Received February 5, 2022, accepted March 20, 2022, date of publication April 1, 2022, date of current version April 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3164250

A Parallelizable Algorithm for Stabilizing Large Sparse Linear Systems With Uncertain Interconnections

ALEKSANDAR ZEČEVIĆ, (Senior Member, IEEE), AND MARYAM KHANBAGHI, (Senior Member, IEEE)
Department of Electrical and Computer Engineering, Santa Clara University, Santa Clara, CA 95053, USA
Corresponding author: Maryam Khanbaghi (mkhanbaghi@scu.edu)

ABSTRACT This paper proposes a new method for permuting sparse matrices into an upper block triangular from. The algorithm is highly parallelizable, which makes it suitable for large-scale systems with uncertain interconnection patterns. In such cases, the proposed decomposition can be used to develop flexible decentralized control strategies that produce a different gain matrix whenever the configuration changes. Applications to interconnected microgrids and supply and demand networks are provided to illustrate the versatility of the proposed approach.

INDEX TERMS Large-scale systems, reconfigurable interconnections, uncertainty, decentralized control, sparse matrices, block triangular structure, parallel graph theoretic decompositions.

I. INTRODUCTION

The control of large-scale systems with uncertain interconnections has been studied extensively over the past few decades [1]–[11]. One of the key early results in this field was based on vector Lyapunov functions and the comparison principle, which were used to establish conditions under which the system would remain stable for a range of structural perturbations in the system [1]. More recently, techniques such as Linear Matrix Inequalities (LMIs), H_2 and H_∞ control design, adaptive neural networks and fuzzy control design have also been applied to this problem [12]–[24].

Despite the large body of work that exists on this topic, there are relatively few papers which examine how graph theoretic decompositions can be applied to control sparsely interconnected complex systems. Some research along these lines has been done on utilizing overlapping and bordered-block diagonal decompositions to identify appropriate structures for the gain matrix [25], [26]. There have also been a number of results related to networked control systems, where it is desirable to determine the sparsest control network that can satisfy a given set of performance requirements [27]–[29]. However, none of these papers consider reconfigurable systems in which the matrix that describes the interactions can potentially be permuted into an upper block triangular form for every permissible interconnection pattern. This possibility is clearly relevant in problems where interactions between subsystems can change unpredictably (as is the case with microgrids, for example) or when the subsystems are connected through a reconfigurable network whose topology is not fixed. With that in mind, the principal objective of this paper will be to develop an efficient decomposition algorithm that can determine if (and how) a large, sparse matrix can be reordered into an upper block triangular form.

The speed with which this is done is very important for large-scale applications, because decisions (including possible changes to the control design) often have to be made in a short amount of time. This requirement becomes even more critical if many different configurations need to be examined and evaluated. As a result, the algorithm that we propose will focus on ways to perform the decomposition in a multiprocessor environment.

The problem of permuting a sparse matrix into an upper (or lower) block diagonal form was addressed in the early 1960s by Sargent and Westerberg [30], who developed a graph theoretic algorithm for this purpose. Their approach was based on examining each edge in the graph separately, and building paths that eventually produce strong components. This idea was further developed by Tarjan [31], who increased the efficiency of the algorithm by introducing a simple way to keep track of the information that is gathered as the edges are searched.

The associate editor coordinating the review of this manuscript and approving it for publication was Feiqi Deng.

For the most part, Tarjan's algorithm (with its various refinements) remains the most effective sequential approach for identifying an upper block triangular structure (its complexity is $O(n + \tau)$ for an $n \times n$ matrix with τ nonzero elements) [32], [33]. For the applications that we will be interested in, however, this algorithm may not be sufficiently fast, so it is necessary to explore viable alternatives. The algorithm that we propose in this paper represents one such possibility, whose distinguishing feature is its inherent parallelizability (as well as the fact that it does not need to search for strong components).

It should be noted in this context that a number of authors have proposed methods for parallelizing a simpler version of this problem, which is known as *topological sorting* [34]–[37]. The objective in this case is to permute the matrix into a strictly upper (or lower) triangular form, which is important for a wide range of problems (including instruction and task scheduling in computer science, critical path analysis in project management, digital logic synthesis, etc.). For our purposes, however, this approach is not sufficiently general, since it implicitly assumes that the graph is *acyclic*. As a result, it cannot handle matrices where an upper block triangular structure is the only available option.

To the best of our knowledge, the problem of identifying an upper block triangular structure using multiple processors was addressed only in [38]. We should point out, however, that the approach described in this paper requires each processors to search the entire graph, and eventually uses the partial results to identify strong components. The method that we propose is considerably simpler in that respect, because it examines only a very limited subset of nodes and edges.

The paper is organized in the following manner. In Section II we formulate the control problem that we wish to solve, and identify classes of systems where the proposed strategy can be effective. In Sections III and IV we describe a highly parallelizable graph theoretic algorithm that can quickly permute large sparse matrices into an upper block triangular form (whenever this is possible, of course). Finally, in Section V we provide examples of practical large-scale models where this paradigm can be applied.

II. THE MODEL AND THE CONTROL PARADIGM

In this paper we will examine large sparse linear systems of the form

$$\dot{x} = A(t)x + Bu \tag{1}$$

in which the interconnections between the subsystems are subject to unpredictable changes, both in structure and in strength. In problems of this sort, decentralized control is typically the preferred option, both from a computational standpoint and from the standpoint of implementation. We will therefore assume that the overall system dynamics can be described as

$$\dot{x}_i = A_{ii}x_i + B_i u_i + \sum_{j \neq i} A_{ij}(t)x_j \quad (i = 1, 2, \ldots, s) \tag{2}$$

where $x_i \in R^{n_i}$ represents the state of the i-th subsystem, and $u_i \in R^{m_i}$ is the input that is associated with it. We will further assume that the elements of matrices A_{ii} are fixed and known, while the elements of matrices $A_{ij}(t)$ are uncertain and time varying.

In its most general form, this is a difficult problem, particularly when the number of subsystems is very large. We will therefore concentrate on the following two important scenarios, which allow for certain simplifications.

Scenario 1: All possible interconnection patterns that can arise are known, but the system can switch from one configuration to another unpredictably

Scenario 2: The system topology is *not* fixed, and we cannot anticipate all possible interconnection patterns in advance. However, once a new configuration becomes available, we have the option of rejecting it if we determine that it will adversely affect the stability of the system.

The first scenario has a long history, dating back to 1970s [2]. In problems of this sort, it is often convenient to describe the elements of matrices A_{ij} as $a_{pq}(t) = e_{pq}(t)\bar{a}_{pq}$, where $\bar{a}_{pq}$ denotes the "nominal" value of the element, and $0 \leq e_{pq}(t) \leq 1$. Changes in the system topology can then be modeled by setting e_{pq} to zero at appropriate locations in the matrix.

Following the definition introduced in [2], system (2) is said to be *connectively stable* if it is stable for all permissible values $e_{pq}(t)$. Necessary conditions for this were established in [1], using the concept of vector Lyapunov functions and the comparison principle. These conditions allow for the interactions to have the general form

$$h_i(x) = \sum_{i=1}^{s} e_{ij}(t)h_{ij}(x_j) \tag{3}$$

where $h_i(x)$ is a nonlinear function that can be bounded as

$$\|h_i(x)\| \leq \sum_{i=1}^{s} \bar{e}_{ij}\xi_{ij}\|x_j\| \tag{4}$$

Constants ξ_{ij} that appear in expression (4) are assumed to be positive, and $\bar{e}_{ij}$ are elements of the so-called *fundamental interconnection matrix* $\bar{E}$, which captures all possible interconnection patterns.

Although this result is general and mathematically elegant, we should point out that it is often conservative (in part because matrix $\bar{E}$ must include every single configuration that could potentially arise). In view of that, it would be desirable to identify class of systems where a simpler approach is possible.

The second scenario is more complicated, because the structure of matrices A_{ij} cannot be predicted. As a result, we do not know in advance which of its elements will have nonzero values in the next configuration. In such cases, it becomes necessary to consider each configuration as it becomes available, and establish (in a limited amount of time) whether it can be stabilized using decentralized control laws.

One way to approach both of these scenarios would be to determine whether the system matrix A can be permuted into an upper block triangular structure for every permissible interconnection pattern. If that turns out to be possible, one could ensure the stability of each configuration by applying appropriate decentralized control laws to the diagonal blocks.

When examining whether such a permutation exists for a given matrix A, it is common practice to represent each subsystem as a node in a directed graph, where an edge pointing from node j to node i indicates that subsystem j influences subsystem i. In our case, this would be equivalent to saying that edge (j, i) appears in the graph if and only if matrix A_{ij} contains at least one nonzero element. Because such a representation doesn't take into account the internal structure of the individual subsystems, it is commonly referred to as a *condensation* of the graph that corresponds to the system matrix A [2], [39]. Working with such reduced graphs is clearly desirable when dealing with large-scale systems, since it allows us to analyze a smaller matrix $\bar{A}$ whose dimensions are $s \times s$ (where s denotes the number of subsystems).

From a control perspective, it would be ideal if matrix $\bar{A}$ could be permuted into a *strictly upper triangular* form for all possible configurations, because we could then guarantee stability by simply ensuring that each subsystem is stable. Under such circumstances, we could use the *same* decentralized control law in all cases, and could treat the elements of matrices A_{ij} as completely uncertain.

In cases when we can only obtain an upper block triangular structure, it becomes necessary to design decentralized control laws that can stabilize clusters of subsystems (each of which corresponds to a different diagonal block). It is important to recognize, however, that this structure still offers some significant advantages, because it allows us to apply techniques such as LMI-based decentralized control design to *individual blocks* (as opposed to the entire matrix A, which is large). This makes a great deal of difference, since LMI optimization can be a computationally intensive procedure [40], [41]. We should also note that the control design can be decoupled in this case, since each diagonal block consists of a different set of subsystems.

In light of these observations, we will now focus our attention on developing an efficient algorithm that can determine whether (and how) a large, sparse matrix can be permuted into an upper block triangular form. Although a number of algorithms have been developed for this purpose over the past few decades, they are usually not suitable for the kinds of problems that we are interested in, because execution speed is critical in most such cases. It is not difficult to see why this is so - in Scenario 1, for example, it may be necessary to examine a very large number of different interconnection patterns, and determine if each one of them is stable (or stabilizable). In Scenario 2, on the other hand, we need to quickly identify whether a potential change in the configuration is acceptable, and modify the control laws accordingly if this is indeed the case.

The algorithm that we will describe in the following sections is well suited for such applications, because it is simple, and is inherently parallelizable. What distinguishes it from other techniques is the fact that it does not need to identify cycles and/or strong components by an exhaustive search of all edges in the graph. Instead, it focuses on a limited subset of nodes and edges, and checks whether they belong to a cycle (which can be done in parallel).

III. THE DECOMPOSITION ALGORITHM

As we already noted in the previous section, a nonsymmetric matrix A can be represented using a directed graph in which an edge pointing from node i to node j indicates that $a_{ij} \neq 0$. For our purposes, however, it will be convenient to take a somewhat different approach, and describe the matrix using a *bipartite graph* (or bigraph, for short). In such a graph, node x_i corresponds to column i of matrix A, and y_j corresponds to row j.

It is well known that a matrix can be permuted into a strictly upper triangular form if and only if the corresponding graph is *acyclical* [42]. For that reason, most techniques for permuting a matrix into an upper block triangular form have focused on identifying all the cycles in the graph, and using this information to cluster vertices into *strong components* (a strong component is defined as a subgraph in which there is a closed path that passes through *all* of its nodes, but does not include nodes from other subgraphs [39]).

The first step in most such procedures is to ensure that the matrix has nonzero diagonal elements, using an appropriate pre-permutation. Doing so can be challenging in general, and involves finding what is known as a *maximal transversal* (or perfect matching) [33], [43]. For the problems that we are interested in, however, this will not be an issue, since the diagonal elements of the reduced matrix $\bar{A}$ represent *individual subsystems*. As a result, nodes x_i and y_i are guaranteed to be connected for $i = 1, 2, \ldots, s$.

Once a maximal transversal is identified, existing algorithms typically proceed by checking every edge in the graph, in order to establish whether or not it belongs to a cycle. This normally involves a depth-first search, combined with some sort of "backtracking" scheme. There have been some attempts to distribute this procedure across multiple processors [38], but even then, each processor searches the entire graph (which can be time consuming when the matrix is large).

The algorithm that we propose utilizes a different approach, which entails examining only a limited subset of nodes and edges. We will describe it in two stages, starting with the simpler case when the matrix can be permuted into a strictly upper triangular form. We will then show how this algorithm needs to be modified in order to produce an upper block triangular structure. As we proceed, we will highlight those elements of the proposed method that distinguish it from other techniques that have similar objectives.

Our ordering scheme proceeds by recursively reducing the size of set V whose elements represent a subset of nodes in the

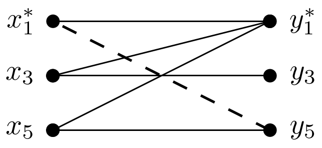

FIGURE 1. The first component of the bipartite graph.

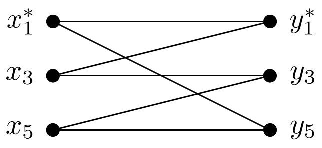

FIGURE 2. The second component of the bipartite graph.

graph. In the first iteration, this set includes *all* of the nodes, and the process terminates when V becomes empty.

Each iteration consists of three simple steps, which are described below.

STEP 1. Form a bipartite graph starting with the node in set V that has the smallest index. If node y_j is connected to some node x_i that appeared *before* it, flag node x_i.

STEP 2. Repeat Step 1 starting from a node that has not yet been visited. If any nodes in this graph have already appeared in a previous step, they should be flagged as well.

STEP 3. Continue executing Step 2 until all nodes in set V have been visited. Once this condition is met, remove all unflagged nodes from V, and place them into set W.

The following simple example explains the logic behind this strategy, and demonstrates how "flagging" certain nodes eventually produces an upper triangular stucture.

Example 1: Suppose that we are given a 5×5 matrix

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & * & & * & & * \\ 2 & & * & & * & \\ 3 & & & * & & \\ 4 & & & * & * & \\ 5 & * & & & & * \end{array} \tag{5}$$

and that we would like to permute it into an upper triangular form. If we represent this matrix as a bipartite graph, we initially obtain the configuration shown in Fig. 1, in which node y_i (which represents row i of the matrix) is connected to all nodes x_k for which $a_{ik} \neq 0$. As this graph is constructed, the next y-node is processed only after all x-nodes associated with the previous one have been added.

The fact that node y_5 is connected to a node that appeared *before* it (in this case, node x_1) is indicated by a dashed line in the graph. To understand the significance of such edges, it suffices to observe that the submatrix composed of rows and columns 1, 3 and 5 has the form

$$\begin{array}{c|ccc} & 1 & 3 & 5 \\ \hline 1 & * & * & * \\ 3 & & * & \\ 5 & \odot & & * \end{array} \tag{6}$$

This matrix is obviously *not* upper triangular, due to the presence of element a_{51}. In view of that, we will "flag" nodes x_1 and y_1 in the graph (which is indicted by an asterisk).

Since we haven't exhausted all the rows and columns of the matrix, we now continue building the bipartite graph starting from node 2 (which is the node with the smallest index among those that have yet to be examined). When we do so, we obtain a second component whose form is shown in Fig. 2.

We should recognize at this point that node 3 already appeared in the previous step. This means that there will be a nonzero element in the lower triangular part of the permuted matrix

$$\begin{array}{c|ccccc} & 1 & 3 & 5 & 2 & 4 \\ \hline 1 & * & * & * & & \\ 3 & & * & & & \\ 5 & * & & * & & \\ 2 & & & & * & * \\ 4 & & \odot & & & * \end{array} \tag{7}$$

In order to avoid that, we will "flag" node 3 as well.

If we now place all the unflagged nodes (in order of their appearance) in set $W(1) = \{5, 2, 4\}$ and the flagged ones in set $V(1) = \{1, 3\}$, the corresponding permutation $P = \{W(1), V(1)\} = \{5, 2, 4, 1, 3\}$ will produce a matrix whose first three rows have the form

$$\begin{array}{c|ccc:cc} & 5 & 2 & 4 & 1 & 3 \\ \hline 5 & * & & & * & \\ 2 & & * & * & & \\ 4 & & & * & & * \end{array} \tag{8}$$

Note that this submatrix has an upper triangular structure, which is precisely what we wanted to achieve.

In the next iteration, we need to repeat the process on the nodes in set $V(1) = \{1, 3\}$, and place any unflagged ones into set $W(2)$ (in order of their appearance). If sets $W(1)$ and $W(2)$ happen to be disjoint, we can group the remaining flagged nodes into set $V(2)$ and continue the process. As we do so, the size of sets $V(k)$ will *decrease* in each iteration, since $V(k-1) = W(k) \cup V(k)$ by construction. This ensures that we will encounter an empty set V at some point, unless the algorithm terminates prematurely.

The problem with this particular example is that the bipartite graph that originates at node 1 (which is used in the second iteration) contains node 5, which already appeared in $W(1)$. When this happens, the algorithm *terminates automatically*, because repeated elements are indicators of cycles in the graph (in this case, the cycle is $1 \rightarrow 5 \rightarrow 1$). As we already mentioned, this means that the matrix cannot be permuted into an upper triangular form.

The following theorem formalizes these observations, and shows that the presence of a cycle in the graph will necessarily result in a node that reappears after being included in some previous set $W(k)$.

Theorem 1: Let A be an $n \times n$ matrix, and suppose that the bipartite graph which corresponds to it contains a cycle. In that case, one of the nodes that belongs to the cycle will necessarily reappear, and the algorithm will terminate prematurely.

Proof: Suppose conversely that there is a cycle in the bipartite graph that corresponds to matrix A, but that the algorithm *does not* terminate prematurely. If this is true, there will be a pair of nodes a and b such that there is a path from node a to b and vice versa.

Let us now assume (without loss of generality) that node a appears before b in the first iteration of the algorithm. In that case there will be a dashed edge from y_b to x_a, and node a will be flagged. As such, it will be placed in set $V(1)$. As far as node b is concerned, there are two possibilities which we need to consider separately.

Case 1. If b is *not* flagged in the first iteration, it will belong to set $W(1)$. Given that $a \in V(1)$, we are bound to encounter this node again in the next iteration. Note, however, that the same holds true for node b, since there is a path from a to b. Because b already appeared in $W(1)$, the algorithm will end prematurely, which contradicts our assumption.

Case 2. If b is flagged in the first iteration, it will belong to set $V(1)$ (together with node a). Since we assumed that the algorithm does not terminate prematurely, one of these two nodes (let's say node b) will eventually appear in some set $W(m)$ (this is inevitable, because sets $V(k)$ decrease in size in each iteration). We should recognize at this point that nodes a and b can never be unflagged simultaneously, since there is a path from a to b and from b to a. Recalling that node b belongs to $W(m)$ (and is therefore *not* flagged in iteration m), it follows that node a must be placed in set $V(m)$.

Since the iterative process continues until no flagged nodes are left, we know that node a must eventually appear in some set $W(m+k)$. When that happens, node b will show up again as part of the bigraph that is used to form sets $W(m+k)$ and $V(m+k)$ (because there is a path from a to b). This, however, leads to a contradiction, because the algorithm will automatically terminate under such circumstances. **Q.E.D.**

Corollary 1: Suppose that the algorithm has executed k iterations without terminating prematurely, and that no nodes which belong to sets $W(1), W(2), \ldots, W(k)$ have reappeared. Then, these sets must be disjoint.

Proof: Suppose conversely that that the algorithm has executed k iterations without terminating, but that sets $W(i)$ and $W(j)$ are *not* disjoint. In that case, there must exist a node a that belongs to both $W(i)$ and $W(j)$.

If we assume that $i < j$, we can additionally claim that node a will appear in set $W(i)$ before it shows up in set $W(j)$. The fact that it appears again in the bigraph used to form sets $W(j)$ and $V(j)$ leads to a contradiction, however, since this means that the algorithm terminates prematurely in step j (and $j \leq k$). **Q.E.D.**

Corollary 1 indicates that the algorithm can proceed without modifications for as long as sets $W(1), W(2), \ldots, W(k)$ are disjoint. If there are no cycles in the graph, the union of these sets will ultimately produce a permutation that transforms matrix A into a strictly upper triangular form. We will consider how cycles can be handled in the following section, but before we do that, we first need to demonstrate how the process described in Example 1 can be parallelized using n processors (where n is the dimension of matrix A).

The idea behind the parallelization is remarkable simple, and is based on the observation that each processor can construct a bipartite graph like the one in Fig. 1 starting from a different node. In this process, four ordered sets are formed for each node i, and are stored as linked lists.

1) Set T_i, which contains all the nodes that are reachable from x_i (these are the *only* nodes that appear in the bigraph that originates at node i).
2) Set F_i, which consists of all nodes that are *flagged* due to connections that are labeled by dashed lines.
3) Set U_i, which is defined as $U_i = T_i \backslash F_i$.
4) Set L_i, which contains all the edges that correspond to dashed lines.

When evaluating the efficiency of this procedure, it is important to recognize that in a sparse matrix of dimension $n \times n$, the number of nodes in any given set T_i is typically much smaller than n. As a result, sets T_i, F_i, U_i and L_i can be constructed quickly. Once this procedure is completed, each processor sends the sets that it formed to a root processor (by default we will assume that this processor corresponds to node 1). This is a straightforward communication task, which can be executed in time proportional to $\log n$ on a hyper cube [44].

The following example demonstrates how the process works on a matrix whose graph contains no cycles.

Example 2: Consider the 9×9 matrix

$$\begin{array}{c|ccccccccc|} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline 1 & * & & & & & & * & & * \\ 2 & & * & & & & & & & \\ 3 & & * & * & & & & * & & \\ 4 & & * & & * & & & & * & \\ 5 & & & & * & * & & & * & \\ 6 & & * & & & & * & & & \\ 7 & & & & * & & * & * & & \\ 8 & & & & & & * & & * & \\ 9 & & & & & & & & * & * \end{array} \tag{9}$$

The algorithm starts from the bipartite graph that originates at node 1, which is shown in Fig. 3. It is not difficult to see that the corresponding sets T_1, F_1 and U_1 have the form $T_1 = \{1, 7, 9, 4, 6, 8, 2\}$, $F_1 = \{6\}$ and $U_1 = T_1 \backslash F_1 = \{1, 7, 9, 4, 8, 2\}$, respectively (note that these sets preserve the order in which the nodes appear). This allows us to initialize matrices T, F and U (which are associated with the entire matrix A), as $T(0) = T_1$, $F(0) = F_1$ and $U(0) = U_1$.

From set $T(0)$, it is readily observed that nodes 3 and 5 have not yet appeared in the graph, so our next step will be to consider the graph that originates at node 3. It is important to keep in mind that the corresponding sets T_3, F_3 and U_3 are already available to processor 1 at this point, so we can use them directly to update sets $T(0)$, $F(0)$ and $U(0)$.

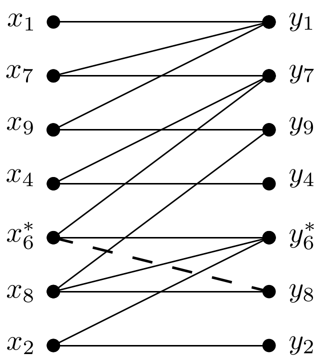

FIGURE 3. The bigraph that originates at node 1.

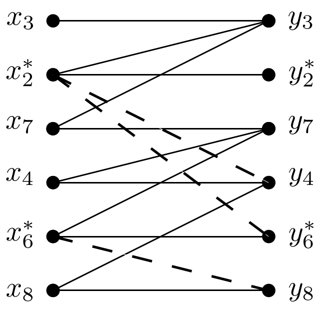

FIGURE 4. The bigraph that originates at node 3.

For the sake of completeness, in Fig. 4 we show the bigraph that originates at node 3, and gives rise to sets T_3, F_3 and U_3. As before, dashed lines correspond to situations when a y-node is connected to an x-node that appeared *before* it. Since there are three such edges in this case ((4, 2), (6, 2) and (8, 6)), nodes 2 and 6 need to be flagged, and sets T_3, F_3 and U_3 take the form $T_3 = \{3, 2, 7, 4, 6, 8\}$, $F_3 = \{2, 6\}$ and $U_3 = T_3 \backslash F_3 = \{3, 7, 4, 8\}$.

The process of updating sets T, F and U now proceeds in two stages, the first of which involves identifying which unflagged nodes in set T_3 have already appeared in one of the previous steps. These nodes (which ought to be flagged) are place in auxiliary set Q_3, while new nodes that arose in this step are placed in set R_3. Comparing Figs. 3 and 4, it is readily observed that nodes 7, 4 and 8 already appeared in the previous step, while node 3 is new. As a result, we have that $Q_3 = \{7, 4, 8\}$ and $R_3 = \{3\}$.

In the second stage, sets $T(0)$, $F(0)$ and $U(0)$ are updated in the following manner

$$\begin{aligned} T(1) &= T(0) \cup R_3 = \{1, 7, 9, 4, 6, 8, 2, 3\} \\ F(1) &= F(0) \cup F_3 \cup Q_3 = \{6, 2, 7, 4, 8\} \\ U(1) &= T(1) \backslash F(1) = \{1, 9, 3\} \end{aligned} \tag{10}$$

It is readily observed that $U(1)$ contains all the nodes that remain unflagged at this point, while $F(1)$ consists exclusively of flagged nodes. This allows us to partition set $T(1)$ (which contains all the nodes that have been visited so far) into two disjoint subsets - $U(1)$ and $F(1)$.

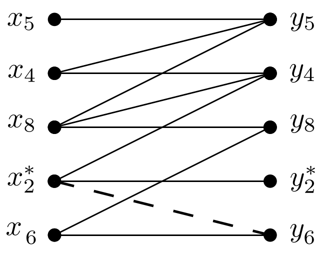

FIGURE 5. The bigraph that originates at node 5.

Since node 5 still hasn't appeared in the graph, we need to repeat the process one more time, using the bipartite graph that originates at node 5. This graph is shown in Fig. 5, and the corresponding sets T_5, F_5 and U_5 are $T_5 = \{5, 4, 8, 2, 6\}$, $F_5 = \{2\}$ and $U_5 = T_5 \backslash F_5 = \{5, 4, 8, 6\}$.

Recalling that this information is already available to processor 1, we easily obtain $Q_5 = \{4, 8, 6\}$ and $R_5 = \{5\}$, and matrices $T(1)$, $F(1)$ and $U(1)$ are updated as

$$\begin{aligned} T(2) &= T(1) \cup R_5 = \{1, 7, 9, 4, 6, 8, 2, 3, 5\} \\ F(2) &= F(1) \cup F_5 \cup Q_5 = \{6, 2, 7, 4, 8\} \\ U(2) &= T(2) \backslash F(2) = \{1, 9, 3, 5\} \end{aligned} \tag{11}$$

At this point the first iteration is complete, since $T(2)$ includes all of the nodes. We can therefore group the unflagged nodes into set $W(1) = U(2) = \{1, 9, 3, 5\}$ and the flagged ones into set $V(1) = F(2) = \{6, 2, 7, 4, 8\}$.

The second iteration focuses on the nodes in $V(1)$, starting with node 2 (which has the lowest index). Repeating the procedure described above, we obtain $T(2) = \{2, 4, 8, 6, 7\}$, $F(2) = \{2, 6, 4, 8\}$ and $U(2) = \{7\}$ (the corresponding bigraphs that appear at different stages in the process are shown in Fig. 6). Since sets $U(2)$ and $W(1)$ are obviously disjoint, we can set $W(2) = U(2) = \{7\}$ and $V(2) = F(2) = \{2, 6, 4, 8\}$, which allows us to partition the nodes as

$$W(1) \cup W(2) = \{1, 9, 3, 5, 7\} \tag{12}$$

and

$$V(2) = \{2, 4, 6, 8\} \tag{13}$$

Observing that node 2 has the smallest index in set $V(2)$, in the next iteration we obtain a graph that corresponds to the first two blocks in Fig. 6. The corresponding sets $T(1)$, $F(1)$ and $U(1)$ are obviously $T(1) = \{2, 4, 8, 6\}$, $F(1) = \{2\}$ and $U(1) = \{4, 8, 6\}$. The fact that sets $U(1)$ and $W(1) \cup W(2)$ are disjoint allows us to set $W(3) = U(1) = \{4, 8, 6\}$ and $V(3) = F(1) = \{2\}$, which produces

$$W(1) \cup W(2) \cup W(3) = \{1, 9, 3, 5, 7, 4, 8, 6\} \tag{14}$$

and

$$V(3) = F(1) = \{2\} \tag{15}$$

Since $V(3)$ contains only node 2 (which is not connected to any other nodes), it is readily observed that $W(4) = \{2\}$

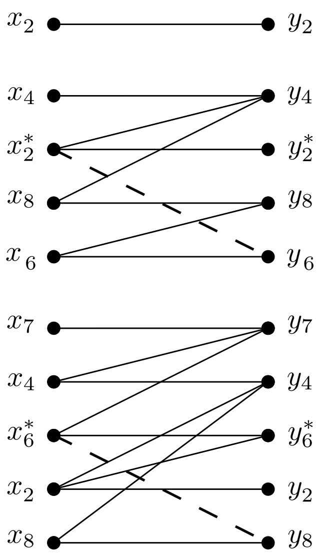

FIGURE 6. Graph that corresponds to the second iteration.

and that $V(4) = \emptyset$. Given that set V is now empty, it follows that

$$W(1) \cup W(2) \cup W(3) \cup W(4) = \{1, 9, 3, 5, 7, 4, 8, 6, 2\} \tag{16}$$

represents the desired ordering. The corresponding permuted matrix will then have the form

$$\begin{array}{c|ccccccccc|} & 1 & 9 & 3 & 5 & 7 & 4 & 8 & 6 & 2 \\ 1 & * & & & & * & & * & & \\ 9 & & * & & & & & * & & \\ 3 & & & * & & * & & & & * \\ 5 & & & & * & & * & * & & \\ 7 & & & & & * & * & & * & \\ 4 & & & & & & * & * & & * \\ 8 & & & & & & & * & * & \\ 6 & & & & & & & & * & * \\ 2 & & & & & & & & & * \end{array} \tag{17}$$

IV. THE MODIFIED ALGORITHM

If the algorithm terminates prematurely after disjoint sets $W(1), W(2), \ldots W(m)$ have been formed, we know that submatrix A_m (which corresponds to set $V(m)$) will *not* be permutable into an upper triangular form. In view of that, we will now consider a modification that allows us to continue the reordering process, and identify an upper block triangular structure in matrix A_m (if such a structure exists, of course). This modification consists of four straightforward steps which can be described as follows.

STEP 1. Check each dashed edge in the bigraph that corresponds to matrix A_m, and determine whether it is part of a cycle.

STEP 2. Group the cycles into k disjoint subsets $\{X_1, X_2, \ldots, X_k\}$, and label nodes that do not belong to any cycle as $X_{k+1}, X_{k+2}, \ldots, X_r$.

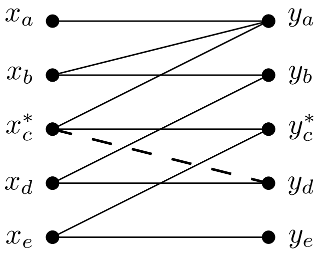

FIGURE 7. Dashed edge that does not belong to a cycle.

STEP 3. Form a directed graph whose nodes correspond to sets $\{X_1, X_2, \ldots, X_r\}$. In this graph, nodes i and j will be connected if there exists a nonzero element a_{pq} in matrix A_m such that $p \in X_i$ and $q \in X_j$.

STEP 4. Apply the algorithm described in Section III to the matrix that corresponds to the directed graph obtained in Step 3.

To explain why these steps are necessary (and how they can be implemented in a multiporocessor environment), we should first observe that matrix A_m is smaller in size than A (often significantly so, if matrix A is sparse). As a result, we will need to examine only a *limited subset* of the nodes and edges that appear in the original condensed graph. To see what this entails, let us assume that node a has the smallest index in set $V(m)$, in which case the ordering of matrix A_m will start with x_a and y_a. We should recognize at this point that all the dashed edges that occur in this bigraph are already available to processor 1, and are contained in sets L_i $(i = 1, 2, \ldots, n)$. With that in mind, the first step of the modified algorithm is to check which of these edges actually belong to cycles. This is necessary, because dashed edges needn't always correspond to cycles (the bipartite graph in Fig. 7 shows one such possibility, where edge (d, c) is dashed, but does not belong to a cycle).

Checking whether or not edge (α, β) belongs to a cycle can be done by examining the bipartite graph that originates at node β, and verifying if node α appears in it (which is easy to do, since set T_β is already available). If this happens to be the case, all the nodes on the path from β to α will belong to the same cycle, and we need to record them.

It is important to keep in mind in this context that checking dashed edges is *not* equivalent to searching for strong components (which entails finding a path that includes *all* the nodes in the subgraph). Instead, we are only interested in finding the *shortest cycle* that includes nodes α and β. In that respect, the proposed algorithm is significantly simpler than the ones proposed in the existing literature. It also allows us to examine dashed edges *in parallel*, by assigning each one to a different processor (together with the corresponding set T_β).

Because the dashed edges are already known at the start of this process, there is no need to perform an exhaustive search of all the edges in the graph, or to use a "backtracking" procedure to identify the cycle that they may belong to (which all other techniques for identifying block triangular structures do in one way or another). Instead, everything is performed in a *single step*, using p processors (where p represents the number of dashed lines in the bigraph).

The communication tasks that are associated with this search are a single node scatter operation that originates at processor 1, and a single node gather operation that occurs once the status of each edge has been determined. Both of these tasks can be executed in time proportional to $\log p$ on a hypercube architecture with p processors [44].

After all the dashed edges have been checked, processor 1 can compare and aggregate the paths that were obtained, and partition the nodes that belong to cycles into r disjoint sets. These sets correspond to different "supernodes" (or composite nodes, as they are sometimes called) in the graph, which can be combined with the nodes that do not belong to any cycle. At that point, we can apply the algorithm described in the previous section to identify a permutation that reorders matrix A_m into an upper block triangular form.

The following example demonstrates how the modified procedure works in practice.

Example 3: Suppose that the original algorithm terminated prematurely after m iterations, and that matrix A_m (which corresponds to the nodes in set $V(m)$) has the form

$$\begin{array}{c|ccccccc|} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & * & * & & & & * & \\ 2 & & * & & & & & \\ 3 & & & * & & * & & \\ 4 & & * & * & * & * & & \\ 5 & & & & * & * & & \\ 6 & * & & & & & * & \\ 7 & & & & & & * & * \end{array} \quad (18)$$

It is easily verified that this matrix cannot be permuted into a strictly upper triangular form, since the corresponding directed graph contains cycles. As a result, we need to apply the modified algorithm, and establish whether A_m can be permuted into an upper block triangular form.

The first step of this algorithm produces the bipartite graph shown in Fig. 8, which contains four dashed edges: (6, 1), (4, 3), (4, 5), and (1, 6).

We can easily verify whether each of these edges corresponds to a loop by examining the bipartite graphs that originate in nodes 1, 3, 5 and 7, respectively (as we already noted, this can be done in parallel). These graphs (which are shown in Fig. 9) indicate that the following four sets of nodes belong to cycles: $C_1 = \{1, 6\}$, $C_2 = \{6, 1\}$, $C_3 = \{3, 5, 4\}$ and $C_4 = \{5, 4\}$. Note that the bipartite graphs in this figure are not complete, because we can stop at the point when the desired node appears for the first time. This obviously contributes to the efficiency of the algorithm.

Having obtained sets C_1, C_2, C_3 and C_4, we can now easily aggregate them and partition the rows and columns of matrix A_m into disjoint sets. When we do so, we obtain $X_1 = \{1, 6\}$, $X_2 = \{3, 4, 5\}$, $X_3 = \{2\}$ and $X_4 = \{7\}$ (the last two sets contain only one element, because nodes 2 and 7 are not part of any cycle). Using this information, we can form

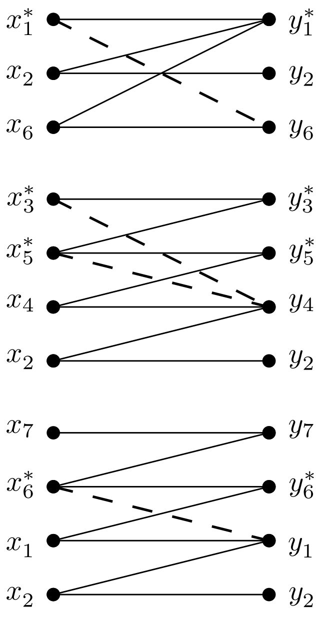

FIGURE 8. The initial bigraph.

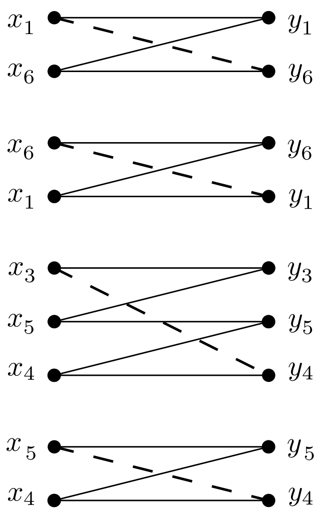

FIGURE 9. Possible cycles in the graph.

a condensation of the graph that corresponds to matrix A_m, which is composed of supernodes X_1 and X_2, together with nodes X_3 and X_4.

In order to do that, we need to examine each node in sets X_1, X_2, X_3 and X_4, and establish whether it is connected to any nodes outside of its own set. This can be easily done in parallel, by having a different processor check each node. When we do so, we obtain the following reduced matrix

$$\begin{array}{c|cccc|} & X_1 & X_2 & X_3 & X_4 \\ \hline X_1 & * & & * & \\ X_2 & & * & * & \\ X_3 & & & * & \\ X_4 & * & & & * \end{array} \quad (19)$$

Applying our original algorithm to this matrix, we easily obtain permutation vector $P_C = \{X_2, X_4, X_1, X_3\}$, which translates into $P = \{3, 4, 5, 7, 1, 6, 2\}$ once the supernodes are expanded. The resulting reordered matrix

$$\begin{array}{c|ccc|c|cc|c|}
 & 3 & 4 & 5 & 7 & 1 & 6 & 2 \\
\hline
3 & * & & * & & & & \\
4 & * & * & * & & & & * \\
5 & & * & * & & & & \\
\hline
7 & & & & * & & * & \\
\hline
1 & & & & & * & * & * \\
6 & & & & & * & * & \\
\hline
2 & & & & & & & * \\
\end{array} \tag{20}$$

is obviously upper block triangular.

V. APPLICATIONS

To illustrate the practical value of the proposed approach (as well as its versatility), in this section we will consider two very different problems - energy exchanges between interconnected microgrids and optimal inventory management. In both cases the system matrix can be large and sparse, and the interconnection patterns between subsystems can change unpredictably.

A. CONTROL OF INTERCONNECTED MICROGRIDS

A schematic representation of a system of interconnected microgrids is shown in Fig. 10. This model assumes that individual microgrids can exchange energy through a common bus, and can indicate their needs through a communication network.

Each microgrid that constitutes such a system can be described by a pair of linear differential equations, which reflect its battery dynamics and the way power is generated, distributed and consumed. For microgrid i, the first of these equations has the form

$$\frac{dE_b^{(i)}}{dt} = (1 - \alpha_i)P_s^{(i)} + P_b^{(i)} - \gamma_i E_b^{(i)} \tag{21}$$

where $E_b^{(i)}$ is the energy stored in the battery, $P_s^{(i)}$ denotes the generated solar power, and $P_b^{(i)}$ is the power associated with battery discharge. The constants α_i and γ_i represent the fraction of the solar power that is delivered to the load, and rate of self-discharge, respectively [45].

The second equation stems from the requirement that the generated solar power ($P_s^{(i)}$), the battery discharge ($P_b^{(i)}$), the power consumed by the load ($P_L^{(i)}$) and the net power that microgrid i transfers to other microgrids ($P_{ex}^{(i)}$) must be balanced. This condition can be formally stated as

$$P_L^{(i)} = \alpha_i P_s^{(i)} + P_b^{(i)} - P_{ex}^{(i)} \tag{22}$$

In the most general scenario, microgrid i can deliver energy to some microgrids and receive it from others. Because there is some flexibility in the way such exchanges are implemented, it is always possible to schedule them so that the power delivered by microgrid i to other microgrids is proportional to the energy absorbed by load i up to time t. If we take that into account, $P_{ex}^{(i)}$ can be expressed as

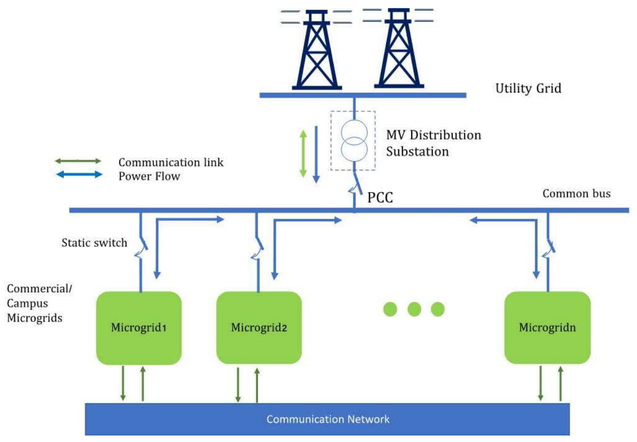

FIGURE 10. Schematic description of a system of interconnected microgrids.

$$P_{ex}^{(i)} = \frac{dE_{ex}^{(i)}}{dt} = \sum_{k \neq i} a_{ki} E_L^{(i)} - \sum_{k \neq i} a_{ik} E_L^{(k)} \tag{23}$$

where $E_L^{(k)}$ is the energy consumed by load k up to time t, and coefficients a_{ki} determine the amount of energy that is exchanged between pairs of microgrids. Since these coefficients represent design parameters, they can be chosen in a way that optimizes battery storage while satisfying the load demand [45].

Setting $E_b^{(i)}$ and $E_L^{(i)}$ as the state variables for microgrid i, the overall system can be represented as

$$x_i = A_{ii}x_i + B_i u_i + \sum_{j \neq i} A_{ij}(t)x_j + P_i \quad (i = 1, \ldots, s) \tag{24}$$

where $P_i = [(1 - \alpha_i)P_s^{(i)} \quad \alpha_i P_s^{(i)}]^T$ and input u_i denotes the control action (which amounts to charging or discharging battery i). Matrices A_{ij} are assumed to be time-dependent in this model because energy exchange patterns can vary unpredictably. However, any such change must first be approved by a higher level operating unit, which needs to take into account the possible stability implications, and determine if the feedback law needs to be modified.

Because the number of microgrids can potentially be large (and the exchanges between them are limited), matrix A is likely to be sparse. Given the physical and economic constraints under which such systems operate, it is also likely that this matrix will be permutable into an upper triangular (or upper block triangular) form for most configurations that are of practical interest. It is therefore reasonable to assume that models of this sort will typically conform to Scenario 2 that was described in Section II. As we already noted, in such cases it is essential to have an efficient ordering algorithm that will allow us to quickly determine whether additional control action is necessary.

B. INVENTORY CONTROL

Today's globalized economy requires ever increasing levels of supply chain agility and inventory management in order to improve operational efficiency. From a manufacturing perspective, this means that the 'making-to-stock' mode will need to be replaced by the 'making-to-order' mode (and perhaps the 'engineering-to-order' mode as well). In order to implement such a change, production scheduling and inventory control strategies will have to become considerably more flexible. This task is complicated, however, by uncertainties in the manufacturing process and other unpredictable factors (such as the availability of raw materials or customer demand).

When analyzing the impact of consumer demand on production scheduling and inventory management, it is important to recognize that different industries require different levels of agility. Manufacturers that provide basic components to industries with a volatile demand (such as those that deliver thin glass to companies that make flat panels) often serve more than a dozen large customers, and must be able to quickly respond to their needs. This is not the case with the automotive industry, for example, where manufacturers have more time to react.

Models that are used to study production scheduling and inventory management must also account for forecast variability. Although it is often possible to identify the largest sources of uncertainty, the fact remains that no demand forecast is completely accurate. This is due to a number of factors, which include (but are not limited to) competition, consumer behavior and the availability of supplies. One must also consider the unpredictable nature of global economic conditions, as well random events such as political crises or the recent COVID pandemic.

The impact of uncertain demand forecasting can be minimized by optimal inventory/safety stocks management. Over the past few decades, a number of papers have addressed this problem using control theory [46]–[53]. One possible approach involves modeling inventory levels as discrete-time dynamic systems of the form

$$\begin{aligned} x_i^{(j)}(k+1) &= x_i^{(j)}(k) + \gamma_{ji} F_i^{(j)}(k) - d_i(k) \\ &\quad - \sum_{l \neq i} e_{ji} x_l^{(j)}(k) \\ y_i^{(j)}(k+1) &= y_i^{(j)}(k) \end{aligned} \tag{25}$$

In equation (25), $x_i^{(j)}(k)$ represents the inventory level at time k for a specific product attribute j at location i, $F_i^{(j)}(k)$ denotes the product volume at time k for attribute j at location i, and $d_i(k)$ is the customer demand associated with this product volume (which is a stochastic variable in general). Constants γ_{ji} and e_{ji} are assumed to be known, and $y_i^{(j)}(k)$ represents the measured inventory level (which is what one would like to optimize).

This model assumes that a product can have as many as q different attributes, and that the inventory is distributed across

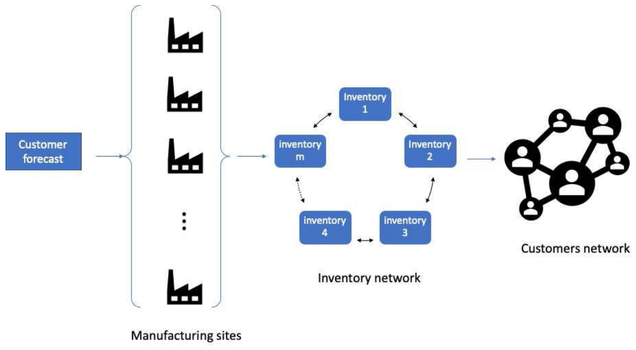

FIGURE 11. **Simplified schematic description of a supply and demand network.**

m different locations. If we introduce an aggregate vector

$$x(k) = \left[x_1^{(1)} \cdots x_1^{(q)} \cdots x_m^{(1)} \cdots x_m^{(q)} \right]^T \tag{26}$$

and treat the product volume as the system input (which we will denote by $u(k)$), the inventory dynamics can be represented as

$$x(k+1) = Ax(k) + Bu(k) - d(k) \tag{27}$$

where A is a matrix of dimension $mq \times mq$. When m and q are large, matrix A is typically sparse, since only a limited amount of material from one inventory location needs to be moved to another one at any given time. When such exchanges occur, the material is usually moved to only a few other locations (which further contributes to the sparsity of matrix A).

From the perspective of this paper, it is important to recognize that the directed graph associated with matrix A can contain cycles. This possibility is indicated in Fig. 11, which provides a schematic description of the supply and demand network.

In such cases, matrix A is usually permutable into an upper block triangular form (given the nature of the inventory network). Since this matrix needs to be reordered whenever there is a change in supply or demand (which happens frequently), it is important to have an algorithm that can perform this task quickly. The need for an efficient reordering algorithm is amplified by the fact that fluctuations in demand are unpredictable, which means that the exact structure of matrix A cannot be known until they actually occur. In that respect, inventory dynamics bears a certain resemblance to energy exchanges between interconnected microgrids.

VI. CONCLUSION

In this paper, we presented a new method for permuting large, sparse matrices into an upper block triangular structure. A distinguishing feature of the algorithm is its inherent parallelizability, and the fact that it does not need to identify cycles and/or strong components by an exhaustive search of all edges in the graph. As such, it is well suited for certain classes of large-scale problems (particularly those in which interconnection patterns change frequently). Applications to interconnected microgrids and inventory management were provided to illustrate the practical value and versatility of the proposed approach.

REFERENCES

[1] D. D. Siljak, "Stability of large-scale systems under structural perturbations," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-2, no. 5, pp. 657–663, Nov. 1972.

[2] D. D. Siljak, "Dynamic systems," in *Stability and Structure*. Amsterdam, The Netherlands: North-Holland, 1978.

[3] M. Garcia-Rubio and R. J. Thomas, "Decentralized stabilizability conditions for large-scale electric power systems," in *Proc. 23rd IEEE Conf. Decis. Control*, Dec. 1984, pp. 185–190.

[4] L. T. Grujic, A. A. Martynyuk, and M. Ribens-Pavella, *Large Scale Systems Stability under Structural and Singular Perturbations* (Lecture Notes in Control and Information Sciences). Berlin, Germany: Springer, 1987.

[5] L. Shi and S. K. Singh, "Decentralized control for interconnected uncertain systems: Extenstions to higher order uncertainties," *Int. J. Control*, vol. 57, pp. 1453–1468, 1993.

[6] S. Jain, F. Khorrami, and B. Fardanesh, "Adaptive nonlinear excitation control of power systems with unknown interconnections," *IEEE Trans. Control Syst. Technol.*, vol. 2, no. 4, pp. 436–446, Dec. 1994.

[7] M. Akar and U. Ozguner, "Decentralized techniques for the analysis and control of Takagi-Sugeno fuzzy systems," *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 6, pp. 691–704, Dec. 2000.

[8] D. M. Stipanoviá and D. D. Šiljak, "Connective stability of discontinuous dynamic systems," *J. Optim. Theory Appl.*, vol. 115, no. 3, pp. 711–726, Dec. 2002.

[9] A. Swarnakar, H. J. Marquez, and T. Chen, "A new scheme on robust observer-based control design for interconnected systems with application to an industrial utility boiler," *IEEE Trans. Control Syst. Technol.*, vol. 16, no. 3, pp. 539–547, May 2008.

[10] A. I. Zecevic and D. D. Siljak, "Control of complex systems," in *Structural Constraints Uncertainty*. Springer, 2010.

[11] M. Pirani, T. Costa, and S. Sundaram, "Stability of dynamical systems on a graph," in *Proc. 53rd IEEE Conf. Decis. Control*, Dec. 2014, pp. 613–618.

[12] S.-W. Kim, B. K. Kim, and C.-J. Seo, "Robust and reliable H_∞state feedback control: A linear matrix inequality approach," *Trans. Control, Autom. Syst. Eng.*, vol. 2, pp. 31–39, Dec. 2000.

[13] D. Stipanovic, G. Inhalan, R. Teo, and C. Tomlin, "Decentralized overlapping control of a formation of unmanned arial vehicles," *Automatica*, vol. 40, pp. 1285–1296, Apr. 2004.

[14] A. I. Zecevic and D. D. Siljak, "Global low-rank enhancement of decentralized control for large-scale systems," *IEEE Trans. Autom. Control*, vol. 50, no. 5, pp. 740–744, May 2005.

[15] H. N. Wu and H. Y. Zhang, "Reliable H_∞fuzzy control for continuous-time nonlinear systems with actuator failures," *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 6, pp. 609–618, Dec. 2006.

[16] W. Chen and J. Li, "Decentralized output-feedback neural control for systems with unknown interconnections," *IEEE Trans. Syst., Man, Cybern. B. Cybern.*, vol. 38, no. 1, pp. 258–266, Feb. 2008.

[17] O. Ou, Q. Hui, and H. Zhang, "Stability analysis and h infinity decentralized control for discrete-time nonlinear large-scale systems via fuzzy control approach," in *Proc. 6th Int. Conf. Fuzzy Syst. Knowl. Discovery*, 2009, pp. 166–170.

[18] A. Atig, F. Drua, D. Lefebvre, K. Abderrahim, and R. Ben Abdennour, "Neural network control for large scale systems with faults and perturbations," in *Proc. Conf. Control Fault-Tolerant Syst. (SysTol)*, Oct. 2010, pp. 305–310.

[19] L. J. Long and J. Zhao, "Decentralized adaptive fuzzy output-feedback control of switched large-scale nonlinear systems," *IEEE Trans. Fuzzy Syst.*, vol. 23, no. 5, pp. 1844–1860, Oct. 2015.

[20] L. Cao, H. Li, N. Wang, and Q. Zhou, "Observer-based event-triggered adaptive decentralized fuzzy control for nonlinear large-scale systems," *IEEE Trans. Fuzzy Syst.*, vol. 27, no. 6, pp. 1201–1214, Jun. 2016.

[21] E. J. Davison, A. G. Aghdam, and D. E. Miller, *Decentralized Control of Large-Scale Systems*. Springer, 2020.

[22] V. Vesely, "Novel approach to decentralized controller design for large scale uncertain linear systems," *Int. J. Innov. Comput., Inf. Control*, vol. 17, pp. 1571–1580, Dec. 2021.

[23] J.-D. Liu, B. Niu, Y.-G. Kao, P. Zhao, and D. Yang, "Decentralized adaptive command filtered neural tracking control of large-scale nonlinear systems: An almost fast finite-time framework," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3621–3632, Aug. 2021.

[24] S. Tong, Y. Li, and Y. Liu, "Observer-based adaptive neural networks control for large-scale interconnected systems with nonconstant control gains," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 4, pp. 1575–1585, Apr. 2021.

[25] A. I. Zeáeviâ and D. D. Šiljak, "A new approach to control design with overlapping information structure constraints," *Automatica*, vol. 41, no. 2, pp. 265–272, Feb. 2005.

[26] A. I. Zeáeviâ and D. D. Šiljak, "Control design with arbitrary information structure constraints," *Automatica*, vol. 44, no. 10, pp. 2642–2647, Oct. 2008.

[27] L. Furieri, Y. Zheng, A. Papachristodoulou, and M. Kamgarpour, "Sparsity invariance for convex design of distributed controllers," *IEEE Trans. Control Netw. Syst.*, vol. 7, no. 4, pp. 1836–1847, Dec. 2020.

[28] F. Dörfler, M. R. Jovanović, M. Chertkov, and F. Bullo, "Sparsity-promoting optimal wide-area control of power networks," *IEEE Trans. Power Syst.*, vol. 29, no. 5, pp. 2281–2291, Sep. 2014.

[29] M. Razeghi-Jahromi and A. Seyedi, "Stabilization of networked control systems with sparse observer-controller networks," *IEEE Trans. Autom. Control*, vol. 60, no. 6, pp. 1686–1691, Jun. 2015.

[30] R. W. Sargent and A. W. Westerberg, "Speed-up in chemical engineering design," *Trans. Inst. Chem. Eng.*, vol. 42, pp. 190–197, Dec. 1964.

[31] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, Jun. 1972.

[32] I. S. Duff and J. K. Reid, "An implementation of Tarjan's algorithm for the block triangularization of a matrix," *ACM Trans. Math. Softw.*, vol. 4, no. 2, pp. 137–147, Jun. 1978.

[33] A. J. Osiadacz, "Direct methods for sparse matrices," *Automatica*, vol. 26, no. 2, pp. 441–443, Mar. 1990.

[34] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM J. Comput.*, vol. 10, no. 4, pp. 657–675, 1981.

[35] M. C. Er, "A parallel computation approach to topological sorting," *Comput. J.*, vol. 26, no. 4, pp. 293–295, Nov. 1983.

[36] J. Ma, K. Iwama, T. Takaoka, and Q.-P. Gu, "Efficient parallel and distributed topological sort algorithms," in *Proc. IEEE Int. Symp. Parallel Algorithms Archit. Synth.*, Dec. 1997, pp. 378–383.

[37] P. Chaudhuri, "Parallel incremental algorithms for analyzing activity networks," *Parallel Algorithms Appl.*, vol. 13, no. 2, pp. 153–165, Aug. 1998.

[38] A. Mocanu and N. Tapus, "Sparse matrix permutations to a block triangular form in a distributed environment," in *Proc. IEEE 9th Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, Sep. 2013, pp. 331–338.

[39] F. Harary, *Graph Theory*. Boca Raton, FL, USA: CRC Press, 1994.

[40] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan," *Linear Matrix Inequalities in System and Control Theory*. Philadelphia, PA, USA: SIAM, 1994.

[41] L. El Ghaoui and S. Niculescu, *Advances in Linear Matrix Inequalities Methods in Control*. Philadelphia, PA, USA: SIAM, 2000.

[42] J. Bang-Jensen and G. Gutin, *Classes Directed Graphs*. Springer, 2018.

[43] I. S. Duff, K. Kaya, and B. Uçcar, "Design, implementation, and analysis of maximum transversal algorithms," *ACM Trans. Math. Softw.*, vol. 38, no. 2, pp. 1–31, 2011.

[44] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.

[45] M. Khanbaghi and A. I. Zecevic, "Jump linear quadratic control for microgrids with commercial load," *Energies*, vol. 13, no. 18, p. 4997, Oct. 2020.

[46] R. Akella and P. R. Kumar, "Optimal control of production rate in a failure prone manufacturing system," *IEEE Trans. Autom. Control*, vol. AP-31, no. 2, pp. 116–126, Feb. 1986.

[47] S. Chopra and P. Meindl, *Supply Chain Management*. London, U.K.: Pearson, 2012.

[48] K. G. Kempf, "Control-oriented approaches to supply chain management in semiconductor manufacturing," in *Proc. Amer. Control Conf.*, 2004, pp. 4563–4576.

[49] E. Perea, I. Grossmann, E. Ydstie, and T. Tahmassebi, "Dynamic modeling and classical control theory for supply chain management," *Comput. Chem. Eng.*, vol. 24, nos. 2–7, pp. 1143–1149, Jul. 2000.

[50] E. Perea-López, B. E. Ydstie, and I. E. Grossmann, "A model predictive control strategy for supply chain optimization," *Comput. Chem. Eng.*, vol. 27, nos. 8–9, pp. 1201–1218, Sep. 2003.
[51] E. Porteus, *Foundations of Stochastic Inventory Theory*. Stanford, CA, USA: Stanford Univ. Press, 2002.
[52] W. Wang and D. Rivera, "Model predictive control for tactical decision-making in semiconductor manufacturing supply chain management," *IEEE Trans. Control Syst. Technol.*, vol. 16, no. 5, pp. 841–855, Sep. 2008.
[53] E. K. Boukas, "Manufacturing systems: LMI approach," *IEEE Trans. Autom. Control*, vol. 51, no. 6, pp. 1014–1018, Jun. 2006.

ALEKSANDAR ZEČEVIĆ (Senior Member, IEEE) received the B.S. degree in electrical engineering from the University of Belgrade, Yugoslavia, in 1984, and the M.S. and Ph.D. degrees from Santa Clara University, Santa Clara, CA, USA, in 1990 and 1993, respectively. He is currently a Professor with the School of Engineering, Santa Clara University, where he teaches courses in the area of electric circuits, control theory, and nonlinear systems. He has published books and articles on a broad range of topics, including graph theoretic decomposition algorithms, control of large-scale systems, high-performance computing, and power systems.

MARYAM KHANBAGHI (Senior Member, IEEE) received the bachelor's degree from the Université de Nice-Sophia Antipolis, France, and the master's and Ph.D. degrees in electrical engineering from École Polytechnique, Montreal, Canada. She worked for several years at the Pulp and Paper Research Institute, Montreal and Vancouver, Canada, as a Research Engineer. She joined Corning Inc., in 2000, and during her 12 years tenure, she worked in research and development centers, corporate engineering, manufacturing, and commercial technology. She is currently an Assistant Professor at the Department of Electrical Engineering, Santa Clara University, Santa Clara, CA, USA, and the Director of power systems and sustainable energy graduate program. She has published over 30 papers in journals, conference proceedings, and company internal research papers. She has two patents. Her research interests include control system design/optimization for large systems, such as manufacturing and power systems. She was an Associate Editor of *IEEE Control System Magazine*.