
Leineweber, D. B., Schäfer, A., Bock, H. G., & Schlöder, J. P. (2003). An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization: Part II: Software aspects and applications. *Computers & Chemical Engineering*, *27*(2), 167–174.
Ling, J., Biegler, L. T., & Grant Fox, V. (2005). Design and optimization of pressure swing adsorption systems with parallel implementation. *Computers & Chemical Engineering*, *29*(2), 393–399.
Luong, M., Maquin, D., Huynh, C., & Ragot, J. (1994). Observability, redundancy reliability and integrated design of measurement systems. In *Proceeding of 2nd IFAC symposium on intelligent components and instrument control applications* Budapest, Hungary,
Madron, F., & Veverka, V. (1992). Optimal selection of measuring points in complex plants by linear models. *AIChe Journal*, *38*(2), 227.
Meyer, M., Le Lann, J., Koehret, B., & Enjalbert, M. (1994). Optimal selection of sensor location on a complex plant using a graph oriented approach. *Computers & Chemical Engineering*, *18*(Suppl.), S535–S540.
Narasimhan, S., & Rengaswamy, R. (2007). Quantification of performance of sensor networks for fault diagnosis. *AIChe Journal*, *53*(4), 902–917.
Nguyen Thanh, D. Q., Siemanond, Kitipat, & Bagajewicz, M. (2006). Downside financial loss of sensor networks in the presence of gross errors. *AIChE Journal*, *52*(2), 3825–3841.
Nguyen, D. Q., & Bagajewicz, M. (2008). Design of nonlinear sensor networks for process plants. *Industrial and Engineering Chemistry Research*, *47*(15), 5529–5542.
Nguyen, D. Q., & Bagajewicz, M. (2009). On the impact of sensor maintenance policies on stochastic-based accuracy. *Computers & Chemical Engineering*, *33*(9), 1491–1498.
Nguyen, D. Q., & Bagajewicz, M. (submitted for publication). New sensor network design and retrofit method based on value of information. *AIChe Journal*.
Nguyen, D. Q., & Bagajewicz, M. (in press). New efficient breadth-first/level traversal tree search method for the design and upgrade of sensor networks. *AIChe Journal*.
Pacheco, P. S. (1997). *Parallel programming with MPI*. San Francisco, USA: Morgan Kaufmann Publishers.
Peng, J. K., & Chmielewski, D. J. (2006). Covariance based hardware selection part 2: Simultaneous sensor and actuator selection. *IEEE Transactions on Control Systems Technology*, *14*(2), 362–368.
Sánchez, M., & Bagajewicz, M. (2000). On the impact of corrective maintenance in the design of sensor networks. *Industrial and Engineering Chemistry Research*, *39*(4), 977–981.
Shimizu, Y., Miura, T., & Ikeda, M. (2009). A parallel computing scheme for large-scale logistics network optimization enhanced by discrete hybrid PSO. *Computer Aided Chemical Engineering*, *27*, 2031–2036.
Smith, F. G., III, & Dimenna, R. A. (2004). Simulation of a batch chemical process using parallel computing with PVM and Speedup. *Computer Aided Chemical Engineering*, *28*(9), 1649–1659.