influence of turning radius for the turning and welding process is analyzed. It is found that the result of the vehicle turning without any turning radius will be the best from the global perspective.

References

[1] Z. Fang, D. Xu, and M. Tan, "A vision-based self-tuning fuzzy controller for fillet weld seam tracking," *IEEE/ASME Trans. Mechatronics*, vol. 16, no. 3, pp. 540–550, Jun. 2011.

[2] M. D. Ngo, V. H. Duy, N. T. Phuong, and S. B. Kim, "Robust control of welding robot for tracking a rectangular welding line," *Int. J. Adv. Robot. Syst.*, vol. 3, no. 3, pp. 239–248, 2006.

[3] T. L. Chung, T. H. Bui, T. T. Nguyen, S. B. Kim, and M. S. Oh, "Two-wheeled welding mobile robot for tracking a smooth curved welding path using adaptive sliding-mode control technique," *Int. J. Control, Autom, Syst.*, vol. 5, no. 3, pp. 283–294, 2007.

[4] H. Chen, Y. Meng, and X. Wang, "Combined planning between welding pose and welding parameters for an arc welding robot," *Robot. Welding, Intell., Autom.*, vol. 88, pp. 73–80, 2011.

[5] D. Chang, D. Son, J. Lee, D. Lee, T. Kim, K. Y. Lee, and S. J. Kim, "A new seam-tracking algorithm through characteristic-point detection for a portable welding robot," *Robot. Comput.-Integr. Manuf.*, vol. 28, no. 1, pp. 1–13, 2011.

[6] Y. Xiao, Z. Du, and W. Dong, "Smooth and near time-optimal trajectory planning of industrial robots for online applications," *Industrial Robot: An Int. J.*, vol. 39, no. 2, pp. 169–177, 2012.

[7] Y. B. Jeon, S B Kim, and S. S. Park, "Modeling and motion control of mobile robot for lattice type welding," *KSME Int. J.*, vol. 16, no. 1, pp. 83–93, 2002.

[8] M. Dinham, G. Fang, and J. Zou, "Experiments on automatic seam detection for a MIG welding robot," *Artif. Intell. Comput. Intell.*, vol. 7003, pp. 390–391, 2011.

[9] Z. Liu, W. Bu, and J. Tan, "Motion navigation for arc welding robots based on feature mapping in a simulation environment," *Robot. Comput.-Integr. Manuf.*, vol. 26, no. 2, pp. 137–144, 2010.

[10] X. Z. Chen, Y. M. Huang, and S. B. Chen, "Model analysis and experimental technique on computing accuracy of seam spatial position information based on stereo vision for welding robot," *Ind. Robot: An Int. J.*, vol. 39, no. 4, pp. 349–356, 2012.

[11] S. B. Chen, X. Z. Chen, and J. Q. Li, "Acquisition of weld seam dimensional position information for arc welding robot based on vision computing," *J. Intell. Robot. Syst.*, vol. 43, no. 1, pp. 77–97, 2005.

[12] N. Chakraborty, S. Akella, and J. T. Wen, "Coverage of a planar point set with multiple robots subject to geometric constraints," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 1, pp. 111–122, Jan. 2010.

[13] M. D. Ngo, N. T. Phuong, V. H. Duy, H. K. Kim, and S. B. Kim, "Control of two wheeled welding mobile manipulator," *Int. J. Adv. Robot. Syst.*, vol. 4, no. 3, pp. 293–302, 2007.

[14] J. Kim, S. R. Kim, S. J. Kim, and D. H. Kim, "A practical approach for minimum-time trajectory planning for industrial robots," *Ind. Robot: An Int. J.*, vol. 37, no. 1, pp. 51–61, 2010.

[15] M. H. Wu, X. F. Gao, W. X. Yan, Z. Fu, Y. Z. Zhao, and S. B. Chen, "New mechanism to pass obstacles for magnetic climbing robots with high payload, using only one motor for force-changing and wheel-lifting," *Ind. Robot: An Int. J.*, vol. 38, no. 4, pp. 372–380, 2011.

Hybridization of Bound-and-Decompose and Mixed Integer Feasibility Checking to Measure Redundancy in Structured Linear Systems

Manish Bansal, Kiavash Kianfar, Yu Ding, *Member, IEEE*, and Erick Moreno-Centeno

***Abstract*—Computing the degree of redundancy for structured linear systems is proven to be NP-hard. A linear system whose model matrix is of size $n \times p$ is considered structured if some p row vectors in the model matrix are linearly dependent. Bound-and-decompose and 0-1 mixed integer programming (MIP) are two approaches to compute the degree of redundancy, which were previously proposed and compared in the literature. In this paper, first we present an enhanced version of the bound-and-decompose algorithm, which is substantially (up to 30 times) faster than the original version. We then present a novel hybrid algorithm to measure redundancy in structured linear systems. This algorithm uses a 0-1 mixed integer feasibility checking algorithm embedded within a bound-and-decompose framework. Our computational study indicates that this new hybrid approach significantly outperforms the existing algorithms as well as our enhanced version of bound-and-decompose in several instances. We also perform a computational study that shows matrix density has a significant effect on the runtime of the algorithms.**

***Note to Practitioners*—People have long realized the importance of having sensor or measurement redundancy in a system as this redundancy safeguards the system against sensor failures or measurement anomalies, so much so that the degree of redundancy is a reflection of the system's reliability or fault-tolerance capability. Because of dependence relationship among the system's components or subsystems, computing the degree of redundancy is not a straightforward matter for practical systems which embed certain structure. Our paper presents an enhanced version of an existing method as well as a novel hybrid algorithm to calculate degree of redundancy, which are significantly faster than the existing methods in many cases. These algorithms are a step forward in addressing this challenging problem.**

***Index Terms*—Degree of redundancy, mixed integer programming, bound-and-decompose, NP-hard, structured linear model.**

I. Introduction

This paper presents new algorithms for evaluating the degree of redundancy in linear systems, which are more efficient than existing methods in several cases. In the engineering literature, a linear model of the following format has been a popular choice for establishing connections between sensor measurements $\mathbf{y}$ and system states $\mathbf{x}$, through a system matrix $\mathbf{H}$:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{e} \tag{1}$$

where $\mathbf{y}$ and $\mathbf{e}$ are $n \times 1$ vectors, $\mathbf{x}$ is a $p \times 1$ vector, $\mathbf{H}$ is an $n \times p$ matrix ($n \geq p$) and $r(\mathbf{H}) = p$, where $r(\mathbf{H})$ denotes the rank of $\mathbf{H}$. The last term $\mathbf{e}$ is the residual term, including measurement noises as well as higher-order nonlinear effects neglected by the above model due to the linearization. Equation (1) is, in fact, the observation equation used

Manuscript received July 30, 2012; revised December 11, 2012; accepted April 03, 2013. Date of publication May 23, 2013; date of current version October 02, 2013. This paper was recommended for publication by Associate Editor T. Breti and Editor L. Shi upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant CMMI-1100343.

The authors are with the Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX 77843-3131 USA (e-mails: bansal@tamu.edu; kianfar@tamu.edu; yuding@iemail.tamu.edu; e.moreno@tamu.edu).

Digital Object Identifier 10.1109/TASE.2013.2259479

in a typical linear state-space model [1] and the Kalman filter [2]. In practice, we almost always have $r(\mathbf{H}) = p$. For this reason and for simplicity of the presentation, we assume $r(\mathbf{H}) = p$ throughout this paper. However, one should note that from a theoretical perspective, with slight modifications, all the algorithms in this paper can also be applied in the case where $r(\mathbf{H}) < p$.

In equation (1), the number of measurements, n, should be greater than the number of states, p; otherwise, one will run into an ill-posed system where there is no unique estimation of the states. Since there are more measurements than system states, loosely speaking, those beyond the smallest number of measurements necessary for uniquely estimating $\mathbf{x}$ are considered redundant. The degree of redundancy of the matrix $\mathbf{H}$, denoted by $d(\mathbf{H})$, is formally defined in the literature (see, for example, [3]) as follows:

$$d(\mathbf{H}) = \min\left\{\delta - 1 : \exists \mathbf{H}_{(-\delta)} \text{ s.t. } r\left(\mathbf{H}_{(-\delta)}\right) < p\right\} \qquad (2)$$

where $\mathbf{H}_{(-\delta)}$ is a reduced matrix after deleting δ rows from the original $\mathbf{H}$ matrix, and $r(\cdot)$ is the rank function. Based on (2), the redundancy degree of the linear system is the minimum number of sensor failures (or measurement outliers) which should happen before the identifiability of any state is compromised.

Researchers consider the measurement redundancy as an index of the system's reliability [3]–[5] and this explains the importance of quantitatively calculating it. When the system has no intrinsic structure, mathematically it means that any p row vectors of $\mathbf{H}$ are linearly independent and $d(\mathbf{H}) = n - p$. In this case, $\mathbf{H}$ is called unstructured. But, when there exists some intrinsic structure in the system, some p row vectors of $\mathbf{H}$ are linearly dependent and $\mathbf{H}$ is called structured. In this case, $d(\mathbf{H})$ is smaller than $n - p$ but evaluating it quantitatively is no longer a straightforward task. Vardy [21] proved that the minimum distance problem in binary coding, which can be reduced (in polynomial time) to the problem of finding $d(\mathbf{H})$ for a special structured matrix $\mathbf{H}$, is NP-hard. This implies that a polynomial-time algorithm for the problem of finding the degree of redundancy of a general structured matrix unlikely exists [9].

Structured linear systems are pervasive in engineering applications. Some instances reported in the literature include wireless sensor networks [6], distributed (wired) sensor systems in multistation assembly [7], and sensor systems in electric power networks [8]. Take the multistation assembly process in [7] for example: it is a three-station assembly process, where four parts are assembled in a sequential manner. There are a total of $n = 26$ sensor measurements distributed over the three stations, and their outputs constitute the vector $\mathbf{y}$. The process variables under monitoring are the deviations of fixture locators from their nominal positions. The fixture locators are the mechanical devices holding a part in space during an action of assembly, so their positioning precision affects the quality of the final assembly. Each pair of fixtures provide constraints in three degrees of freedom for a given part. So there are a total of 12 degrees of freedom associated with the four parts; they are included in the vector $\mathbf{x}$, and thus, $p = 12$. By a simple analysis using rigid body transformation, followed by linearization, a model in the format of (1) can be obtained; for details, please see [7].

On the surface, people could mistake the total degree of redundancy in this assembly system as $n - p = 26 - 12 = 14$. This turns out to be wrong. The resulting $\mathbf{H}$ matrix for this multistation assembly indeed embeds structure due to the particular sequence in which parts are transitioned and how sensor measurements are distributed over the three stations. The actual redundancy in this case is only 4, a value significantly smaller than 14. What this implies is that without knowing the precise degree of redundancy of the sensor system, one can be overly optimistic about the system's ability to tolerate potential sensor failures, and therefore face undue delays in terms of sensor repair or replacement. Moreover, in a statistical analysis, using $n-p$ as the degrees of freedom for noise variance estimation will underestimate the variance and falsely boost the power of statistical inference that the sensor system provides.

For this 26×12 example, computation is not much of a problem. But for an assembly process with, for instance, ten or more assembly stations, which is quite typical in industry, the size of matrix $\mathbf{H}$ can easily get to a few hundred, or even thousands, of rows and columns. Due to the relevance of structured model matrices in engineering systems, it is important to devise efficient methods to quantitatively evaluate the degree of redundancy for large-scale systems.

The simplest algorithm for finding the degree of redundancy is the exhaustive rank testing [3]. The idea is to increment δ by one at each iteration starting from $\delta = 1$, and for each δ find the rank of all possible submatrices $\mathbf{H}_{(-\delta)}$ until a submatrix whose rank is smaller than p is found. As discussed in [10], this algorithm is very inefficient, and therefore, is discarded in the rest of this paper. Cho *et al.* [7], [9] presented a bound-and-decompose algorithm to calculate the degree of redundancy, which uses results from matroid theory to decompose the problem into smaller problems. In this paper, we denote this algorithm by BDOLD. As we will discuss in Section II, solving the smaller problems in the BDOLD algorithm involves rank testing of certain submatrices of $\mathbf{H}$. In previous implementations of BDOLD in the literature [7], [10], this rank testing was performed using singular value decomposition (SVD) [12]. We will denote this implementation by BDOLDSVD. BDOLD is efficient for the problems in which $\mathbf{H}$ has a border block diagonal (BBD) form (see Section II) with reasonable block sizes and a narrow border (hence the algorithm can do significant decomposition), and is not so efficient otherwise (see Section II). Recently, Kianfar *et al.* [10] tackled the problem from a different angle. They formulated the problem as a 0-1 mixed integer program (MIP) and solved it using the branch-and-cut method embedded in the commercial optimization solver CPLEX [11]. They showed that in several instances MIP outperforms BDOLDSVD, especially in instances where BDOLDSVD cannot do significant decomposition. However, as the size of instance increases MIP reaches its performance limit. In fact, there are instances in [10] which are neither solved by BDOLDSVD nor by MIP within a 10-h time limit. Note that the MIP algorithm does not exploit BBD form of the matrix, which arises frequently in the real-world structured linear systems.

In this paper, we first present an enhanced version of the BDOLD algorithm, denoted by BDNEW. We also study the effect of using QR factorization to perform the submatrix rank-testings in BDOLD and BDNEW (the resulting new algorithms are denoted by BDOLDQR, BDNEWSVD, and BDNEWQR). We will see that BDNEWQR substantially outperforms all other versions. In particular, it is up to 30 times faster than BDOLDSVD. We next present a novel hybrid algorithm to calculate $d(\mathbf{H})$, denoted by BDMIF, which uses a 0-1 mixed integer feasibility (MIF) checking algorithm embedded within a bound-and-decompose framework. BDMIF capitalizes on benefits of both bound-and-decompose and MIP. Our computational study indicates that this new hybrid approach significantly outperforms MIP and BDNEWQR (and hence other versions of BDOLD and BDNEW) in several instances. We also report the results of a computational experiment that shows, as the density of the blocks and border rows of $\mathbf{H}$ increases, the runtimes of the algorithms considered in this paper substantially increase.

The remainder of this paper is organized as follows. Section II reviews the algorithms previously introduced in the literature, i.e., BDOLD and MIP. Section III introduces BDNEW and the usage of QR factorization for rank testing, which results in BDOLDQR, BDNEWSVD, and BDNEWQR algorithms. In Section IV, we introduce the BDMIF algorithm. Section V presents our computational studies followed by discussions on merits and limitations of the studied algorithms. We address our computational study on the effect of density in Section VI and make some concluding remarks in Section VII.

II. Related Work

We use the same notation used in [10]: for a $V \subseteq \{1,\dots,n\}$, $\mathbf{H}_V$ denotes the matrix obtained by removing from $\mathbf{H}$ all rows $\mathbf{h}_i, i \in \{1,\dots,n\}\backslash V$, and we define the operator I as $I(\mathbf{H}_V) = V$. The problem of evaluating the degree of redundancy $d(\mathbf{H})$ is to find the minimum integer value δ^* such that there exists a $V \subseteq \{1,\dots,n\}$, where $|V| = n - \delta^*$ and $r(\mathbf{H}_V) < p$. Then, $d(\mathbf{H}) = \delta^* - 1$.

If the system matrix has a significant BBD structure as it is the case in many practical applications, then the BDOLD algorithm introduced in [7] has substantial computational benefits. Using a transformation algorithm [7], [13], the rows and columns of the system matrix can be rearranged into the BBD form so that we have

$$\mathbf{H} = \begin{bmatrix} \mathbf{B}_1 & & & \\ & \mathbf{B}_2 & & \\ & & \ddots & \\ & & & \mathbf{B}_r \\ \mathbf{S}_1 & \mathbf{S}_2 & \cdots & \mathbf{S}_r \end{bmatrix}$$

where the block $\mathbf{B}_t$ is an $n_t \times p_t$ matrix, and $\mathbf{S}_t$ is an $n_S \times p_t$ matrix for $t = 1,\dots,r$. The $\mathbf{S}_t$ matrices form the border. We have $n_s + \sum_{t=1}^{r} n_t = n$ and $\sum_{t=1}^{r} p_t = p$. The unwritten elements are zero. Let $g^*(\mathbf{H}^T)$ denote the cogirth of the vector matroid defined over columns of the matrix $\mathbf{H}^T$, i.e., the cardinality of the smallest minimal dependent set in the dual of this vector matroid (please refer to [22] for more on the concepts of matroid and cogirth). Cho *et al.* [9] showed that $d(\mathbf{H}) = g^*(\mathbf{H}^T) - 1$, and used matroid theory to model the problem of finding the degree of redundancy as follows. Let U be a subset of $\{1,\dots,r\}$, and define $\mathbf{H}[U]$ as the reduced BBD matrix obtained by removing from $\mathbf{H}$ the submatrices $\mathbf{B}_t$ and $\mathbf{S}_t$ for $t \in \{1,\dots,r\}\backslash U$. Let $n_U, p_U, \mathbf{h}_i^U$, and h_{ij}^U denote the number of rows, the number of columns, the ith row, and the (i,j)th entry of $\mathbf{H}[U]$, respectively. The following decomposition property is proved in [9]: For any $k \in \{1,\dots,r\}$, if $g^*(\mathbf{H}^T) \geq (k+1)/(k)n_S - 1$, we have $g^*(\mathbf{H}^T) = \min\{g^*(\mathbf{H}[U]^T) : U \subseteq \{1,\dots,r\} \text{ and } |U| = k\}$ (see [9] for details). This result provides the basis of the decomposition in the BDOLD algorithm.

BDOLD Algorithm [7], [9], [10]

```
Step 0. $\delta \leftarrow 1$.
Step 1. If $\delta < (r)/(r-1)n_S - 1$, then $k^* \leftarrow r$ and go to Step 2;
    otherwise find $k^*$ that gives minimum number of matrices to be
    rank-tested based on the decomposition property:
    $k^* \leftarrow \operatorname{argmin}_{n_S/(\delta - n_S + 1) \leq k \leq r} \left[ \sum_{U \subseteq \{1,\dots,r\}; |U| = k} \binom{n_S + \sum_{t \in U} n_t}{\delta} \right]$.
Step 2. Check the rank of all the matrices $\mathbf{H}[U]_V$, where
    $U \subseteq \{1,\dots,r\}, |U| = k^*, V \subseteq I(\mathbf{H}[U])$, and $|V| = n - \delta$,
    one by one until a $\mathbf{H}[U]_V$ is found such that $r(\mathbf{H}[U]_V) < p_U$.
    If found (which means $g^*(\mathbf{H}^T) = \delta$), then $d(\mathbf{H}) \leftarrow \delta - 1$ and
    stop. Else $\delta \leftarrow \delta + 1$ and go to Step 1.
```

In the rest of this paper, we refer to the value $(r)/(r-1)n_S - 1$ as the "decomposition bound" because in the BDOLD algorithm as long as δ is smaller than this value, no decomposition occurs in Step 2 (we have $U = \{1,\dots,r\}$, hence $\mathbf{H}[U] = \mathbf{H}$). In the implementation of the BDOLD algorithm in [7], [10], the rank of $\mathbf{H}[U]_V$ in Step 2 is calculated using the *rank* function in MATLAB, which calculates the rank using SVD [12]. In this paper, we implemented the same algorithm in C++ and we denote it by BDOLDSVD.

More recently, Kianfar *et al.* [10] proposed the MIP approach for the redundancy degree problem. They formulated the problem as a 0-1 mixed integer program (please refer to [14] and [15] for more on mixed integer programming). This formulation is based on the fact that the redundancy degree problem can be solved by finding the minimum number of vectors that if deleted from $\mathbf{H}$, the remaining matrix has a nonzero null space. Note that the null space of matrix $\mathbf{H}$ is the set of all vectors $\mathbf{x}$ such that $\mathbf{Hx} = 0$. The redundancy degree would be one less than this minimum. The MIP formulation looks for a nonzero $\mathbf{x} \in \mathbb{R}^p$ such that the number of rows $\mathbf{h}_i$'s, $i = 1,\dots,n$, for which $\mathbf{h}_i\mathbf{x} \neq 0$ is minimized. The MIP formulation presented in [10] is as follows:

$$\min \quad \sum_{i=1}^{n} q_i \tag{3}$$

$$\text{subject to}$$

$$-q_i \leq \sum_{j=1}^{p} h_{ij}x_j \leq q_i \quad i = 1,\dots,n \tag{4}$$

$$-1 + 2z_j \leq x_j \leq 1 \quad j = 1,\dots,p \tag{5}$$

$$\sum_{j=1}^{p} z_j = 1 \tag{6}$$

$$x_j \in \mathbb{R}, q_i, z_j \in \{0,1\} \quad i = 1,\dots,n; j = 1,\dots,p. \tag{7}$$

Based on constraints (4), if $\mathbf{h}_i\mathbf{x} \neq 0$ for any i, then the 0-1 variable q_i will get a value of 1 and if $\mathbf{h}_i\mathbf{x} = 0$ it will get a value of zero because the objective is to minimize the summation of all q_i's. In other words, to get a nonzero null space, if $q_i = 1$, the row $\mathbf{h}_i$ is to be deleted from $\mathbf{H}$, and if $q_i = 0$, the row $\mathbf{h}_i$ is to remain. Therefore, objective (3) minimizes the total number of vectors out of all $\mathbf{h}_i, i = 1,\dots,n$ which if deleted from $\mathbf{H}$, result in a matrix with a nonzero null space. For this reason, $d(\mathbf{H})$ will be the optimal objective value minus 1. To guarantee that there is at least one $\mathbf{x}$ that satisfies constraint (4), it is assumed that all the row vectors of $\mathbf{H}$ are scaled such that $\|\mathbf{h}_i\|_1 = \sum_{j=1}^{p} |h_{ij}| = 1$. In other words, if $\|\mathbf{h}_i\|_1 \neq 1$, then $\mathbf{h}_i \leftarrow \mathbf{h}_i/\|\mathbf{h}_i\|_1$. Constraints (5) and (6) enforce $\mathbf{x}$ to be nonzero (please refer to [10] for more details).

We observe that the MIP approach does not utilize any special structure in the matrix $\mathbf{H}$. In [10], the MIP formulation (3)–(7) was implemented in AMPL [23] and the commercial solver CPLEX [11] was used to solve it for the studied instances. It was shown in [10] that in several instances, where the decomposition bound is large, MIP significantly outperforms BDOLD.

III. Faster Versions of Bound-and-Decompose Algorithm

The most time-consuming operation in the BDOLD algorithm is in its Step 2, where each time a matrix $\mathbf{H}[U]_V$ is formed, its rank should be calculated and compared to p_U. Each run of the algorithm includes calculating the rank of a large number of submatrices $\mathbf{H}[U]_V$ and these rank calculations are the time bottleneck of the BDOLD algorithm.

BDNEW, our new version of the bound-and-decompose algorithm, differs from BDOLD in the value chosen for k^* in Step 1. In Step 1 of BDOLD, the value of k^* in the case where δ is greater than or equal to the decomposition bound is calculated such that the number of rank testings resulting from the decomposition is minimized (i.e., the number of submatrices $\mathbf{H}[U]_V$ is minimized). Notice that the total run time of Step 2 for each δ depends on not only the number of $\mathbf{H}[U]_V$ matrices, but also the size of these matrices. Our computational experiments showed that choosing $k^* = \lceil n_S/(\delta - n_S + 1) \rceil$, which minimizes the size of $\mathbf{H}[U]_V$ matrices, instead of the value chosen in Step 1 of BDOLD, which minimizes the number of $\mathbf{H}[U]_V$ matrices, reduces the total time taken by Step 2. Therefore, $k^* = \lceil n_S/(\delta - n_S + 1)\rceil$ is the value we use in BDNEW.

BDNEW Algorithm

```
Step 0. $\delta \leftarrow 1$.
Step 1. If $\delta < (r)/(r-1)n_S - 1$, then $k^* \leftarrow r$ and go to Step 2;
    otherwise $k^* \leftarrow \lceil n_S/(\delta - n_S + 1)\rceil$ in order to have the
    smallest submatrices in Step 2.
Step 2. Same as Step 2 of BDOLD.
```

Another issue of concern at Step 2 of both BDOLD and BDNEW is how the ranks of submatrices $\mathbf{H}[U]_V$ are calculated. As mentioned in Section II, in the previous implementation of BDOLD in the literature [7], [10], the rank calculation is performed using SVD. In this paper, we employ a substantially more efficient method for rank calculation. This method is based on the QR factorization of the matrix. In principle, the QR factorization (or QR decomposition) of a matrix is expressing that matrix as the product $\mathbf{QR}$ of a matrix $\mathbf{Q}$ whose columns are orthogonal unit vectors and an upper triangular matrix $\mathbf{R}$ [12], [17]. In [18] and [19], it has been shown that for any given $s \times t$ matrix $\mathbf{A}$ (where $s \geq t$), there exists a $t \times t$ permutation matrix $\mathbf{\Pi}$ such that the QR factorization $\mathbf{A\Pi} = \mathbf{QR}$ exhibits the rank of $\mathbf{A}$ in the $t \times t$ upper triangular matrix $\mathbf{R}$. Such factorization, which can be used to determine the rank of $\mathbf{A}$, is called rank-revealing QR factorization (RRQR factorization). The way the rank of $\mathbf{A}$ is exhibited in $\mathbf{R}$ is as follows: The matrix $\mathbf{R}$ resulting from the RRQR factorization of $\mathbf{A}$ will have a structure like

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{R}_{22} \end{bmatrix}$$

where $\mathbf{R}_{22}$ is a $m \times m$ matrix, $\mathbf{R}_{22} = \mathbf{0}$, and $\mathbf{R}_{11}$ is nonsingular. Then, we will have $r(\mathbf{A}) = t - m$ (note that m may be zero in which case $r(\mathbf{A}) = t$). In practice, due to numerical issues, typically $\mathbf{R}_{22}$ will not be exactly a zero matrix. Instead, one would look for a matrix $\mathbf{R}_{22}$ such that $\mathbf{R}_{22} \simeq \mathbf{0}$, i.e., $\|\mathbf{R}_{22}\|_2 \leq \epsilon$ for a small enough positive ϵ ($\|\mathbf{R}_{22}\|_2$ is the Euclidean norm of $\mathbf{R}_{22}$, which is equal to its maximum singular value) [12], [16].

Theoretically, although the flop (floating point operation) count to calculate the rank of a $s \times t$ matrix (where $s \geq t$) using either SVD or QR factorization is $O(st^2)$, the number of flops in the SVD based rank calculation is approximately twice the number of flops in the QR based algorithm [12]. Moreover, both QR and SVD are numerically quite reliable [12].

In Section V, we will perform computational experiments to analyze the performance of both BDOLD and BDNEW algorithms with both SVD- and QR-based rank calculation. We will see that for all instances, BDNEWSVD and BDNEWQR outperform BDOLDSVD and BDOLDQR, respectively. Moreover, BDOLDQR and BDNEWQR are significantly (up to 11 times) faster than BDOLDSVD and BDNEWSVD, respectively. As a result, BDNEWQR is always the fastest among the four.

IV. BDMIF: A Hybrid Algorithm

In Step 2 of the BDNEW algorithm, for each δ and $U \subseteq \{1,\ldots,r\}$, where $|U| = k^*$, one solves the problem of whether $d(\mathbf{H}[U]) = \delta - 1$ or not. The way this problem is solved in BDNEW is by an exhaustive rank testing, i.e., the rank of all possible reduced matrices $\mathbf{H}[U]_V$, which are obtained by removing δ rows from $\mathbf{H}[U]$, are calculated. As exhaustive rank testing is an inefficient method, its repeated use on all matrices $\mathbf{H}[U]_V$ is a major drawback of the BDNEW algorithm (obviously BDOLD has the same drawback too).

In this section, we propose a new algorithm (BDMIF) which replaces the exhaustive rank testing of each submatrix $\mathbf{H}[U]$ with solving a mixed integer feasibility (MIF) problem. A MIF problem is the problem of finding out whether the mixed integer set defined by a collection of linear constraints contains a feasible point or not. Our BDMIF algorithm is a hybrid algorithm in which MIF problems are repeatedly solved within a bound-and-decompose framework. As a result, BDMIF makes it possible to simultaneously exploit both the decomposable structure of a BBD matrix $\mathbf{H}$ and the superiority of MIP over the exhaustive rank testing.

BDMIF Algorithm

```
Step 0. $\delta \leftarrow 1$.
Step 1. If $\delta < (r)/(r-1)n_S - 1$, then $k^* \leftarrow r$ and go to Step 2;
    otherwise $k^* \leftarrow \lceil n_S/(\delta - n_S + 1)\rceil$ in order to have a minimum
    size MIF problem in Step 2.
Step 2. Do the following for all the matrices $\mathbf{H}[U]$,
    where $U \subseteq \{1,\ldots,r\}$ and $|U| = k^*$, one by one:
    • normalize all the row vectors, i.e., $\mathbf{h}_i^U \leftarrow \mathbf{h}_i^U/\|\mathbf{h}_i^U\|_1$;
    • solve the MIF problem defined as follows:
```

$$\sum_{i=1}^{n_U} q_i = \delta \tag{8}$$

$$-q_i \leq \sum_{j=1}^{p_U} h_{ij}^U x_j \leq q_i \quad i = 1,\ldots,n_U \tag{9}$$

$$-1 + 2z_j \leq x_j \leq 1 \quad j = 1,\ldots,p_U \tag{10}$$

$$\sum_{j=1}^{p_U} z_j = 1 \tag{11}$$

$$x_j \in \mathbb{R}, q_i, z_j \in \{0,1\} \quad i = 1,\ldots,n_U; j = 1,\ldots,p_U \tag{12}$$

```
    until a $\mathbf{H}[U]$ is found for which the MIF (8)–(12) has a feasible
    solution. If found (which means $g^*(\mathbf{H}^T) = \delta$), then $d(\mathbf{H}) \leftarrow \delta - 1$
    and stop. Else $\delta \leftarrow \delta + 1$ and go to Step 1.
```

It is observed that in Step 2 of BDMIF algorithm, the MIF problem (8)–(12) defined for $\mathbf{H}[U]$ is solved to see whether $d(\mathbf{H}[U]) = \delta - 1$ or not. Constraints (9)–(12) of this MIF are the same as constraints (4)–(7) of the MIP model written for the matrix $\mathbf{H}[U]$ instead of $\mathbf{H}$ and their role is the same as explained in Section II and [10]. Constraint (8) forces the total number of q_i variables that have nonzero values to be equal to δ. Based on constraints (9)–(12), this means that if the MIF problem (8)–(12) has a feasible solution, then there exists a nonzero vector $\boldsymbol{x} \in \mathbb{R}^{p_U}$ such that $\mathbf{h}_i^U\mathbf{x} = 0$ for $n_U - \delta$ vectors out of all $\mathbf{h}_i^U, i = 1,\ldots,n_U$. This implies that $d(\mathbf{H}[U]) \leq \delta - 1$. On the other hand, $d(\mathbf{H}[U]) \geq \delta - 1$ because $d(\mathbf{H}[U]) \geq d(\mathbf{H})$, and at iteration δ of BDMIF, we already know that $d(\mathbf{H}) \geq \delta - 1$. Therefore, $d(\mathbf{H}[U]) = \delta - 1$.

We solve the MIF problem (8)–(12) using CPLEX solver [11]. Since CPLEX only solves MIP problems, i.e., it needs an objective function to optimize, we use a constant objective function such as "$\max 0$" along with constraints (8)–(12) to solve the MIF. Clearly, the MIP solver stops after finding the first feasible solution to (8)–(12).

The choice of k^* in Step 1 of BDMIF is similar to that in Step 1 of BDNEW. This k^* makes the size of the MIF problems in Step 2 as small as possible (note that the number of all variables and the number of constraints (9) and (10) depends on k^*). The smaller the MIF problem is, the faster it is solved using CPLEX. Of course, the smallest k^* may not result in the smallest number of U sets (hence MIFs), however, in our computational experiments we found that it is

TABLE I
CHARACTERISTICS OF INSTANCE CATEGORIES AND AVERAGE $d(\mathbf{H})$ OF INSTANCES IN EACH CATEGORY

	Model Matrix **H**				
No.	$n \times p$	n_s	r	$bound^{\#}$	Average $d(\mathbf{H})$
1	26 × 12	2	4	1.67	4
2	66 × 27	3	9	2.37	7
3	154 × 72	2	2	3	5
4	316 × 144	8	4	9.67	3.8
5	485 × 360	5	8	4.7	1.6
6	222 × 55	2	11	1.2	14
7	1009 × 252	1	42	0.025	17
8	2018 × 504	2	84	1.024	18
9	501 × 384	9	4	11	$\geq 1, \leq 34$

#: The decomposition bound $\frac{r}{r-1}n_s - 1$.

always better to minimize the size of MIFs rather than the number of them.

V. Computational Results and Discussion

In this section, we present the results of our computational experiment to compare the performance of our proposed algorithms (BDOLDQR, BDNEWSVD, BDNEWQR, and BDMIF) with the existing algorithms (BDOLDSVD and MIP).

To make a fair comparison between the algorithms, we used a common implementation platform. We implemented all the existing and new algorithms in C++. We used *dgesdd* and *dgeqp3* routines of CLAPACK 3.2.1 to perform the SVD and RRQR factorizations, respectively, in BDOLDSVD, BDNEWSVD, BDOLDQR, and BDNEWQR. CLAPACK is the implementation of the well-known subroutine library LAPACK (Linear Algebra PACKage) in C language and has one of the most efficient implementations of SVD and RRQR factorization (see [20] for details). The MIP formulation was solved in C++ using the CPLEX 12.0 C++ concert technology [11] (which is a C++ library to model and solve optimization problems including integer programming). The BDMIF algorithm was also implemented in C++ and CPLEX 12.0 C++ concert technology was used to solve the MIF problems within the algorithm.

The results of our computational experiments are shown in Tables I–III. Each row of these tables corresponds to an instance category. Nine instance categories, each containing five random instances with similar structural characteristics were generated. The randomness is in the values of the matrix entries. The characteristics (size, number of blocks, and number of border rows) of each category are shown in Table I. The characteristics of instance categories 1 and 2 are based on the multistation assembly applications reported in [9]. For instance categories 3 to 7 and 9, the characteristics are inspired by the instances in [10]. The instance category 8 is of a much larger size than the instances in [10]. All instances are available at http://ise.tamu.edu/people/faculty/kianfar/dol/RedundancyDegree-Instances.zip. For each instance category, Table I also shows the decomposition bound for instances in the category. The last column of Table I is obtained after completing the algorithm runs and shows the average of $d(\mathbf{H})$ values of the five instances in each category.

Instances in category 1 correspond to the assembly example that we explained in Section I. Using the existing methods, including the exhaustive rank testing (doable for this small sized problem), the degree of redundancy in these instances can be determined as 4, which, as we mentioned earlier, is strikingly different from the rough estimate, had someone used $n - p$. Our new algorithms of course produce the same

TABLE II
RUNTIMES OF ALGORITHMS FOR INSTANCE CATEGORIES

	Run Time			
No.	BDOLDSVD	BDNEWSVD	BDOLDQR	BDNEWQR
1	0.09 sec.	0.04 sec.	0.02 sec.	**0.01 sec.**
2	1.6 min.	23.52 sec.	10.42 sec.	**3.13 sec.**
3	26.9 min.	26.5 min.	3.8 min.	**3.7 min.**
4	> 10 hours	> 10 hours	> 10 hours	> 10 hours
5	2.65 hours	2.63 hours	13.6 min.	**13.4 min.**
6	15.5 min.	15.4 min	6.5 min.	**4.62 min.**
7	> 10 hours	> 10 hours	2.9 hours	**2.9 hours**
8	> 10 hours	> 10 hours	> 10 hours	> 10 hours
9	> 10 hours	> 10 hours	> 10 hours	> 10 hours

TABLE III
RUNTIMES OF ALGORITHMS FOR INSTANCE CATEGORIES

	Run Time		
No.	BDNEWQR	MIP	BDMIF
1	**0.01 sec.**	0.12 sec.	4.55 sec.
2	**3.13 sec.**	21.28 sec.	48.35 sec.
3	3.7 min.	**3.60 sec**.	22.44 sec.
4	> 10 hours	**55.46 sec.**	3.7 min.
5	13.4 min.	**41.87 sec.**	6.8 min.
6	4.62 min.	> 10 hours	**2.1 min.**
7	2.9 hours	> 10 hours	**31.2 min.**
8	> 10 hours	> 10 hours	**1.7 hours**
9	> 10 hours	> 10 hours	> 10 hours

redundancy value. As will be mentioned shortly, our BDNEWQR algorithm is the fastest among all in calculating the degree of redundancy of instances in this category because they are of a small size.

For each instance category, Tables II and III show the average solution time of each algorithm over the five instances in the category. The bold numbers indicate the fastest algorithm in each category. All the algorithms were run on a PC which has two Intel Xeon E5620 2.40 GHz processors and 12 GB of RAM. A time limit of 10 h was imposed on the running time of each instance. The cells containing ">10 hours" mean that the run time of the algorithm for all five instances in the category reached 10 h without finding the degree of redundancy. All instances except those in category 9 are completely solved within 10 h by at least one of the algorithms. Therefore, the exact values of $d(\mathbf{H})$ for all instances in each of categories 1 to 8 are obtained and the average $d(\mathbf{H})$ of instances in each category is reported in Table I. None of the four algorithms was able to find $d(\mathbf{H})$ for any of the five instances in category 9 within 10 h. As a result only upper and lower bounds are reported for them which will be explained in more detail at the end of this section.

Based on Tables II and III, we make the following observations regarding the efficiency of our new algorithms versus the existing algorithms. First of all, in Table II, we observe that for all instance categories our BDOLDQR and BDNEWQR are significantly (up to 11 times) faster than BDOLDSVD and BDNEWSVD, respectively. This signifies the superiority of using QR factorization over SVD for rank calculation at Step 2. We also observe that for all instance categories BDNEWQR and BDNEWSVD are faster than BDOLDQR and BDOLDSVD, respectively. As a result, BDNEWQR is always fastest among the four versions of the BD algorithm studied here. For example, in instance category 2, BDNEWQR is 30, 10, and 3 times faster than BDOLDSVD, BDNEWSVD, and BDOLDQR, respectively.

Moreover, going to Table III we observe that except for very small instances (categories 1 and 2), BDMIF is consistently faster than BDNEWQR. BDNEWQR's performance is better than that of BDMIF on small instances because when submatrices $\mathbf{H}[U]$ are very small, rank testing with QR factorization is faster than solving the MIF problem in BDMIF using CPLEX. But as the submatrices get slightly larger, MIF performs much faster than rank testing. Since the solution times for small instances are less than a minute anyway, using BDMIF instead of BDNEWQR in general is a sound recommendation.

When comparing MIP with BDMIF we see that in instances where the decomposition bound is small, i.e., when the BDMIF algorithm can do significant decomposition (instances 6 to 8), BDMIF significantly outperforms MIP (e.g., for instance category 6, BDMIF is more than 285 times faster than MIP). It is interesting to see that our BDMIF algorithm is the only algorithm out of the four algorithms that can solve the large instances in category 8 within the 10-h time limit (the average BDMIF runtime on this category in only 1.7 h). However, when the decomposition bound is large (instances 3 to 5), MIP performs better than BDMIF and BDNEWQR because for such instances the latter two algorithms cannot perform significant decomposition. This is the same observation made in [10] while comparing BDOLD and MIP.

None of the algorithms are able to solve any instance in category 9 within 10 h. On one hand, these instances have large decomposition bound for BDMIF to be effective, and on the other hand, they are too large for MIP to be effective. It is important to note that although the algorithms cannot solve the problems, they provide upper and lower bounds on the degree of redundancy. MIP provided upper bounds of either 33 or 34 for the five different instances in category 9. This upper bound is the value of objective function (3) for the best integer solution found in 10 h by the MIP algorithm. However, the lower bound obtained by MIP for all instances in category 9 was the trivial bound 1. BDMIF can only provide lower bound. This lower bound was 2 (better than the MIP lower bound) for one of the instances in category 9 and the trivial 1 for the other four instances.

VI. Effect of Density

During our study we observed that the density (percentage of nonzero entries) of blocks and border rows of the instance has a major effect on the runtime of all algorithms. In this section we present the results of a computational experiment performed in order to acquire insight into the effect of density on the runtimes of BDNEWQR, BDMIF, and MIP (we discard BDOLDSVD, BDNEWSVD, and BDOLDQR as their runtimes are dominated by that of BDNEWQR). All our instances in this experiment have the structural characteristics of category 4 in Table I, i.e., they are 316×144 instances containing four equally sized blocks and eight border rows.

We considered various levels of block density and border density, as shown in Tables IV–VI. For each combination of block density and border density, we generated random instances with those block and border densities. Tables IV, V, and VI show the average runtime of BDNEWQR, MIP, and BDMIF, respectively, over the five instances in each combination. It can be seen that as the border density or block density increases, the runtimes of all three algorithms increase substantially. We point out that although here we only show the results for matrices with structural characteristic of category 4, our computations on matrices with other structures show a similar pattern regarding the effect of density.

As a side note, observe that based on Tables IV–VI, the MIP algorithm does better than BDMIF, and BDMIF does better than BDNEWQR on this instance structure as it was the case for category 4 in Table III (the instances in category 4 of Tables I–III had block density of 10% and border density of 40%).

TABLE IV
Effect of Density on Runtime of BDNEWQR

Computation Time (sec.)						
Block Density (%)	Border Density (%)					
	10	20	40	60	80	100
5	0.15	0.17	2149.2	13482	> 36000	
10	132.73	5713.8	> 36000			
> 15	> 36000					

TABLE V
Effect of Density on Runtime of MIP

Computation Time (sec.)						
Block Density (%)	Border Density (%)					
	10	20	40	60	80	100
5	0.08	0.09	1.17	3.27	2304.3	11892
10	0.99	1.54	55.4	159.26	18109	> 36000
15	45.47	61.22	256.38	1161.2	> 36000	
20	851.19	1269.7	30385	> 36000		
25	21957	> 36000				
> 30	> 36000					

TABLE VI
Effect of Density on Runtime of BDMIF

Computation Time (sec.)						
Block Density (%)	Border Density (%)					
	10	20	40	60	80	100
5	0.43	0.44	8.65	41.47	14656	> 36000
10	8.52	10.98	220.3	1379.5	> 36000	
15	117.3	2445.9	11455	> 36000		
> 20	> 36000					

VII. Concluding Remarks

We presented three enhanced versions of the BD algorithm for calculating the degree of redundancy of a matrix, of which BDNEWQR was shown to be the most efficient, being up to 30 times faster than the existing version of BD algorithm (BDOLDSVD). We also proposed the new hybrid algorithm BDMIF for calculating the degree of redundancy of a matrix. BDNEWQR benefits from decomposing the matrix into submatrices which are smaller than those in BDOLDSVD and also using the QR factorization in calculating the rank of these submatrices. BDMIF exploits solution of mixed integer feasibility problems within a bound-and-decompose framework. We conclude with the following remarks.

a) The BDNEWQR algorithm in all cases is significantly faster than BDOLDSVD presented in [7], [10] and BDMIF outperforms BDNEWQR except for very small instances, which have very short runtimes anyway. However, BDMIF is slower than MIP in instances where the decomposition bound is not small.
b) The success of BDMIF is because it combines the power of BDNEW in exploiting the BBD form of the matrix and the power of MIF checking to detect the degree of redundancy of smaller submatrices. Even when BDMIF is not able to completely solve the problem in a given time, it provides a lower bound on the degree of redundancy.
c) Nevertheless, if the instance size and the decomposition bound are large, neither BDMIF nor MIP can solve the instance efficiently. As a result, although BDMIF is a step forward, the challenge of solving large instances with large decomposition bound remains.

d) Our computational study on the effect of density suggests that spending reasonable effort to transform a given matrix $\boldsymbol{H}$ to the sparsest possible form may have a large effect on reducing the runtime of the algorithms used for finding the degree of redundancy. This is a valuable direction for future research.

References

[1] W. J. Rugh, *Linear System Theory*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[2] M. S. Grewal and A. P. Andrews, *Kalman Filtering: Theory and Practice*. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

[3] M. Staroswiecki, G. Hoblos, and A. Aitouche, "Sensor network design for fault tolerant estimation," *Int. J. Adaptive Control Signal Process.*, vol. 18, pp. 55–72, 2004.

[4] Y. Ali and S. Narasimhan, "Redundant sensor network design for linear processes," *AIChE J.*, vol. 41, pp. 2237–2249, 1995.

[5] J. Levine and R. Marino, "On fault-tolerant observers," *IEEE Trans. Automatic Control*, vol. 35, no. 5, pp. 623–627, May 1990.

[6] J. J. Cho, Y. Ding, Y. Chen, and J. Tang, "Robust calibration for localization in clustered wireless sensor networks," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 1, pp. 81–95, Jan 2010.

[7] J. J. Cho, Y. Chen, and Y. Ding, "Calculating the breakdown point condition of sparse linear models," *Technometrics*, vol. 51, pp. 34–46, 2009.

[8] L. Mili, M. G. Cheniae, and P. J. Rousseeuw, "Robust state estimation of electric power systems," *IEEE Trans. Circuits. Syst.*, vol. 41, no. 5, pp. 349–358, May 1994.

[9] J. J. Cho, Y. Chen, and Y. Ding, "On the (co)girth of connected matroids," *Discrete Appl. Math.*, vol. 155, pp. 2456–2470, 2007.

[10] K. Kianfar, A. Pourhabib, and Y. Ding, "An integer programming approach for analyzing the measurement redundancy in structured linear systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 2, pp. 447–450, Apr. 2011.

[11] IBM ILOG CPLEX 12 Documentation [Online]. Available: http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r2/index.jsp available at:

[12] G. H. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, MD, USA: Johns Hopkins, 1996.

[13] C. Aykanat, A. Pinar, and U. V. Catalyurek, "Permuting sparse rectangular matrices into block-diagonal form," *SIAM J. Scientific Comput.*, vol. 25, pp. 1860–1879, 2004.

[14] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley, 1988.

[15] L. A. Wolsey, *Integer Programming*. New York, NY, USA: Wiley, 1998.

[16] G. H. Golub, V. C. Klema, and G. W. Stewart, "Rank degeneracy and least squares problems," Dept. Comput. Sci,, Stanford, Univ., Stanford, CA, USA, Tech. Rep. STAN-CS76-559, Aug. 1976.

[17] K. M. Hoffman and R. Kunze, *Linear Algebra*, 2nd ed. Englewood Cliffs, NJ, USA: Prentice-Hall, 2002.

[18] T. F. Chan, "Rank revealing QR factorizations," *Linear Algebra and Its Applications*, vol. 88–89, pp. 67–82, 1987.

[19] M. Gu and S. C. Eisenstat, "Efficient algorithms for computing a strong rank-revealing QR factorization," *SIAM J. Scientific Comput.*, vol. 17, no. 4, pp. 848–869, 1996.

[20] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999.

[21] A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Trans. Inf. Theory*, vol. 43, no. 6, pp. 1757–1766, Nov. 1997.

[22] J. Oxley, *Matroid Theory*. London, U.K.: Oxford Univ. Press, 1992.

[23] R. Fourer, D. M. Gay, and B. W. Kernighan, AMPL: A Modelling Language for Mathematical Programming. Duxbury, CA, USA, 2003.

On Deciding the Existence of a Liveness Enforcing Supervisory Policy in a Class of Partially Controlled General Free-Choice Petri Nets

N. Somnath and R. S. Sreenivas, *Senior Member, IEEE*

***Abstract*—If there are transitions in a Petri net (PN) that cannot be prevented from firing by a supervisory policy, then we have a *partially controlled* PN. The existence of a *liveness enforcing supervisory policy* (LESP) in a partially controlled *ordinary* PN is undecidable. Consequently, there can be no algorithms that synthesize an LESP for an arbitrary ordinary (general) PN. In contrast, we identify a class of general *free-choice* PN (FCPN) structures, which strictly includes the class of ordinary FCPN structures, where the existence of an LESP in any marked member of the class is decidable.**

***Note to Practitioners*—Every computer user has encountered situations where an unresponsive program enters into a state of suspended animation for perpetuity. A scheduling policy that can guarantee livelock-freedom is highly desirable in this, and other instances of concurrent systems. The results of this paper show that if the concurrent system is modeled as a Petri net that belongs to the class identified in this paper, then it is possible to determine if there is a policy that avoids livelocks, which paves the way for algorithm-development for livelock-avoidance.**

***Index Terms*—Petri nets (PNs), supervisory control.**

I. Introduction

A *Petri net* (PN) where are all arc weights are unitary is an *ordinary* PN, and a PN without restrictions on the arc weights is a *general* PN (see [1, Sec. 5.3]). A PN is a *free-choice Petri net* (FCPN) if each arc from a place to a transition is either the unique output arc of the place or is the unique input arc to the transition. Applications of FCPNs include the modeling of product-flow in manufacturing environments [2] and flow of control in processor networks [3].

A PN is *live* if, irrespective of the past transition firings, every transition in the PN can fire at some point in the future. A system modeled by a live PN does not experience *livelocks*. A PN model that is not live can be made live with the help of a supervisory policy that prevents the firing of a select group of transitions at each *marking*. This paper is about *liveness enforcing supervisory policies* (LESPs) for general FCPNs.

We identify a class of general FCPN structures $\mathcal{F}$ where, for any member in $\mathcal{F}$, the set of initial markings for which there is an LESP is *right-closed* [4]. If a marking is present in a right-closed set, then all markings that are larger than it are also present in the set. The class $\mathcal{F}$ strictly includes ordinary FCPN structures. As a consequence of the results in [5], it follows that the existence of a supervisory policy that enforces liveness in the class $\mathcal{F}$ is decidable.

Each member of $\mathcal{F}$ is identified by the following property—if a place has multiple output transitions, at least one of which is uncontrollable, then the weight associated with the arc that originates from the place to an uncontrollable transition, must be the smallest of all outgoing arc weights from the place, that is, if any output transition of this place is

Manuscript received August 09, 2012; revised October 16, 2012; accepted October 22, 2012. Date of publication January 17, 2013; date of current version October 02, 2013. This paper was recommended for publication by Associate Editor M. Dotoli and Editor M. C. Zhou upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant CNS-0834409.

The authors are with Industrial and Enterprise Systems Engineering Department, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA.

Digital Object Identifier 10.1109/TASE.2012.2227961