Neural Computation, vol. 10, No 1.

The Canonical Form of Nonlinear Discrete-Time Models

Gérard DREYFUS and Yizhak IDAN
ESPCI, Laboratoire d'Électronique
10, rue Vauquelin, 75005 Paris, France
dreyfus@neurones.espci.fr

Abstract

Discrete-time models of complex nonlinear processes, whether physical, biological or economical, are usually under the form of systems of coupled difference equations. In order to analyze such systems, one of the first tasks is that of finding a state-space description of the process, i.e. a set of state variables and the associated state equations. We present a methodology for finding a set of state variables and a canonical representation of a class of systems described by a set of recurrent discrete-time, time-invariant equations. In the field of neural networks, this is of special importance since the application of standard training algorithms requires the network to be in a canonical form. Several illustrative examples are presented.

1 Introduction

Black-box modeling, whether linear or nonlinear, is a highly valuable engineering tool, whose limitations are well known: the training data is usually corrupted with random noise, or with deterministic but unmeasured disturbances, it may not be representative of the whole range of operation of the process, etc. The bayesian approach, imposing constraints on the parameters based on prior knowledge, is one way to circumvent some of these problems; alternatively, when confronted with the task of modeling a complex process, one may take advantage of the wealth of knowledge that usually exists on the process, or on part of it, under the form of mathematical equations derived from physics (or from chemistry, biology, economy, etc.); these equations may be approximate or incomplete, hence may not meet the accuracy requirements of the application under consideration, but they are nevertheless useful for describing the deterministic behavior of the process. *Semi-physical modeling* is the approach whereby a first model, usually containing unknown parameters, is designed from prior mathematical knowledge, is complemented with "black boxes" wherever necessary, and whereby the unknown parameters are finally determined from measurements as in black-box modeling (Lindskog 1994). This technique has been successfully applied to the modeling of a

complex industrial process (Ploix et al. 1994, Ploix et al. 1996). One of the problems of this approach is the following: the discretized equations of the physical model are a set of coupled nonlinear difference equations which are not, in general, in the form of a state-space model. Handling such a model usually requires finding a set of state variables and deriving the corresponding discrete-time state equations.

The black-box capabilities of discrete-time recurrent neural nets for modeling dynamic processes have been extensively investigated (see for instance Zbikowski et al. 1995); in most cases, the dynamic models used are input-output models, consisting of a feedforward net whose output is fed back to the input with one or several unit delays. Such an architecture can readily be trained by backpropagation through time. However, in the case of semi-physical modeling, the equations of physics may suggest much more complicated architecture, with feedback within the neural network itself for instance (various such architectures will be considered in the paper). In that case, the alternative is the following: either derive and implement an *ad hoc* training algorithm for the specific architecture used to solve the specific modeling problem, or put the network into an equivalent canonical form (introduced in Nerrand et al. 1993) which can be trained by backpropagation through time; in this sense, backpropagation is *generic*, in that it can be applied to *any* neural network, whether recurrent or feedforward, however complex, *provided its canonical form has been derived*.

In the present paper, we show that, given a set of discrete-time, time-invariant difference equations of the form (1) shown below, it is possible to find automatically a set of state variables and the corresponding canonical form; the proposed procedure is based on a sequence of transformations and simple computations on a graph representation of the model, reminiscent of the flow graph technique used in linear systems theory (linear control, electronics, ...). In nonlinear modeling, bond-graph techniques (Thoma 1991) have been extensively used for deriving mathematical models from physical equations; the use of bond graphs may be viewed as a "preprocessing": it may provide a set of difference equations which can subsequently be handled as described in this paper in order to derive an appropriate canonical form.

The paper is organized as follows: in section 2, we present the problem in detail and provide definitions which will be used throughout the paper. Section 3 describes the various steps of the derivation of the order of the network, and subsequently of the derivation of a state vector and of the state equations. Section 4 shows typical examples of the procedure described in section 3. The appendices provide formal proofs and a detailed treatment of the examples.

2 Definitions and presentation of the problem

2.1 The canonical form of a discrete-time nonlinear model

It is well known from linear systems theory that a process described by a given transfer function may be represented by a number of state-space representations, corresponding to different sets of state variables. All these state-space representations are strictly equivalent, but some of them have specific properties which are apt to make them more useful or more easily tractable than others; such specific representations are termed canonical forms (Jordan canonical form, first companion form, etc.).

For nonlinear model, the term canonical form does not have a universal meaning. It has been defined for specific families of models, such as S-systems (Voit 1991). In the present framework, we consider that a discrete-time model is in a canonical form if it is in the form

$$\underline{z}(n+1) = \varphi\,[\underline{z}(n), \underline{u}(n)]$$
$$\underline{y}(n+1) = \psi[\underline{z}(n+1)]$$

where $\underline{z}(n)$ is the *minimal* set of ν variables necessary for computing completely the state of the model at time n+1 if the state of the model and its external input vector $\underline{u}(n)$ (control inputs, measured disturbances, ...) are known at time n, and $\underline{y}(n)$ is the output vector.

In terms of recurrent neural network architecture, the dynamic part of the canonical form is made of a feedforward network computing function φ, whose inputs are the state variables and external inputs at time n, and whose outputs are the state variables at time n+1. The output at time n+1 is computed from the state variables at time n+1 by a feedforward neural network implementing function ψ.

2.2 Presentation of the problem: from an arbitrary discrete-time model to a canonical form

We consider a discrete-time model consisting of a set of N equations of the form:

$$x_i(n+1) = \Psi_i\left(\left\langle x_j\,(n - \tau_{ij,k} + 1)\right\rangle_j, \left\langle u_l(n - \tau_{il,k} + 1)\right\rangle_l\right) \quad ,$$
$$i, j = 1, ..., N, \quad l = N+1, ..., N+N', \quad k > 0 \qquad (1)$$

where Ψ_i is an arbitrary function, $\tau_{ij,k}$ is a positive integer denoting the delay of the k-th delayed value of variable x_j used for the computation of $x_i(n+1)$, and where u_l denotes an external input. Relation (1) expresses the fact that the value of variable x_i at time n+1 may be a nonlinear function of (i) all past variables x_j (including x_i itself) and present variables (excluding x_i itself), and (ii) of all external inputs at time n+1 or at previous times. These equations are usually complemented by an output (or observation) equation expressing the relations between the outputs and the state variables of the model.

In the context of neural networks, equations (1) may be considered as the description of a recurrent network where x_i is the output of neuron i, or the output of a feedforward

neural network i, and Ψ_i is the activation function of neuron i, or the function implemented by the feedforward network i.

As a didactic example, consider a process described by the following model :

$$\ddot{x}_1 = f_1\,(x_1, x_2, x_3, u)$$
$$x_2 = f_2(x_1, x_3)$$
$$\ddot{x}_3 = f_3(x_1, \dot{x}_2)$$
$$y = x_3$$

where f_1, f_2 and f_3 are nonlinear functions. After discretization (by Euler's method for instance), these equations have the following form:

$$x_1(n+1) = \Psi_1\,[x_1(n), x_1(n-1), x_2(n-1), x_3(n-1), u_4(n-1)]\,,$$
$$x_2(n+1) = \Psi_2\,[x_1(n+1), x_3(n+1)]\,, \qquad (2)$$
$$x_3(n+1) = \Psi_3\,[x_3(n), x_3(n-1), x_1(n-1), x_2(n), x_2(n-1)]$$
$$y(n+1) = x_3(n+1)\,.$$

Thus, referring to relations (1), one has $N = 3$, $N' = 1$, $\tau_{11,1} = 1$, $\tau_{11,2} = 2$, $\tau_{12,1} = 2$, $\tau_{13,1} = 2$, $\tau_{14,1} = 2$, $\tau_{21,1} = 0$, $\tau_{23,1} = 0$, $\tau_{33,1} = 1$, $\tau_{33,2} = 2$, $\tau_{31,1} = 2$, $\tau_{32,1} = 1$, $\tau_{32,2} = 2$.

The purpose of the paper is to present a methodology that allows one to transform a set of discrete-time equations of the form (1) into a canonical form as defined in 2.1, i.e. to find the minimal set of state variables and the corresponding functions φ and ψ (which will, in general, have parameters which are to be estimated from measured data).

This transformation is performed in three steps:

(i) find the order ν of the system, i.e. find the minimum number of variables $\{z_i\}$ which describe completely the model at time n if their initial values are known, and if the values of the external inputs $\{u_l\}$ are known at times 0 to n;

(ii) find a state vector, i.e. a set of ν state variables,

(iii) transform equations (1) into the state equations which govern the state variables derived in (ii).

In the above example, the equations (2) are not in a canonical form; however, a canonical form is readily derived by substituting the expression of $x_2(n+1)$ into the equations giving $x_1(n+1)$ and $x_3(n+1)$ (or, in the continuous-time model, substituting $x_2(t)$ into the expressions of the second derivatives of $x_1(t)$ and $x_3(t)$): the order of the model is 4, and the state variables are $x_1(n)$, $x_1(n-1)$, $x_3(n)$, $x_3(n-1)$ (or $x_1(t)$, $x_3(t)$ and their first derivatives). We prove in this paper that, for discrete-time models, these derivations and substitutions can be viewed as a sequence of graph transformations which can be performed on a computer in polynomial time.

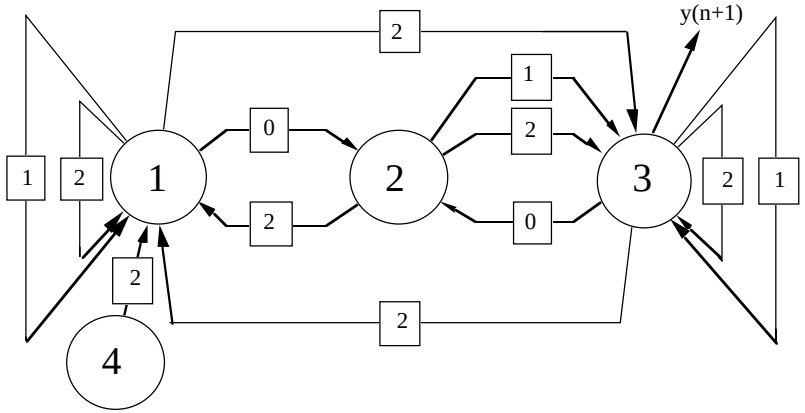
3 Derivation of a state vector

3.1 Graph representation of a dynamic model

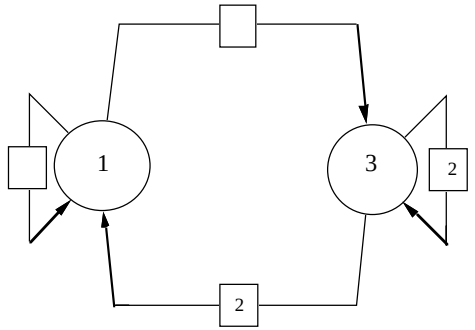
We show in the following that the derivation of a canonical form can be performed by a set of transformations on a graph representation of the recurrent equations (1). We define a finite directed graph $G(E,V)$ consisting of a set of edges E and of a set of vertices V. Each vertex v_i represents a variable x_i. A directed edge e_{ij} from vertex v_j to vertex v_i represents a non-zero term on the right-hand side of equation i of the system of equations (1). The length of each edge is the associated delay $\tau_{ij,k}$: the number of parallel edges from v_j to v_i is equal to the number of different delays $\tau_{ij,k}$. A directed edge from v_j to v_i of length τ is denoted by $e_{ij}{}^{\tau}$ (however, for simplicity, the superscript τ will be omitted whenever the context makes it unnecessary); $\{R_i\}$ denotes the set of outgoing edges from vertex v_i, and the length of the incoming edge to v_i of maximal length is denoted by M_i. $c(v_i)$ is the number of cycles (i.e. the number of paths that start and end at the same vertex) which include vertex v_i; $c(e_{ij})$ is the number of cycles which include edge e_{ij}; A_{ji} is the number of edges e_{ij} from vertex v_j to vertex v_i. Note that the dynamic system is causal if and only if the graph $G(E,V)$ does not contain any cycle of length zero. Figure 1a shows the graph representation of model (2).

3.2 Computation of the order of the model

The first step in the determination of the canonical form of the network consists in finding which variables of the model will give rise to state variables, i.e. will appear as components of the state vector $\underline{z}(n)$ (for instance, in the above example, $\underline{z}(n) = [x_1(n),\ x_1(n\text{-}1),\ x_3(n),\ x_3(n\text{-}1)]^{\mathrm{T}}$: only x_1 and x_3 give rise to state variables). Therefore, we want to reduce the initial graph G_0 of the model to a simpler graph G_1 which contains only the vertices that give rise to state variables (vertices v_1 and v_3 in the above example), and which has the same number of state variables (but not necessarily the same state variables) as the model described by G_0. From this simplified graph we will be able to compute the order of the model.



(a)



(b)

FIGURE 1

(a) Graph representation G_0 of the model described by equations (2); (b) Graph G_1. Following standard notations, the vertices are numbered and the lengths of the edges are written in the squares. The transformations of G_0 to G_1 are described in detail in Appendix 1.

The simplifications of graph G_0 are based on the following considerations:
(i) assume that a model has two state variables $z_i(n)$ and $z_j(n)$ which are such that $z_i(n+1)$ depends on $z_j(n)$, but $z_j(n+1)$ does not depend $z_i(n)$:
$z_i(n+1) = \varphi_i\ [\{z_m(n),\ m = 1,\ ...,\ v\},\ \{u_l(n)\}]$
$z_j(n+1) = \varphi_j\ [\{z_k(n),\ k = 1,\ ...,\ v,\ k \neq i\},\ \{u_l(n)\}]$.
In terms of graph representation, edge e_{ji} is not within a cycle. $z_j(n)$ can be substituted into $z_i(n+1)$ without changing the order of the model. Then $z_i(n+1)$ does not depend *explicitly* on $z_j(n)$, and $z_j(n+1)$ does not depend explicitly on $z_i(n)$. As a consequence, edges of the graph which are not within cycles are irrelevant for the computation of the order of the network;

(ii) assume that a vertex i in G_0 represents an equation of the form

$x_i(n+1) = \Psi_i\left(\{x_j(n+1)\}, \{u_l(n - \tau_{il,k} + 1\}\right)$,

then this relation is static, thus irrelevant for the determination of the state vector;

(iii) assume that one has

$x_m(n+1) = \Psi_m\left(x_j(n - \tau), \{u_l(n - \tau_{ml,k} + 1\}\right)$,

$x_j(n+1) = \Psi_j\left(x_i(n - \tau), \{u_l(n - \tau_{jl,k} + 1\}\right)$.

x_m depends on the external inputs and on x_j only, and x_j depends on the external inputs and on x_i only; in addition, suppose that no variable other than x_m depends on x_j; then variable x_j can be deleted from the model by substitution into x_m without changing the order of the model;

(iv) it has been proved (Nerrand *et al.* 1993) that the order ν of the model represented by a graph G_1 is given by $\nu = \sum_i \omega_i$ where

$$\forall\ v_i \in G_1 \quad \omega_i = \begin{cases} M_i - \min_{e_{ji} \in \{R_i\}} \left(M_j - \tau_{ji}\right) \text{ if } M_i - \min_{e_{ji} \in \{R_i\}} \left(M_j - \tau_{ji}\right) > 0 \\ 0 \qquad \text{otherwise} \end{cases}$$

Assume that two parallel edges, incoming to vertex v_i, exist. Since the computation of ω_i involves the length of the incoming edge of maximal length, only the larger delay is relevant. Now assume that two parallel edges exist, outgoing from vertex v_i, with delays $\tau_{ji}^{\,1}$ and $\tau_{ji}^{\,2}$, $\tau_{ji}^{\,1} > \tau_{ji}^{\,2}$. Then $M_j - \tau_{ji}^{\,1} < M_j - \tau_{ji}^{\,2}$, so that min $(M_j - \tau_{ji}) = M_j - \tau_{ji}^{\,1}$. Thus, when several parallel edges exist, only the edge of maximal length is relevant for the computation of the order of the model.

These remarks result in the following procedure for determining graph G_1 from the initial graph G_0 of the model; G denotes the current state of the graph before and/or after the considered transformation is performed; the tools necessary for performing these transformations, and their time complexity, are described in appendix 1.

I. Delete all edges of G_0 that do not belong to any cycle

$$G \leftarrow G_0 - \{e_{ij} \mid c(e_{ij}) = 0\}$$

and delete all isolated vertices (vertices without incoming nor outgoing edge) that may result.

This transformation stems from remark (i) above. Note that none of the subsequent transformations, described in step II, can generate an edge which does not belong to a cycle. Therefore there is no need for iterating back to this step once the transformations of steps II have been performed. In principle, the present transformation is sufficient for the determination of the order. The transformations of steps II result in further simplifications of the graph, hence of the computations using the information of the graph.

We show in Appendix 1 that the present transformation can be performed in polynomial time.

II. Iterate until no change is possible:

II.1 Delete vertices whose incoming edges are all of zero length, and recombine their incoming and outgoing edges

$$\forall\ v_j, e_{ji}^{0}, e_{kj}^{\tau} \mid M_j = 0 \quad G \leftarrow G - v_j - e_{ji}^{0} - e_{kj}^{\tau} + e_{ki}^{\tau}$$

This transformation stems from remark (ii) above: vertex v_j is deleted, and each pair of edges $(e_{ji}^{\,0}, e_{kj}^{\,\tau})$ is replaced by an edge $e_{ki}^{\,\tau}$ from v_i to v_k with length τ.

II.2 Iterate until no change is possible: if a vertex has one incoming edge only (or one set of parallel incoming edges only) and one outgoing edge only (or one set of parallel outgoing edges only) delete the vertex; if there is a single incoming and a single outgoing edge, merge the edges into a single edge whose length is the sum of the lengths of the merged edges; if there is a set of parallel incoming edges and a set of parallel outgoing edges, merge each pair of one incoming and one outgoing edges into a single edge whose length is the sum of the lengths of the merged edges

$$\forall\ v_j, e_{ij}^{\tau_1}, e_{jk}^{\tau_2} \mid\ A_{ji} \geq 1, A_{jl} = 0\ \forall l \neq i, A_{kj} \geq 1, A_{lj} = 0\ \forall l \neq k\ ,$$
$$G \leftarrow G - v_j - e_{ij}^{\tau_1} - e_{jk}^{\tau_2} + e_{ik}^{\tau_1 + \tau_2}$$

This transformation stems from remark (iii) above.

II.3 Iterate until no change is possible: if several parallel edges between two vertices exist, delete all but the edge of maximum length.

$$\forall\ v_j, e_{ij}^{\tau_1}, e_{ij}^{\tau_2}\ \ G \leftarrow G - e_{ij}^{min(\tau_1, \tau_2)}$$

This transformation stems from remark (iv) above: for each pair of parallel edges, the edge of minimum length is deleted, until only one edge remains.

When no further change is possible, the resulting graph G_1 may be a non-connected graph.

The state equations of the model described by graph G_1 are of the form :

$$\begin{cases} z_1(n) \equiv x_1(n) = \psi_1\left[\{z_{j \in P_1}(n-1)\}\right] \\ z_2(n) = z_1(n-1) \\ \ldots \\ z_{\omega_1}(n) = z_{\omega_1 - 1}(n-1) \end{cases}$$

$$\begin{cases} z_{\omega_1+1}(n) = \psi_2\left[\{z_{j \in P_2}(n-1)\}\right] \\ \ldots \\ z_{\omega_1+\omega_2}(n) = z_{\omega_1+\omega_2-1}(n-1) \end{cases}$$

...

$$\begin{cases} z_{\nu-\omega_{N_V}+1}(n) = y_{N_V}\left[\{z_{j \in P_{N_V}}(n-1)\}\right] \\ \ldots \\ z\ (n) = z\quad (n-1) \end{cases}$$

where N_V is the number of vertices in G_1 and P_i is the set of edges incoming to vertex v_i.

Figure 1b shows the graph G_1 derived from the graph G_0 of Figure 1a. Following the above procedure, edge $e_{14}^{\ 2}$ is deleted, and the output edge from vertex 3 is deleted, since they do not belong to any cycle; then, vertex 2 is deleted since all its incoming edges have zero length, edge $e_{13}^{\ 2}$ and two edges $e_{31}^{\ 2}$ and $e_{31}^{\ 1}$ are generated; finally, parallel edges are deleted iteratively until only edges of maximum length are left. The order of the model is easily derived: one has $M_1 = 2$, $M_3 = 2$, $\omega_1 = 2$, $\omega_3 = 2$, hence $\nu = 4$.

3.3 Determination of a state vector

The order of the model having been computed as shown above, we are looking for a state vector $\underline{z}(n)$, of dimension ν, such that $\underline{z}(n+1) = \varphi[\underline{z}(n), \underline{u}(n)]$, of the form:

$\underline{z}(n) = [x_1(n-k_1) \ldots x_1(n-k_1-w_1+1)\ \ x_2(n-k_2) \ldots x_2(n-k_2-w_2+1) \ldots x_{N_V}(n-k_{N_V}-w_{N_V}+1)]^T$

where k_i and w_i are non-negative integers. w_i is the number of occurrences of the variable x_i in the state vector. If $w_i = 0$, then the variable x_i of the model is not a state variable, and the corresponding k_i is irrelevant; otherwise, k_i denotes the lag of the most recent occurrence of variable x_i in the state vector $\underline{z}(n)$. The w_i's must comply with the following constraint :

$\Sigma_i w_i = \nu$

In the canonical form, the lag between two successive state variables represented by the same vertex is equal to one. Note that several equivalent canonical representations exist: the w_i's may be different from the ω_i's, the only constraint being that the sum of the w_i's must be equal to the order ν.

Thus, one must find a set of $2N_V$ integers $\{k_i, w_i\}$. In order to do this, we first derive a new graph, termed "graph of time constraints", which accounts for the time constraints that exist between the state variables. We subsequently derive the state vector itself.

Determination of the graph of time constraints

The graph of time constraints G_2 is derived from the model graph G_0 by deleting all vertices and edges which are not significant with respect to the time constraints that the state variables must satisfy. The main difference between G_2 and G_1 is the fact that, in order to take the time constraints into account, edges which are not within cycles (thus are not relevant to the determination of the *number* of state variables), but which express a relation between cycles, should be kept because they are relevant to the *choice* of the state variables.

I. Initialize G to G_0. Iterate until no change is possible:

I.1 Delete all vertices whose incoming edges are all of zero length, and recombine their incoming and outgoing edges

$$\forall v_j, e_{ji}^0, e_{kj}^\tau \mid M_j = 0 \quad G \leftarrow G - v_j - e_{ji}^0 - e_{kj}^\tau + e_{ki}^\tau$$

I.2 Iterate until no change is possible: if a vertex has one incoming edge only (or one set of parallel incoming edges only) and one outgoing edge only (or one set of parallel outgoing edges only) delete the vertex and merge the edges into a single edge whose length is the sum of the lengths of the merged edges

$$\forall v_j, e_{ij}^{\tau_1}, e_{jk}^{\tau_2} \mid A_{ji} \geq 1, A_{jl} = 0\ \forall l \neq i, A_{kj} \geq 1, A_{lj} = 0\ \forall l \neq k,$$
$$G \leftarrow G - v_j - e_{ij}^{\tau_1} - e_{jk}^{\tau_2} + e_{ik}^{\tau_1 + \tau_2}$$

I.3 Iterate until no change is possible: if several parallel edges between two vertices exist, delete all but the edge of maximum length.

$$\forall v_j, e_{ij}^{\tau_1}, e_{ij}^{\tau_2}\ \ G \leftarrow G - e_{ij}^{min(\tau_1, \tau_2)}$$

II. Delete all edges that do not have both vertices belonging to at least one cycle:

$$\forall e_{ij} \in G \mid c(v_i) = 0, c(v_j) = 0, \quad G \leftarrow G - e_{ij}$$

The reason for doing this is the following: we are interested in time constraints between state variables only, and we know that state variables arise only from vertices which are within cycles.

The variables of the model which are represented by the vertices of the resulting graph G_2 are the state variables; thus, two integers k_i and w_i ($k_i \geq 0$, $w_i > 0$) are associated to each vertex v_i ; the computation of this set of integers is the final step of the determination of the state vector.

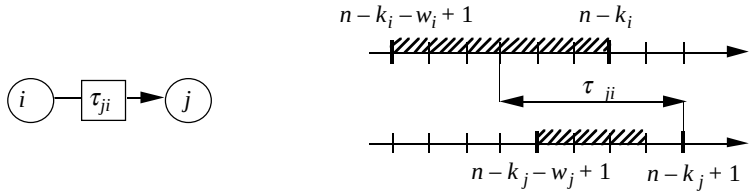


FIGURE 2

(a) The dependence of state variable x_j (represented by vertex v_j) on state variable x_i (represented by vertex v_j) due to the edge e_{ij}^{τ}; (b) the striped zones represent the time spans of the variables x_i (a) and x_j (b)

Determination of the state vector

We denote by N_E the number of edges in the graph of time constraints. Consider an edge e_{ji} of G_2 of length τ_{ji} (Figure 2a): from the very definition of the state vector, and from the construction of the graph of time constraints, it must be possible to compute $x_j(n-k_j+1)$ from one of the state variables, arising from vertex v_i, which are available at vertex v_j at time $n-k_j+1$; these variables must have been computed at vertex v_i at time $n-k_j+1-\tau_{ji}$. Therefore, the following relations must hold if $\tau_{ji} \neq 0$:

$$n - k_i - w_i + 1 + \tau_{ji} \leq n - k_j + 1 \leq n - k_i + \tau_{ji}$$

or equivalently

$$k_j - w_i + \tau_{ji} \leq k_i \leq k_j + \tau_{ji} - 1 \qquad (3)$$

Hence, a set of $2N_E$ such inequalities with $2N_V$ integer variables must be satisfied.

Thus, the problem of finding the state variables and the state equations is amenable to the following linear optimization problem in integer numbers: find the set of integers $\{w_i\}$ such that $\Sigma_i w_i$ is minimum (since the state representation is the *smallest* set of variables that describe the model), under the set of constraints expressed by the inequalities (3). In addition, the value of the minimum is known to be equal to v, whose value is derived as shown in section 3.2.

Note that there is a trivial solution to the set of inequalities (3): $k_i = 0$, $w_i = \max_j \tau_{ji}$. This solution is valid if $\Sigma_i w_i = v$. Otherwise, a solution that satisfies all constraints can be found by linear optimization methods, such as the simplex (Dantzig 1963). The minimized objective function is $\Sigma_i w_i$ and at least one solution with $\Sigma_i w_i = v$ is known to exist. We prove in Appendix 3 that the algorithm (Kuenzi et al. 1971, Press et al. 1992) converges to a solution *with integer values*, which is precisely what is needed. The solution may not be unique.

Once the pairs $\{k_i, w_i\}$ have been determined, the canonical network can be constructed. Note that the effect of merging parallel edges into a maximal delay edge, in step I.3 of the derivation of G_2, eliminates possible singular situations, in which two or more inequalities cannot be simultaneously satisfied: consider two parallel edges of delays $\tau_{ji,1}$, $\tau_{ji,2}$; then two equations (2) should be satisfied simultaneously:

$$k_j - w_i + \tau_{ji,1} \le k_i \le k_j + \tau_{ji,1} - 1 \quad \text{and} \quad k_j - w_i + \tau_{ji,2} \le k_i \le k_j + \tau_{ji,2} - 1$$

If $|\tau_{ji,1} - \tau_{ji,2}| > w_i - 1$, this is impossible; therefore, vertex v_i will be duplicated in the canonical form. The choice of assigning the *largest* delay to merged parallel edges guarantees the feasibility of the canonical form.

4 Examples

The following examples illustrate the application of the proposed method.

4.1 The didactic example

We derived from Figure 1 the order of the model described by equations (2). The graph of time constraints G_2 is identical to G_1. Running the simplex algorithm in this case is useless: from symmetry considerations, and knowing that the order is 4, it is clear that the state vector is $\underline{z}(n) = [x_1(n)\ \ x_1(n\text{-}1)\ x_3(n)\ \ x_3(n\text{-}1)]^T$. From the definition of the canonical form, the state variables and the external inputs are the inputs of its feedforward part; therefore, in order to find this feedforward part, the variables are "backtracked" from the outputs to the inputs (Figure 3): the feedforward part of the canonical form computes $x_1(n+1)$ from the external output and from the components of $\underline{z}(n)$; we see from G_0 that $x_1(n+1)$ is computed from $x_1(n\text{-}1)$, $x_1(n)$, $x_3(n\text{-}1)$, $x_2(n\text{-}1)$, $u_4(n\text{-}1)$; the first three quantities are state variables, so that a direct connection is made between these inputs and vertex 1; $u_4(n\text{-}1)$ will be an input of the feedforward part of the canonical form; $x_2(n\text{-}1)$ is

not present in the input, so that vertex 2 is added; it computes $x_2(n\text{-}1)$ from $x_3(n\text{-}1)$ and $x_1(n\text{-}1)$, which are state variables; thus connections are made from these inputs to vertex 2, which completes this part of the graph; the part of the graph which computes $x_3(n+1)$ is similarly derived, requiring the replication of vertex 2 because of the two parallel edges between vertices 2 and 3 in G_0. The presence of intermediate neurons 2 expresses graphically the fact that the initial model can be put into a canonical form by simply substituting x_2 into the expressions of x_1 and x_3. Note that this network has shared weights: the weights of the inputs of neurons 2. The canonical form of the equations of the model is:

$$\begin{cases} z_1(n+1) \equiv x_1(n+1) = \psi_1\Big[z_1(n),\, z_2(n),\, \psi_2\big[z_2(n),\, z_4(n)\big],\, z_4(n),\, u_4(n-1)\Big] \\ z_2(n+1) = z_1(n) \\ z_3(n+1) = \psi_3\Big[\psi_2\big[z_1(n),\, z_3(n)\big],\, \psi_2\big[z_2(n),\, z_4(n)\big],\, z_2(n),\, z_3(n),\, z_4(n)\Big] \\ z_4(n+1) = z_3(n) \end{cases}$$

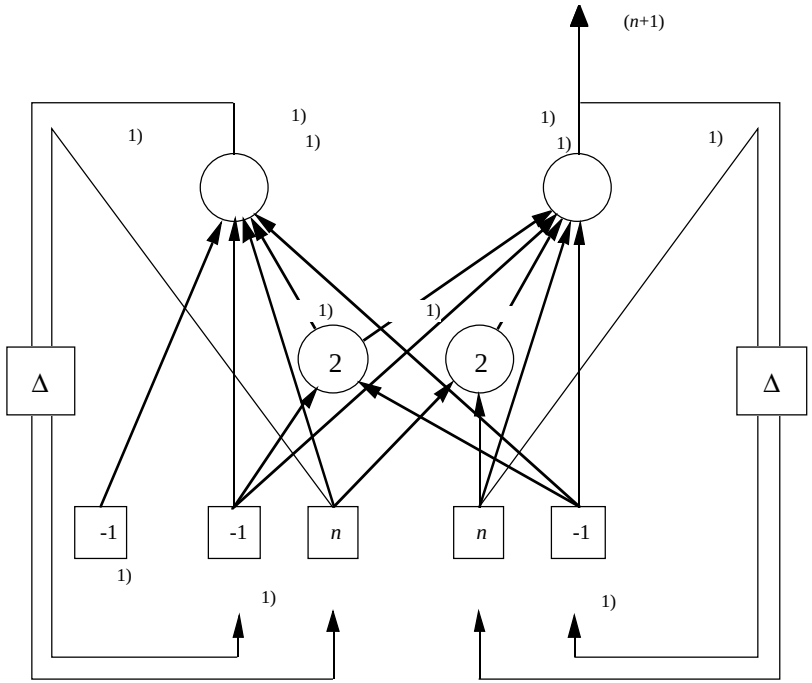


FIGURE 3

The canonical form of the model described by equations (2). Δ stands for unit delays.

4.2 An example where the trivial solution is valid

We consider now the model whose graph G_0 is shown on Figure 4a. It is made of two cascaded parts. G_1 and G_2 are shown on Figures 4b and 4c.

Clearly, the input edges e_{16}, e_{37} and the output edge from vertex 5 are not within cycles; for simplicity, we delete them right away. The mathematical details of the subsequent transformations are given in Appendix 2. The first step of the transformation of the model graph to G_1 is the deletion of edge $e_{42}{}^6$, which does not belong to any cycle. Vertex 5 has one incoming edge ($e_{54}{}^3$) and one outgoing edge ($e_{45}{}^2$), which can be merged into a single edge $e_{44}{}^5$; similarly, vertex 3 has one incoming edge ($e_{32}{}^2$) and one outgoing edge ($e_{13}{}^3$), which can be merged into a single edge $e_{12}{}^5$; in the resulting graph, vertex 1 has only one set of incoming edges ($e_{12}{}^4$ and $e_{12}{}^5$) and one outgoing edge ($e_{21}{}^2$), which can be merged into two edges $e_{22}{}^6$ and $e_{22}{}^7$. Finally, parallel edges are merged, leading to graph G_1 which has two disconnected nodes with one self-loop each. The order ν of graph G_1 is easily derived: $\nu = 12$.

The graph of time constraints G_2 is similarly derived: edges $e_{45}{}^2$, $e_{54}{}^3$, $e_{32}{}^2$ and $e_{13}{}^3$ are deleted, edges $e_{44}{}^5$ and $e_{12}{}^5$ are created. Then edges $e_{12}{}^5$, $e_{12}{}^4$ and $e_{21}{}^2$ are deleted and edges $e_{22}{}^7$ and $e_{22}{}^6$ are generated. Finally, parallel edges are merged. Both ends of edge $e_{42}{}^6$ belong to cycles, thus this edge is kept, resulting in graph G_2. In this case, the trivial solution ($w_i = \max_j \tau_{ji}$; $k_i = 0$) is valid since $\Sigma_j \max_j \tau_{ji} = \nu$. This solution thus defines the state vector:

$$\underline{z}(n) = \left[x_2(n)\ x_2(n\text{-}1)\ \ldots\ x_2(n\text{-}6)\ x_4(n)\ x_4(n\text{-}1)\ \ldots\ x_4(n\text{-}4)\right]^T$$

The corresponding canonical form is shown on Figure 5.

Once this form has been derived, all the weights, either associated to the connections shown, or imbedded in one (or more) feedforward network represented by one (or more) vertex of the graph, can be estimated by training with algorithms using backpropagation through time for computing the gradient of the cost function.

4.3 An example where the trivial solution is not valid

Now the output of the previous model is fed back to one of its inputs with zero delay. Thus, the only difference between the network described in figure 4 and the network described in figure 6a is the addition of edge $e_{35}{}^0$. Since all vertices belong to at least one cycle, no graph simplification is possible: G_1 and G_2 are identical to the original graph.

The order is $\Sigma_i \omega_i = 14$, where:

$\omega_1 = 4 - \min_{e_{21}}(2\text{–}2) = 4$; $\omega_2 = 2 - \min_{e_{32}, e_{42}}(2\text{–}2, 6\text{–}6) = 2$; $\omega_3 = 2 - \min_{e_{13}}(4\text{–}3) = 1$

$\omega_4 = 6 - \min_{e_{54}}(3\text{–}3) = 6$; $\omega_5 = 3 - \min_{e_{35}, e_{45}}(2\text{–}0, 6\text{–}2) = 1$

The trivial solution, which leads to a model of order 16, is thus not valid.

Simplex optimization gives $k_1 = 4$, $w_1 = 1$; $k_2 = 3$, $w_2 = 5$; $k_3 = 6$, $w_3 = 1$; $k_4 = 2$, $w_4 = 2$; $k_5 = 1$; $w_5 = 5$; the state vector is

$$\underline{z}(n) = \left[x_1(n\text{-}4)\ x_2(n\text{-}3)\ x_2(n\text{-}4)\ \ldots\ x_2(n\text{-}7)\ x_3(n\text{-}6)\ x_4(n\text{-}2)\ x_4(n\text{-}3)\ x_5(n\text{-}1)\ \ldots\ x_5(n\text{-}5)\right]^T$$

The canonical form is shown on Figure 6b.

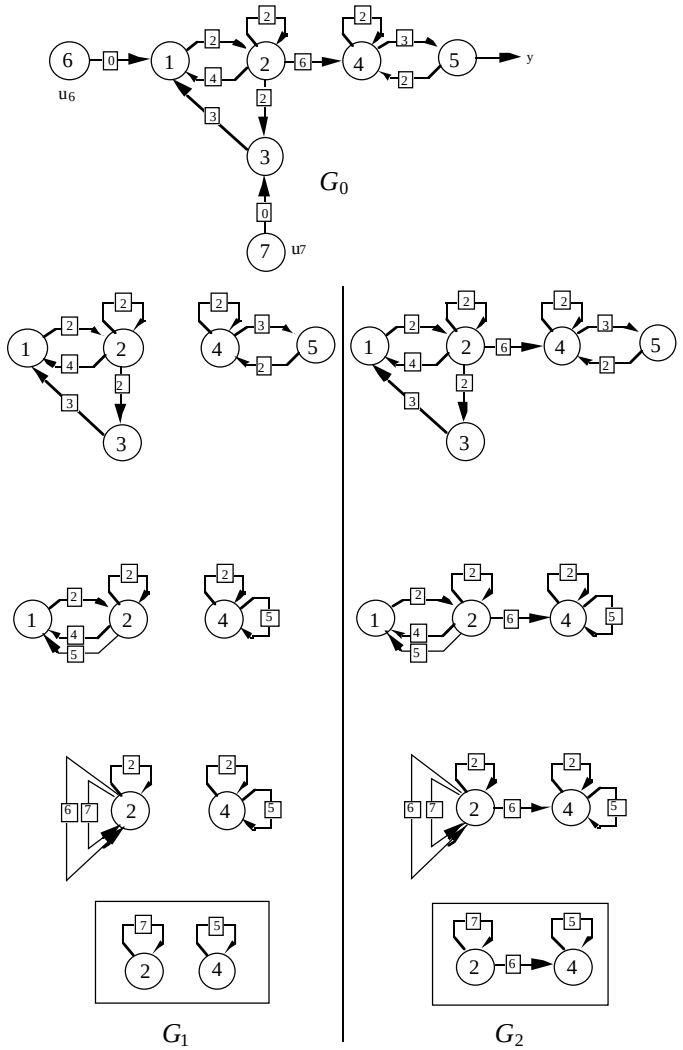


FIGURE 4

Left column, from top to bottom: graph G after step I; graph G after the first iteration of step II.2; graph G after the second iteration of step II.2; graph G_1 (after step II.3).

Right column, from top to bottom :graph G after step I.1; graph G after the first iteration of step I.2; graph G after the second iteration of step I.2; graph G_2 (after step I.3). Details of the transformations are given in Appendix 2

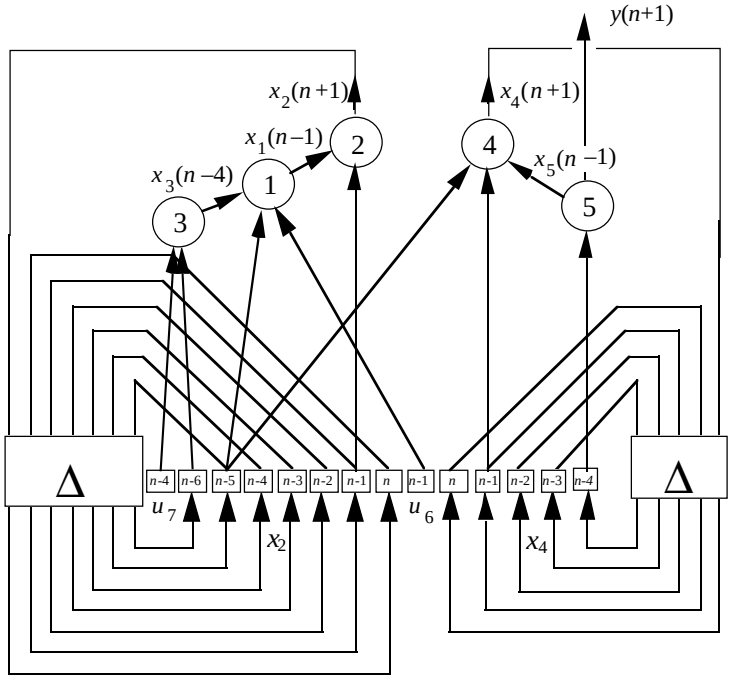


FIGURE 5

Canonical form of the model shown on Figure 4.

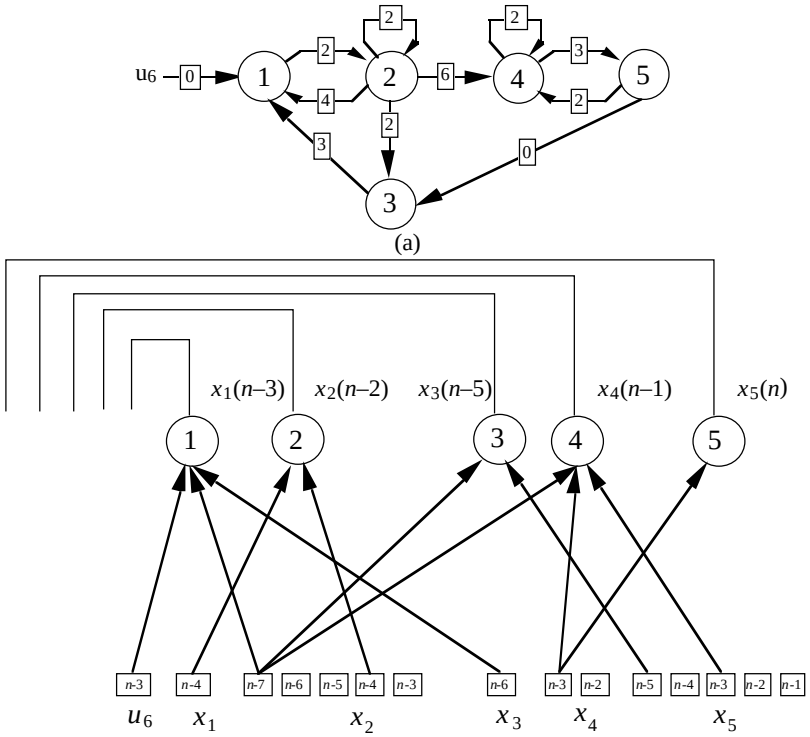
4.4 An example with replicated vertices and shared weights

As a final example, we consider a model whose canonical form makes extensive use of duplicated vertices and shared weights. Its graph representation is shown on Figure 7a. In order to compute the order of this model, we derive the graph G_1 of figure 7b.

The order of the network is 6 ($\omega_1 = 1,\ \omega_2 = 2,\ \omega_3 = 1,\ \omega_5 = 2$) but the trivial solution from the graph of time constraints shown in figure 7c gives a solution of order 8 ($w_1 = w_2 = w_3 = w_5 = 2$), which is not optimal. Note that, during the graph simplification, vertices 1 and 3 have parallel outgoing edges and corresponding constraints that cannot be simultaneously satisfied; thus, these vertices are replicated, and so are the weights corresponding to their inputs and outputs. These edges are $e_{53}{}^0$ and $e_{53}{}^2$ for vertex 3 (once vertex 4 is eliminated) and $e_{21}{}^0$ and $e_{21}{}^2$ for vertex 1.

The solution found by the simplex method is: $k_1 = 2,\ w_1 = 1;\ k_2 = 1,\ w_2 = 2;\ k_3 = 1,\ w_3 = 1;\ k_5 = 0;\ w_5 = 2$.

The canonical form of this network is shown on figure 8. It is exactly the canonical form that was derived "manually" in a previous paper [Nerrand et al.].

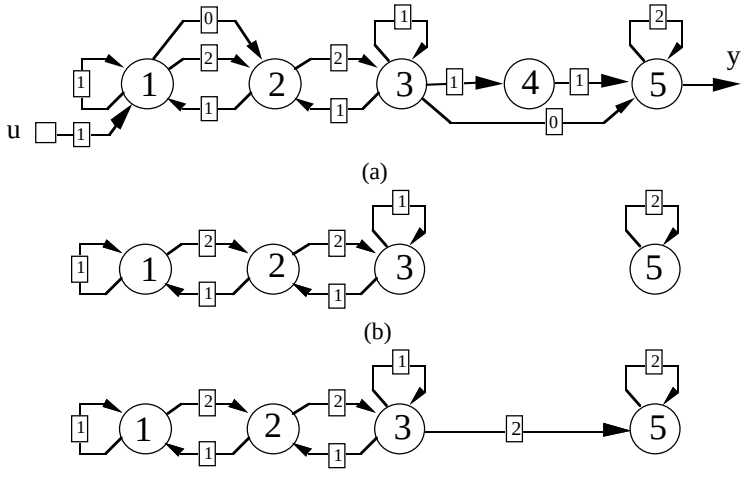


(a)

(b)

FIGURE 6

(a) Graph of the model of Figure 4, with feedback; G_1 and G_2 are identical to G_0 (see Appendix 2); (b) canonical form; for clarity, all the recurrent connections are not shown.



(a)

(b)

(c)

FIGURE 7

(a) Graph representation G_0 of a model, with parallel edges; (b) graph G_1 ; (c) graph G_2.

5 Conclusions

This work was motivated by the present interest in semi-physical modeling, be it "neural" or not. It is clear that black-box modeling is wasteful of information when large, complex systems, such as arise in the process industry for instance, are considered. The importance of canonical forms of models has been recognized long ago in the field of linear automatic control; in nonlinear modeling, the problem of finding a canonical form for a model described by an arbitrary set of coupled nonlinear equations is also important, but it is much more difficult; in addition, when neural networks are used for modeling, putting the network into a canonical form is mandatory for simplicity of implementation of the training algorithms. We have proposed a general procedure for finding the canonical form of a model described by a set of coupled nonlinear discrete-time equations. Knowledge-based neural modeling, which consists in building a neural net complying with the equations of the model, complementing it by black boxes taking into account the part of the dynamics which is not modeled by the initial set of equations, and subsequently training the network from measured data in order to estimate the unknown parameters, has been used successfully for complex industrial applications (Ploix et al. 1994, Ploix et al. 1996). The present work is a step towards making the first part of this task (deriving a neural network from the equations of the model and putting it in canonical form before training) fully automatic.

Acknowledgements : the authors are grateful to Brigitte QUENET for her critical reading of the manuscript.

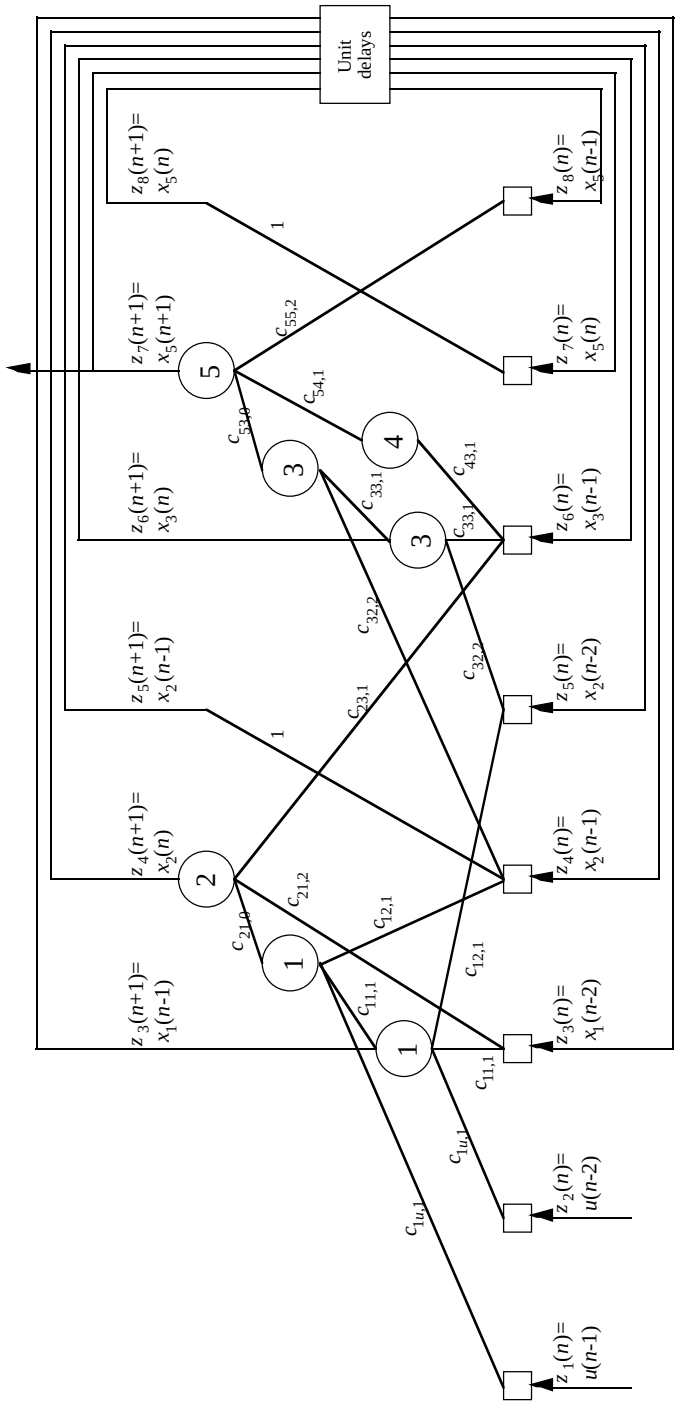


FIGURE 8

Canonical form of the model of Figure 7. Details of the transformations are given in appendix 2.

APPENDIX 1

In this appendix, we describe the computational tools that are necessary for the graph transformations, and we show that the latter have polynomial complexity.

The basic tool is the adjacency matrix A of a graph G. Element A_{ij} of matrix A is equal to the number of edges from vertex i to vertex j. The diagonal terms of the matrix denote the number of self-loops (cycles of length 1).

We consider the adjacency matrix A_0 of graph G_0. We define the matrix A_1 as

$A_1 = \text{sign}(A_0)$: $(A_1)_{ij} = 1$ if $(A_0)_{ij} > 0$, $(A_1)_{ij} = 0$ if $(A_0)_{ij} = 0$.

Transformation of graph G_0 to graph G_1: step I

The first step in the transformation of G_0 to G_1 consists in finding and deleting all edges that do not belong to any cycle. Therefore, for each edge e_{ji}, one has to find whether a path from j to i exists in the graph.

In this task, the actual length of the edges (i.e. the delay associated to each edge) is irrelevant: therefore, in the following, we consider that all edges of graph G have unit length; the length of a path in the graph is thus equal to the number of edges of the path.

We make use of the following result : *consider the sequence of matrices* $\{A_n\}$ *defined by* $A_n = \text{sign}(A_1 A_{n-1})$, $n \geq 2$ *:* $(A_n)_{ij} = 1$ *if and only if there exists at least one path from* i *to* j *in the graph.*

This is easily proved by recursion. Consider matrix A_1^2 :

$$(A_1^2)_{ij} = \sum_k (A_1)_{ik} (A_1)_{kj}$$

Each path of length 2 from i to j contributes a non-zero term to the right-hand side of the above relation; hence $(A_1^2)_{ij}$ is equal to the number of paths of length 2 from vertex i to vertex j. We define $A_2 = \text{sign}(A_1^2)$: a non-zero element $(A_2)_{ij}$ denotes the existence of at least one path of length 2 from vertex i to vertex j in the graph.

Similarly, we define matrix A_n whose element $(A_n)_{ij}$ is equal to 1 if and only if there is at least one path of length n from i to j in the graph, and is equal to 0 otherwise. It is easy to show, as before, that matrix $A_{n+1} = \text{sign}(A_1 A_n)$ has the following property : $(A_{n+1})_{ij} = 1$ if and only if there is at least one path of length n+1 from i to j.

The longest simple path (i.e. path that contains no cycle) in a directed graph of N vertices is of length N. Therefore, in the worst case, the construction of the sequence of matrices $\{A_n\}$ is terminated when $n = N$, thus *in polynomial time*; actually, the sequence will frequently terminate when $A_{n+1} = A_n$ with $n < N$, as shown in the examples below.

Finally, consider matrix

$$A^* = \text{sign}\left(\sum_{n=1} A_n\right)$$

Element $(A^*)_{ij}$ is equal to 1 if and only if there is at least one path in the graph from i to j.

A diagonal element $(A^*)_{ii}$ is equal to 1 if and only if vertex i belongs to at least one cycle.

To summarize: in step I of the transformation of graph G_0 to graph G_1, consider each pair of vertices (i, j): if $(A_0)_{ij} \neq 0$, there is at least one edge between i and j; if $(A_0)_{ij} \neq 0$ and $(A^*)_{ji} = 0$, edges e_{ji}^{τ} are deleted ($(A_0)_{ij}$ is set to zero); otherwise, edges e_{ji}^{τ} are kept in the graph.

Transformation of graph G_0 to graph G_1: step II

Step II.1 : if a vertex j has all incoming edges of length 0, all elements of row j and column j of the adjacency matrix A of the current graph G are set to zero; for each pair of elements (A_{ij}, A_{jk}, $j \neq k$) set to zero, add A_{jk} to A_{ik}.

Step II.2 : consider two adjacent edges e_{ij} and e_{jk}; they are both within at least one cycle (otherwise they would have been deleted at the previous step); vertex j belongs exclusively to this cycle (or set of cycles) if and only if there is no incoming edge to j from vertices other than k, and no outgoing edge from j to vertices other than i; thus one must have $A_{kj} \geq 1$, $A_{lj} = 0$ for all $l \neq k$, and $A_{ji} \geq 1$, $A_{jl} = 0$ for all $l \neq i$: there must be one and only one non-zero off-diagonal element in row j, there must be one and only one non-zero off-diagonal element in column j, and both A_{ii} and A_{jj} must be equal to zero. Elements A_{kj} and A_{ji} are then set to zero, and A_{ki} is increased by $A_{kj}A_{ji}$.

Step II.3 consists in computing sign(A) and appropriately updating the length of the connections.

Transformation of graph G_0 to graph G_2

Steps I.1, I.2 and I.3 are formally the same as steps II.1, II.2, II.3 of the transformation of G_0 to G_1, but they do not act on the same initial graph: in the transformation of G_0 to G_1, these steps are performed on G_0 deprived of the edges which are not within cycles, whereas, in the transformation of G_0 to G_2 they are performed on G_0 itself.

In step II of the transformation from G_0 to G_2, one has to find and delete all edges that do not have both vertices belonging to at least one cycle. Consider each edge in turn; if $(A_0)_{ij} = 0$, there is no edge for consideration; if $(A_0)_{ij} \neq 0$, and if $(A^*)_{ii} = (A^*)_{jj} = 1$, the edge must be kept in G_2; otherwise it is deleted.

Keeping track of the edge lengths during these transformations is very simple.

Example

To illustrate this technique, we consider the model whose graph G_0 is shown on Figure 1a.

Derivation of G_1: step I

The adjacency matrix of graph G_0 is:

$$A_0 = \begin{pmatrix} 2\,1\,1\,0 \\ 1\,0\,2\,0 \\ 1\,1\,2\,0 \\ 1\,0\,0\,0 \end{pmatrix} \text{ therefore } A_1 = \text{sign}(A_0) = \begin{pmatrix} 1\,1\,1\,0 \\ 1\,0\,1\,0 \\ 1\,1\,1\,0 \\ 1\,0\,0\,0 \end{pmatrix} .$$

$$A_1^2 = \begin{pmatrix} 3\,2\,3\,0 \\ 2\,2\,2\,0 \\ 3\,2\,3\,0 \\ 1\,1\,1\,0 \end{pmatrix} \text{ therefore } A_2 = \text{sign}(A_1^2) = \begin{pmatrix} 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \end{pmatrix} .$$

$$A_1 * A_2 = \begin{pmatrix} 3\,3\,3\,0 \\ 2\,2\,2\,0 \\ 3\,3\,3\,0 \\ 1\,1\,1\,0 \end{pmatrix} \text{ therefore } A_3 = \text{sign}(A_1 * A_2) = \begin{pmatrix} 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \end{pmatrix} .$$

Since A_3 is identical to A_2 the sequence construction stops at this point.

$$A^* = \text{sign}\left(\sum_{n=1}^{2} A_n\right) = \begin{pmatrix} 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \\ 1\,1\,1\,0 \end{pmatrix}; \quad A^{*T} = \begin{pmatrix} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 0\,0\,0\,0 \end{pmatrix}$$

Considering each non-zero element of A_1, and the corresponding element of A^{*T}, it is easily seen that all edges belong to at least one cycle, except for edge e_{14}, which is therefore deleted. At this step, the adjacency matrix of the current graph G is :

$$A = \begin{pmatrix} 2\,1\,1\,0 \\ 1\,0\,2\,0 \\ 1\,1\,2\,0 \\ 0\,0\,0\,0 \end{pmatrix}.$$

Derivation of G_1: step II

II.1: since vertex 2 has two incoming edges, which are both of zero length, A_{12} and A_{32} are set to zero, and edges $e_{13}^{\ 2}$, $e_{31}^{\ 1}$ and $e_{31}^{\ 2}$ are generated, resulting in a new adjacency matrix:

$$A = \begin{pmatrix} 2\,0\,3\,0 \\ 0\,0\,0\,0 \\ 2\,0\,2\,0 \\ 0\,0\,0\,0 \end{pmatrix} .$$

II.2: in this matrix, no vertex is such that the corresponding row and the corresponding column both have one and only one non-zero element. Therefore, no simplification can be performed at step II.2.

II.3: finally, matrix A shows that parallel edges exist as self-loops around vertices 1 and 3, and for edges e_{31} and e_{13}. All but the edges of maximal length are deleted.

No further iteration is necessary to any of the steps II; thus, the adjacency matrix for G_1 is:

$$A = \begin{pmatrix} 1\,0\,1\,0 \\ 0\,0\,0\,0 \\ 1\,0\,1\,0 \\ 0\,0\,0\,0 \end{pmatrix} .$$

Graph G_1 is shown on Figure 1b.

Derivation of G_2

We start with matrix A_0:

$$A_0 = \begin{pmatrix} 2\,1\,1\,0 \\ 1\,0\,2\,0 \\ 1\,1\,2\,0 \\ 1\,0\,0\,0 \end{pmatrix}$$

Step I.1: vertex 2 is deleted as in step II.1 above, leading to matrix:

$$A = \begin{pmatrix} 2\,0\,3\,0 \\ 0\,0\,0\,0 \\ 2\,0\,2\,0 \\ 1\,0\,0\,0 \end{pmatrix} .$$

Step I.2: no transformation.

Step I.3: parallel edges are deleted except for the edges of maximal length:

$$A = \begin{pmatrix} 1\,0\,1\,0 \\ 0\,0\,0\,0 \\ 1\,0\,1\,0 \\ 1\,0\,0\,0 \end{pmatrix} .$$

Step II: edge e_{41} has only vertex 1 belonging to a cycle, hence this edge is deleted, leading to an adjacency matrix for G_2 which is identical to that of G_1.

APPENDIX 2

This appendix describes the computations of the graph transformations illustrated by Figures 4, 6 and 7.

Figure 4

It is clear by inspection that edges e_{16} and e_{37} are irrelevant for the determination of the order and of the state variables. In order to simplify the notations, we overlook these edges from the beginning, thus considering only vertices 1 to 5; in a computer implementation of the procedures, these edges are deleted in step I of the transformation of G_0 to G_1, and they are deleted in step II of the transformation of G_0 to G_2.

Derivation of G_1 : step I

$$A_0 = \begin{pmatrix} 0&1&0&0&0\\ 1&1&1&1&0\\ 1&0&0&0&0\\ 0&0&0&1&1\\ 0&0&0&1&0 \end{pmatrix};\ A_1 = A_0;\ A_2 = \begin{pmatrix} 1&1&1&1&0\\ 1&1&1&1&1\\ 0&1&0&0&0\\ 0&0&0&1&1\\ 0&0&0&1&1 \end{pmatrix};\ A_3 = \begin{pmatrix} 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&0\\ 0&0&0&1&1\\ 0&0&0&1&1 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&1\\ 0&0&0&1&1\\ 0&0&0&1&1 \end{pmatrix};\ A_5 = A_4;\ A^* = \begin{pmatrix} 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&1\\ 0&0&0&1&1\\ 0&0&0&1&1 \end{pmatrix};\ A^{*\mathrm{T}} = \begin{pmatrix} 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&1&1\\ 1&1&1&1&1 \end{pmatrix}.$$

Thus, edge e_{42} is deleted, hence the current adjacency matrix : $A = \begin{pmatrix} 0&1&0&0&0\\ 1&1&1&0&0\\ 1&0&0&0&0\\ 0&0&0&1&1\\ 0&0&0&1&0 \end{pmatrix}$

Derivation of G_1: step II

Step II.1: no transformation.

Step II.2: row 5 and column 5 have one and only one non-zero, off-diagonal element (A_{45} and A_{54}) ; same for row and column 3 (A_{23} and A_{31}). Hence edges e_{45}, e_{54}, e_{13} and e_{32} are deleted, one edge e_{44} and one edge e_{12} are generated: $A = \begin{pmatrix} 0&1&0&0&0\\ 2&1&0&0&0\\ 0&0&0&0&0\\ 0&0&0&2&0\\ 0&0&0&0&0 \end{pmatrix}$.

Row 1 and column 1 have one and only one non-zero, non-diagonal element, (A_{12} and A_{21}), hence edges e_{21} and e_{12} are deleted, and two edges from 2 to 2 are generated:

$A = \begin{pmatrix} 0&0&0&0&0\\ 0&3&0&0&0\\ 0&0&0&0&0\\ 0&0&0&2&0\\ 0&0&0&0&0 \end{pmatrix}$

Step II.3: parallel edges are merged, resulting in graph G_1.

Derivation of G_2: step I

Step I.1: no transformation.

Step I.2: as above, edges e_{45}, e_{54}, e_{13} and e_{32} are deleted, edge e_{44} and edge e_{12} are generated : $A = \begin{pmatrix} 0&1&0&0&0\\ 2&1&0&1&0\\ 0&0&0&0&0\\ 0&0&0&2&0\\ 0&0&0&0&0 \end{pmatrix}$; edges e_{21}, $e_{12}{}^4$ and $e_{12}{}^5$ are deleted, and two edges from 2 to 2 are generated: $A = \begin{pmatrix} 0&0&0&0&0\\ 0&3&0&1&0\\ 0&0&0&0&0\\ 0&0&0&2&0\\ 0&0&0&0&0 \end{pmatrix}$.

Step I.3: parallel edges are merged.

Derivation of G_2: step II

Since both ends of edge e_{42} are within cycles (A^*_{22} and A^*_{44} are both non-zero), this edge is kept in G_2.

Figure 6

$$A_0 = \begin{pmatrix} 0&1&0&0&0\\ 1&1&1&1&0\\ 1&0&0&0&0\\ 0&0&0&1&1\\ 0&0&1&1&0 \end{pmatrix};\ A_1 = A_0;\ A_2 = \begin{pmatrix} 1&1&1&1&0\\ 1&1&1&1&1\\ 0&1&0&0&0\\ 0&0&1&1&1\\ 1&0&0&1&1 \end{pmatrix};\ A_3 = \begin{pmatrix} 1&1&1&1&0\\ 1&1&1&1&1\\ 1&1&1&1&0\\ 1&0&1&1&1\\ 0&1&1&1&1 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&1 \end{pmatrix};\ A^* = A_4$$

All edges belong to at least one cycle, so that no simplification is possible.

Figure 7

Derivation of G_1 : step I

$$A_0 = \begin{pmatrix} 1&2&0&0&0\\ 1&0&1&0&0\\ 0&1&1&1&1\\ 0&0&0&0&1\\ 0&0&0&0&1 \end{pmatrix};\ A_1 = \begin{pmatrix} 1&1&0&0&0\\ 1&0&1&0&0\\ 0&1&1&1&1\\ 0&0&0&0&1\\ 0&0&0&0&1 \end{pmatrix};\ A_2 = \begin{pmatrix} 1&1&1&0&0\\ 1&1&1&1&1\\ 0&1&1&1&1\\ 0&0&0&0&1\\ 0&0&0&0&1 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&1\\ 0&0&0&0&1\\ 0&0&0&0&1 \end{pmatrix};\ A_4 = A_3;\ A^* = \begin{pmatrix} 1&1&1&1&1\\ 1&1&1&1&1\\ 1&1&1&1&1\\ 0&0&0&0&1\\ 0&0&0&0&1 \end{pmatrix};\ A^{*T} = \begin{pmatrix} 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&1&1 \end{pmatrix}$$

Edges $e_{43}{}^1$, $e_{54}{}^1$ and $e_{53}{}^0$ are deleted: $A = \begin{pmatrix} 1&2&0&0&0\\ 1&0&1&0&0\\ 0&1&1&0&0\\ 0&0&0&0&0\\ 0&0&0&0&1 \end{pmatrix}$.

Derivation of G_1: step II

Step II.1: no transformation.

Step II.2: no transformation.

Step II.3: parallel edges $e_{21}{}^0$ and $e_{21}{}^2$ are merged into $e_{21}{}^2$.

Derivation of G_2: step I

Step I.1: no transformation.

Step I.2: since row 4 and column 4 of A_0 have one and only one non-zero off-diagonal element, $(A_0)_{34}$ and $(A_0)_{45}$ are set to zero, and $(A_0)_{35}$ is increased by 1, resulting in:

$$A = \begin{pmatrix} 1&2&0&0&0 \\ 1&0&1&0&0 \\ 0&1&1&0&2 \\ 0&0&0&0&0 \\ 0&0&0&0&1 \end{pmatrix}$$

Step I.3: parallel edges ${e_{21}}^2$ and ${e_{21}}^0$ are merged into ${e_{21}}^2$; parallel edges ${e_{53}}^0$ and ${e_{53}}^2$ are merged to ${e_{53}}^2$, resulting in

$$A = \begin{pmatrix} 1&1&0&0&0 \\ 1&0&1&0&0 \\ 0&1&1&0&1 \\ 0&0&0&0&0 \\ 0&0&0&0&1 \end{pmatrix}$$

Derivation of G_2: step II

Since all the relevant diagonal elements of A^* are non-zero, no simplification is possible: the above matrix is the adjacency matrix of G_2.

APPENDIX 3

We proved in Section 3.3 that the problem of the determination of a state vector is amenable to the following linear optimization problem: minimize the cost function

$$\sum_{i=1}^{N_v} w_i$$

subject to the constraints:

$w_i > 0 \ \ \forall i,$

$k_j \geq 0 \ \ \forall j,$

$k_j - k_i - w_i + \tau_{ji} \leq 0 \ \ \forall \tau_{ji}$

$k_i - k_j - \tau_{ji} + 1 \leq 0 \ \ \forall \tau_{ji}$

where all variables k_i and w_i are integers, and where the constant terms τ_{ji} are integers. In this appendix, we prove that the simplex algorithm provides an optimal solution in integer numbers.

The general form of a linear optimization problem

The general form of a linear optimization problem with N variables $\{x_i\}$ is the following: find a vector $[x_1, x_2, ..., x_N]^T$ which maximizes the cost function

$$\sum_{i=1}^{N} f_i x_i$$

under the non-negativity (hereinafter termed *primary*) constraints

$x_i \ \geq \ 0$ for all i (A3.1),

and a set of C additional (hereinafter termed *secondary*) constraints

$$\sum_{j=1}^{N} a_{ij} x_j \leq b_i \ \ (b_i \geq 0) \quad \text{(A3.2)}$$

$$\sum_{j=1}^{N} a_{kj} x_j \geq b_k \geq 0 \quad \text{(A3.3)}$$

$$\sum_{j=1}^{N} a_{lj} x_j = b_l \geq 0 \quad \text{(A3.4)}$$

In our problem, all constant terms $\{b_i, i = 1 \text{ to } C\}$ are integers, all coefficients $\{a_{ij}, i = 1 \text{ to } C, j = 1 \text{ to } N\}$ are equal to -1, 0 or +1. It is desired to find an optimal solution where all variables $\{x_i\}$ are integers.

We first recall an essential result of linear optimization theory:

(i) if an optimal vector exists, and if $N < C$, i.e. if its dimension is smaller than the number of secondary constraints (A3.2) to (A3.4), then there is an optimal vector whose components satisfy N out of C constraints *as equalities*;

(ii) if an optimal vector exists, and if $N > C$, then all C secondary constraints (A3.2) to (A3.4) are satisfied as equalities, and (N-C) primary constraints (A3.1) are also satisfied as equalities, so that the optimal vector has at least (N-C) components equal to zero.

The simplex algorithm takes advantage of this result, by searching an optimal vector among the vectors that satisfy N out of C secondary constraints as equalities if $N < C$, or that satisfy C secondary constraints and (N-C) primary constraints as equalities if $N > C$.

We are going to prove that, if the optimization problem has a solution, then the simplex algorithm provides a solution in integer numbers. The proof is organized in two steps; we first prove that if a linear optimization problem is in restricted normal form (to be defined below) with integer constant terms and coefficients equal to -1, 0 or +1, and if it has a solution, then a solution in integer numbers exists and is found by the simplex algorithm. In the second step, we show that a *general* linear optimization problem with integer constant terms and coefficients equal to -1, 0 or +1 is amenable to an equivalent linear optimization problem *in restricted normal form* with integer constant terms and coefficients equal to -1, 0 or +1. Since we know that the problem of the determination of the state vector has a solution, we conclude that the simplex algorithm provides a solution in integer numbers.

Linear optimization problem in restricted normal form

A linear optimization problem is said to be in restricted normal form if (i) the only constraints are the N non-negativity constraints (A3.1) and C *equality* constraints (A3.4), and if (ii) each equality constraint has at least one variable which has a positive coefficient and appears in one constraint only; these C variables $\{x_i,\ i = 1 \text{ to } C\}$ are called basic variables, and the other (N-C) variables $\{x_i,\ i = C{+}1 \text{ to } N\}$ are called non-basic variables. The C equality constraints can be solved for the basic variables, hence can be written in the form:

$$x_i = b_i + \sum_{j=C+1}^{N} a_{ij} x_j \qquad i = 1 \text{ to } C \quad \text{(A3.5)}$$

All basic variables are on the left-hand side of (A3.5), while on non-basic variables are on the right-hand side. In our problem, all b_i's are integers and all a_{ij}'s are equal to -1, 0 or +1.

By setting all non-basic variables to zero, one obtains from (A3.5) an initial vector, with at most C non-zero components (the basic variables) which are equal to the constant terms on the right-hand side of (A3.5), and at least (N-C) components equal to zero (the non-basic variables); this vector satisfies all constraints as equalities, but is not necessarily optimal. We know from the basic result recalled above that an optimal vector, if it exists, is to be found among the vectors that, similarly to the initial vector, have at most C non-zero components and at least (N-C) components equal to zero. Therefore, a new candidate vector can be obtained from the initial vector by turning one basic variable into a non-basic variable and one non-basic variable into a basic variable (how this can be done will be explained below). Assume that the non-basic variable x_m has been turned into a basic variable, and that the basic variable x_n has been turned into a non-basic variable; then the constraints are in a form similar, and equivalent, to (A3.5), where x_m now

appears on the left-hand side of one of the equations, and x_n appears on the right-hand side of at least one of the equations; by setting all the new non-basic variables to zero, one obtains a new candidate vector whose component x_n is equal to zero, and whose component x_m may be non-zero; yet another candidate vector can be derived from the present candidate vector by performing another exchange, and so on. Thus, the problem of finding an optimal vector can be regarded as a combinatorial problem. A brute-force procedure would consist in trying *all* possible sequences of exchanges of one basic variable for one non-basic variable and selecting the optimal vector thus found; this would involve impractical computation times. The simplex algorithm is a very economical procedure which starts from the initial vector defined above, and performs an appropriate sequence of exchanges of one non-basic variable for one basic variable, maximizing the increase of the cost function at each exchange, until no further increase of the cost function is possible; the details of the algorithm (Press et al. 1992, Kuenzi et al. 1971), i.e. how the decision is made to exchange a certain non-basic variable for a certain basic variable at each step of the procedure, are irrelevant for the present proof. The only important point is the following: assume that it is found desirable, at the first step of the procedure, to exchange the non-basic variable x_m for the basic variable x_n; the n-th equation of (A3.5) can be solved for x_m:

$$x_n = b_n + \sum_{j=C+1}^{N} a_{nj} x_j \Rightarrow x_m = -\frac{b_n}{a_{nm}} - \sum_{\substack{j=C+1 \\ j \neq m}}^{N} \frac{a_{nj}}{a_{nm}} x_j + \frac{x_n}{a_{nm}}$$

Then x_m can be substituted into all the other equations of (A3.5), thereby making x_n a non-basic variable. In our problem, b_n is an integer, and a_{nm} is equal to -1 or +1; therefore, the constant term in the expression of x_m is an integer and the coefficients of the non-basic variables are equal to -1, 0 or +1; similarly, the constant terms in the other constraints after the exchange of x_m and x_n are integers, and the coefficients of the variables are equal to -1, 0 or +1. Clearly, the same result holds true after each exchange of the procedure. After the final exchange, which leads to the optimal combination of non-basic and basic variables, the vector which is found by setting the non-basic variables to zero has N-C components which are equal to zero (the final non-basic variables) and C components which are *integer numbers* (the final basic variables), equal to the constant terms on the right-hand side of the constraints in their final form.

Thus, we have proved that, if a linear optimization problem is in restricted normal form, if it has a solution, if all coefficients of the variables are equal to -1, 0 or +1, and if all constant terms of the constraints are integers, then an optimal solution in integer numbers will be found by the simplex algorithm.

General case

In the case of the determination of the order of a model, we know that an optimal solution exists, but the problem is not in restricted normal form, so that the above result is not directly applicable. We prove in the following that the problem is nevertheless amenable

to an equivalent problem in restricted normal form with coefficients equal to -1, 0 or +1, and with integer constant terms, so that the result of the previous section holds in general. In order to do this, one first turns the inequality constraints (A3.2) and (A3.3) into equality constraints; this is achieved by introducing an additional non-negative variable into each inequality constraint:

$$\sum_{j=1}^{N} a_{ij}\, x_j \le b_i, \quad b_i > 0 \;\rightarrow\; \sum_{j=1}^{N} a_{ij}\, x_j - y_i = b_i, \quad y_i \ge 0 \qquad \text{(A3.6)}$$

$$\sum_{j=1}^{N} a_{kj}\, x_j \ge b_k \ge 0 \;\rightarrow\; \sum_{j=1}^{N} a_{kj}\, x_j + y_k = b_k, \quad y_k \ge 0 \qquad \text{(A3.7)}$$

Note that the coefficients of the new variables are equal to -1 or +1.

The second step of the transformation of the general form to the restricted normal form consists in adding a second set additional variables $\{z_i\}$, which casts (A3.6) and (A3.7) into the form (A3.5):

$$z_i = b_i + y_i - \sum_{j=1}^{N} a_{ij}\, x_j$$

$$z_k = b_k - y_k - \sum_{j=1}^{N} a_{kj}\, x_j$$

This set of equalities defines an optimization problem which is in restricted normal form, with integer constant terms and with coefficients equal to -1, 0 or +1. Any solution of this problem having all z_i's equal to zero is a solution of the original problem; the simplex algorithm is organized in such a way that the solution found has all z_i's equal to zero. Therefore the simplex algorithm finds a solution of the original problem in integer numbers, if a solution exists.

In the case of the determination of the state vector, we know that a solution exists. Therefore, the simplex algorithm finds an optimal set of integers $\{k_i, w_i\}$.

REFERENCES

Dantzig, GB (1963), *Linear programming and extensions*. Princeton University Press.

Kuenzi, H.P., H.G. Tzschach, and C.A Zehnder (1971), *Numerical methods of mathematical optimization*. Academic Press

Lindskog P., *Algorithms and Tools for System Identification Using Prior Knowledge*, Linköping Studies in Science and Technology, thesis # 456.

Nerrand, O., P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos (1993), Neural networks and nonlinear adaptive filtering: unifying concepts and new algorithms, Neural Computation vol. 5, 165-197.

Ploix J.L., G. Dreyfus, J.P. Corriou, D. Pascal (1994), From Knowledge-based Models to Recurrent Networks: an Application to an Industrial Distillation Process, *Neural Networks and their Applications*, J. Hérault ed.

Ploix J.L., G. Dreyfus (1996), Knowledge-based Neural Modeling: Principles and Industrial Applications, *Industrial Applications of Neural Networks*, F. Fogelman, P. Gallinari, eds. (World Scientific).

Press, W.H., S.A. Teukolsky, W.T. Vetterling, B.P. Flannery (1992), *Numerical Recipes in C : the Art of Scientific Computing*, Cambridge University Press.

Thoma, J.(1991), *Simulation by Bond Graphs* (Springer).

Voit, E.O. (1991), *Canonical Nonlinear Modeling* (Van Nostrand Reinhold).

Zbikowski R., K.J. Hunt, eds (1995), *Neural Adaptive Control Technology* (World Scientific).