

***NEW EFFICIENT LEVEL TRAVERSAL TREE SEARCH METHODOLOGY FOR THE
DESIGN AND UPGRADE OF SENSOR NETWORKS***

DuyQuang Nguyen and Miguel J. Bagajewicz^()*

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

^(*) Corresponding Author. TE: (405) 325-5458, FAX: (405) 325-5813, e-mail:
bagajewicz@ou.edu

Keywords: Instrumentation Design, Instrumentation upgrade,

ABSTRACT

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 paper we exploit certain cost property of the different nodes in the tree and present a breath first or level traversal strategy. We show that this strategy allow us to cut considerably the computational time for problems that the branch and prune strategy was inefficient.

1. Introduction

The importance of data treatment techniques like data reconciliation and gross error detection in the process industry has long been recognized. Those data treatment techniques have been the subject of research over the past four decades and many successful industrial applications of data reconciliation have been reported (Bagajewicz, 1997, 2000). Considering the fact that, in reality, only a fraction of process variables are measured by sensors, the observability and /or precision of the estimators depends on the location and precision of the sensors as well as the presence of biases. In fact, software redundancy is needed to perform a good data reconciliation and bias detection. However, the existence of undetected biases also creates some problems. To quantify these effects, a new term has been coined: “Software accuracy” (Bagajewicz, 2005), later extended to “Stochastic software accuracy” (Nguyen and Bagajewicz, 2008).

To maximize the accuracy, one needs to solve the instrumentation location problem, which consist of locating instruments such that the instrumentation cost is minimized and the desired accuracy in key variables (usually those associated to production accounting, control and safety), is met.

Bagajewicz and Sanchez (1999a) developed a method to achieve observability and Bagajewicz and Sanchez (2000a) developed a method for reliable sensor networks. However, it was the cost-optimal formulation (Bagajewicz, 1997), that prevailed. Many constraints were added later, reallocation was considered (Bagajewicz and Sanchez (2000b), and the impact of maintenance was also studied (Bagajewicz and Sanchez (2000c). The duality with the cost constrained maximum precision (Bagajewicz and Sanchez, 1999b).

This instrumentation network design and upgrade problem has not been amenable to be solved as a regular MINLP model for a variety of reasons. Indeed, although Bagajewicz and Cabrera (2001) proposed an MILP model for precision constrained problems their model showed scaling problems. In turn, Chmielewski et al (2002) used LMI and were able to propose convex MINLP formulations, also for precision constrained problems. Finally, Kelly and Zyngier (2008) proposed method based on Schur complements for the precision constrained problem as well. Whereas extensions of these methods to consider filtering of biases cannot be excluded, such

software accuracy-based models have not been developed. Moreover, it is doubtful that they can be developed when stochastic accuracy is used.

The alternatives that have been tried are genetic algorithms and tree search. The former seem to perform well, but they do not guarantee optimality. In the latter case, several alternatives for the linear and non-linear case were attempted: direct variable enumeration, use of union of cutsets with and without decomposition and finally, direct enumeration using an inverted tree. All these problems fail to perform well from the computational point of view in certain cases (Nguyen & Bagajewicz , 2008) although each shows substantial advantages in specific scenarios.

In this paper we propose 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, we propose a few modifications to these strategies and we also provide stopping criteria that are particular to minimum cost problems like the one we intend to solve. The method we propose 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 paper is structured as follows: We first review the problem definition, and we follow with a brief review of the existing tree search techniques. We then present our level traversal strategy (breadth-first tree search) and we finish with illustrations.

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 as base unit (Gala and Bagajewicz, 2006a, 2006b).

The direct enumeration tree search method is illustrated in figure 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.

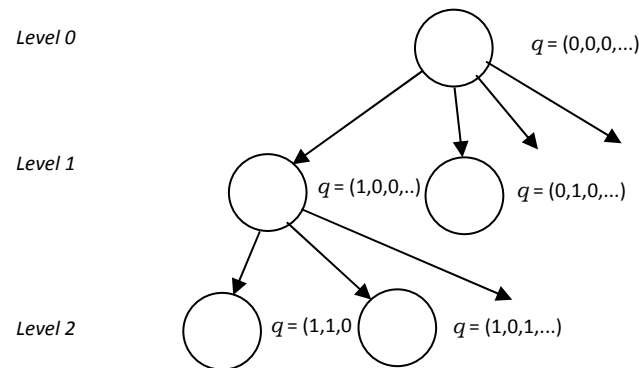


Figure 1. Tree search method

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 (Mayur and Bagajewicz, 2006a, 2006b). The cutsets-based methods were based on the notion that a cutset corresponds to a set of variables with which a material balance can be written. This important characteristic of cutset was utilized by Mayur and Bagajewicz (2006a) who replaced individual measurement by cutset as base unit in the tree search algorithm. The tree search procedure is essentially the same as the individual measurement-based procedure described above except that:

- i) A pre-processing step is needed before the tree search can be conducted, which is to find all cutsets containing at least one variable of interest (called key variable) of the problem,
- ii) In each node, the solution (to be evaluated for cost and feasibility) is the union of active (selected) cutsets,
- iii) When the stopping criterion is satisfied, one should back up two levels.

The virtue of this cutsets-based method is that only meaningful measurements that contribute to the redundancy or observability of variables of interest 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 single measurement-based tree search. These two properties help cutsets-based tree search find feasible nodes much more rapidly than single measurement-based counterpart.

Although tree enumeration using cutsets is suitable for middle scale problem (number of streams ≥ 20), it still has one limitation, which is that for large scale problems (number of streams ≥ 40), the number of cutsets may be too large and the number of nodes in the tree that needs to be explored is prohibitively large: hence the computation time can be as long as several days. To overcome this computational limitation, Mayur and Bagajewicz (2006b) proposed the “decomposition of process graph network” algorithm. The algorithm still makes use of cutsets but the process graph is decomposed into sub-graphs to reduce the number of cutsets in the candidate lists and hence reduce the size of the tree. Original cutsets that contain elements of different subgraphs are reconstructed when two cutsets from these subgraphs are picked. The tree search procedure is almost the same as the procedure without decomposition except that the branching and stopping criterion are modified and a ringsum operations between active cutsets

are performed to find the cutsets that are “missed” when compared with the cutsets list of original process graph. This cutsets-based tree search coupled with decomposition technique has been shown to be a very efficient method for solving linear large scale problems (Mayur and Bagajewicz, 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 equations-based method).

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

Table 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 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.

We now present a level traversal (breath first) technique which takes advantage of certain properties of trees that are constructed using the minimum cost branching criteria.

3. Level Traversal Search

Let us start first by defining the following terms:

Sister nodes: a given node is a sister nodes of a current node when: i) it is at the same level (same number of sensors or cutsets), ii) has a higher cost (is at the right of the current node), and iii) both share the same parent 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 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.

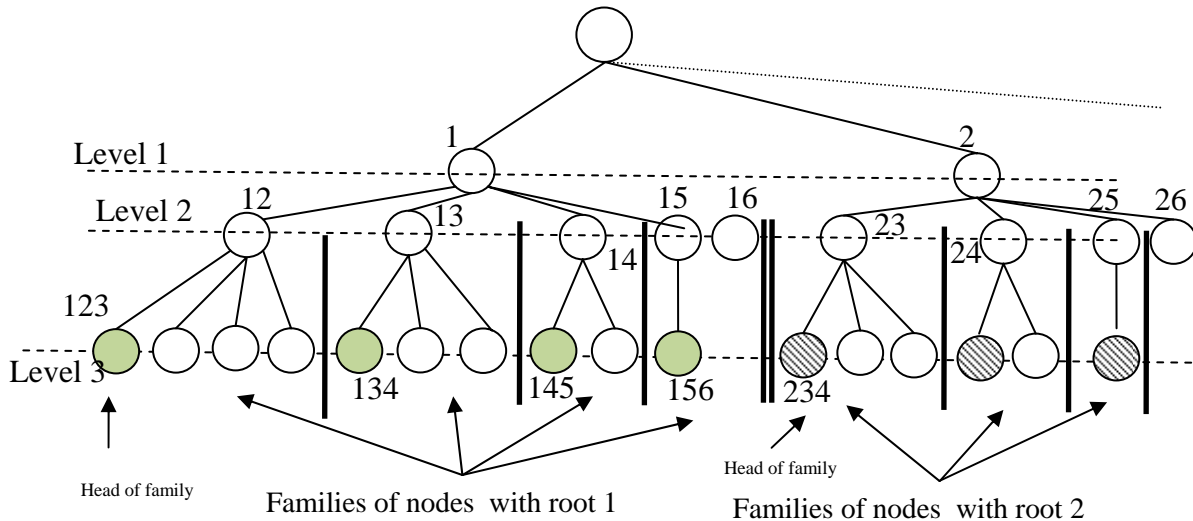


Figure 2. Families of nodes

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 than 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. Some sort of “inverted search” based on exploring the tree from right to left would have to be used. This is beyond the scope of this paper.

We now proceed to describe the calculation procedure

We start with a depth first search using the branching criteria based on adding the cheapest sensor until a feasible node is found or for a certain amount of additional branches. 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 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 4).

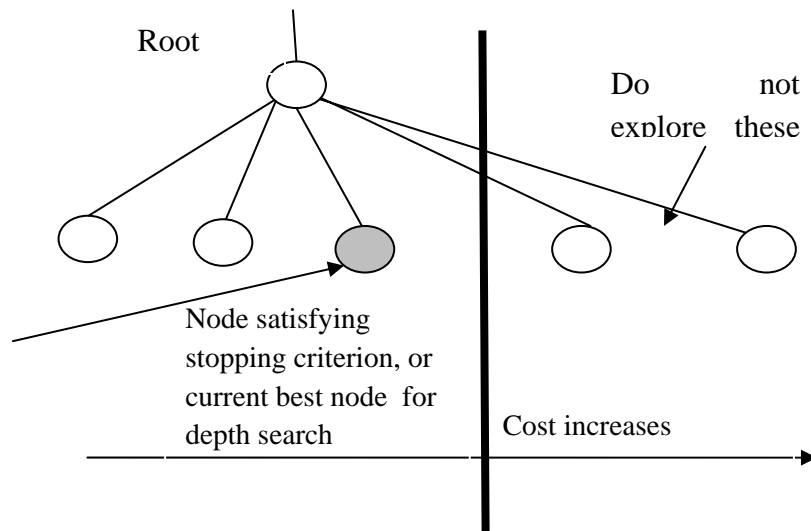


Figure 3. Stopping criterion

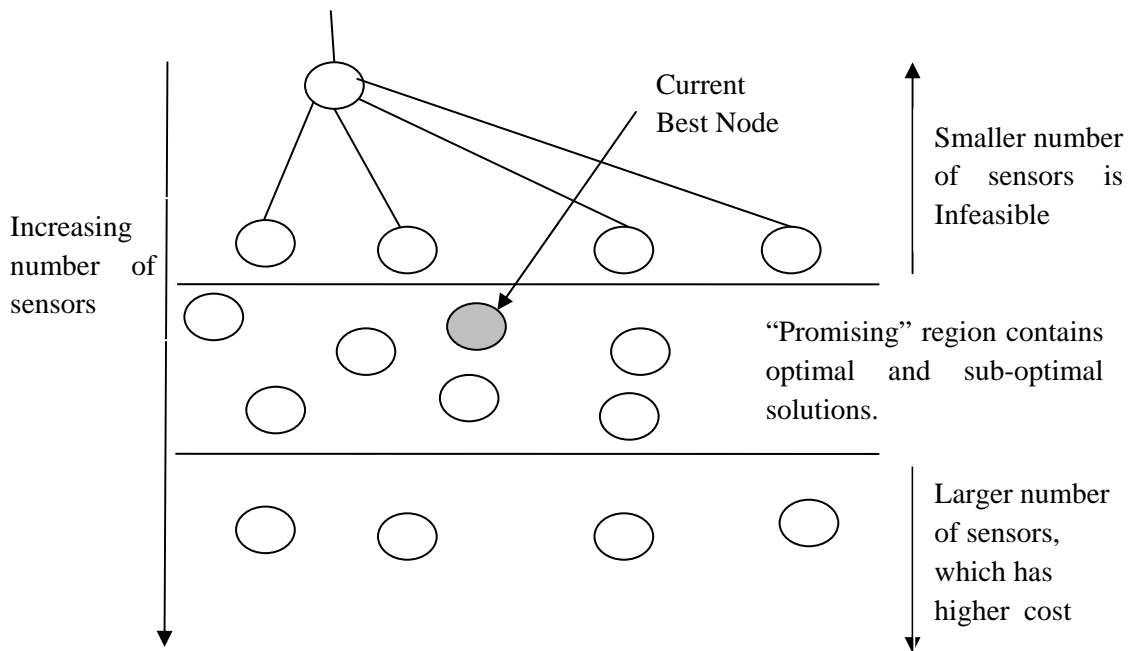


Figure 4. Searched space in horizontal search

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 we present two methods:

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 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.

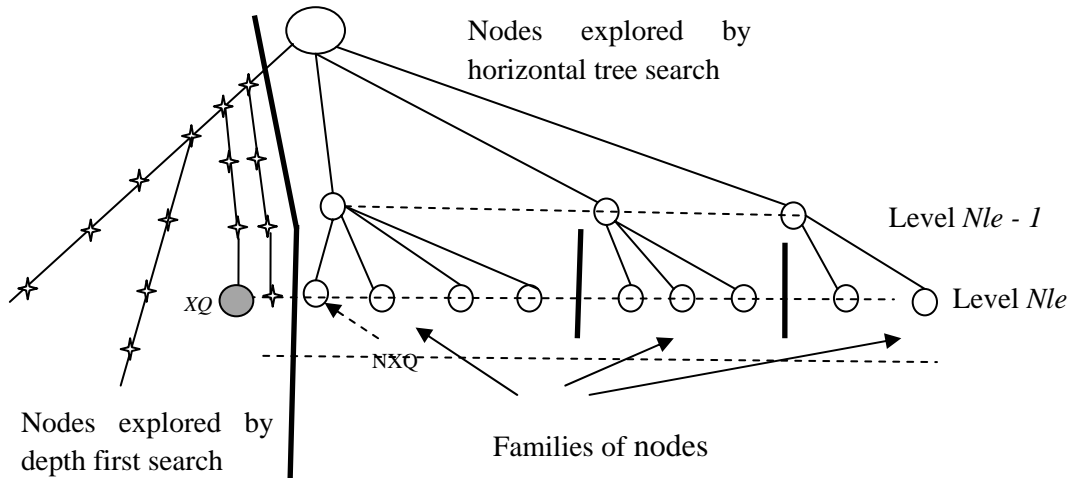


Figure 5. Level by Level Search

5. Hybrid Vertical and Level by Level Search

In this method, we combine breath 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 N
 - 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 6

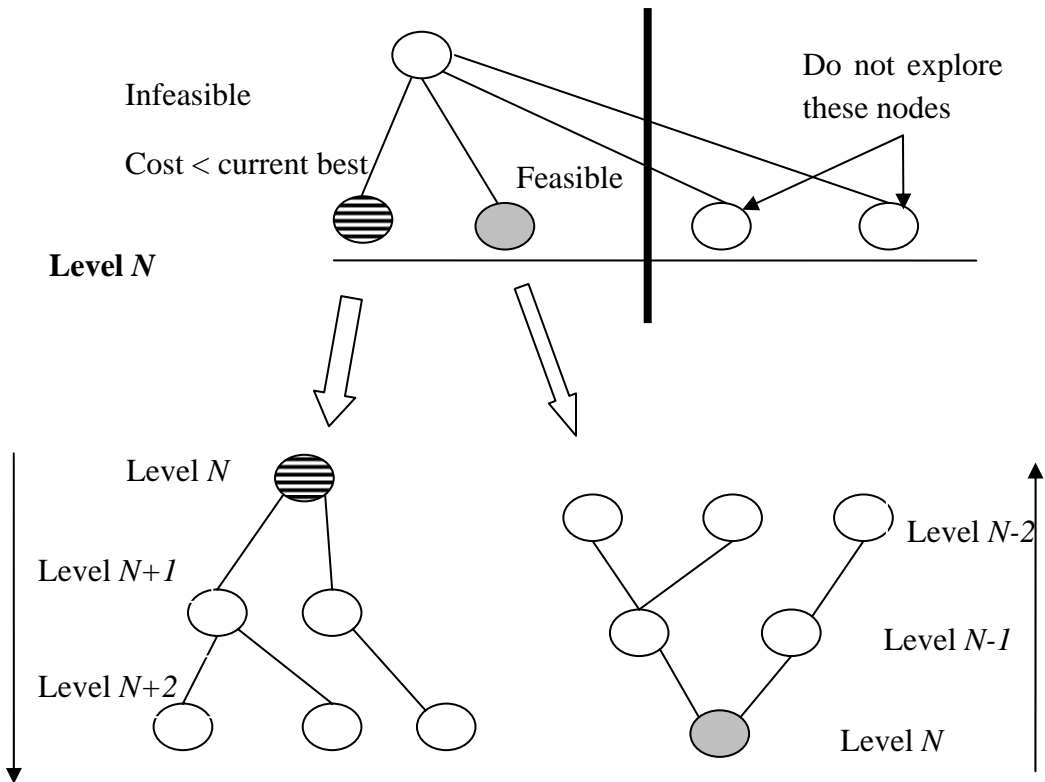


Figure 6. Hybrid Vertical and Level by Level Search

6. Examples

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

Example 1: Mineral flotation process. Consider a middle scale example, the mineral flotation process introduced by Smith and Ichiyen (1973). It is shown in figure 7.

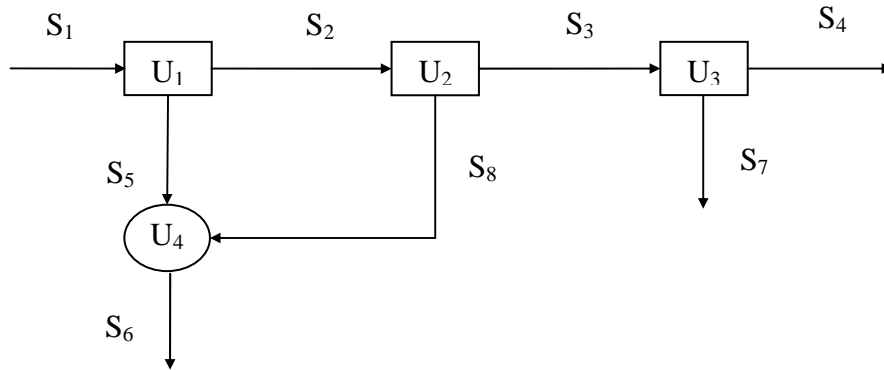


Figure 7. The mineral flotation process

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).

The nominal operating condition is given in table 2 (taken from Narasimhan and Jordache³¹):

Table 2. 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

Table 3. 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 case studies described in Nguyen & Bagajewicz (2008) are considered. 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 4

Table 4. 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 5, 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 5. 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)

We now illustrate the detail of steps in the Level by Level search for the design case 3 (MFP3). 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 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 $q = [123]$ 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 not 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 depth first 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 Nguyen & Bagajewicz (2008), 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 Nguyen & Bagajewicz (2008) 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 is part of ongoing work.

7. Conclusions

In this paper we have explored the efficacy of a new level traversal search in the tree of solutions for the instrumentation network design/upgrade problem. The 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.

REFERENCES

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

Bagajewicz M. Process Plant Instrumentation. Design and Upgrade (ISBN:1-56676-998-1), (Technomic Publishing Company) (<http://www.techpub.com>). Now CRC Press (<http://www.crcpress.com>) (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 D.Q. Nguyen. Stochastic-Based Accuracy of Data Reconciliation Estimators for Linear Systems. *Computers and Chemical Engineering.* 2008, 32(6), 1257-1269

Bagajewicz M. and M. Sánchez. Design and Upgrade of Non-redundant and Redundant Linear Sensor Networks. *AIChE J.* Vol. 45, No. 9, pp. 1927-1939. (1999a)

Bagajewicz M. and M. Sánchez. Duality of Sensor Network Design Models for Parameter Estimation. *AIChE J.*, 45, 3, pp. 661-664 (1999b).

Bagajewicz M. and M. Sánchez. Cost-Optimal Design of Reliable Sensor Networks. *Computers and Chemical Engineering.* Vol. 23, 11/12, pp. 1757-1762 (2000a).

Bagajewicz M. and M. Sánchez. Reallocation and Upgrade of Instrumentation in Process Plants. *Computers and Chemical Engineering.* 24, 8, pp. 1961-1980 (2000b).

Sánchez M. and M. Bagajewicz. On the Impact of Corrective Maintenance in the Design of Sensor Networks. *Industrial and Engineering Chemistry Research*. Vol. 39, no. 4, pp. 977-981 (2000c).

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

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

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.

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