IIE Transactions (2004) **36**, 401–415

ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/07408170490426125

Statistical process control via context modeling of finite-state processes: an application to production monitoring

IRAD BEN-GAL* and GONEN SINGER

Department of Industrial Engineering, Tel Aviv University, Ramat-Aviv, Tel Aviv 69978, Israel
E-mail: bengal@eng.tau.ac.il

Received April 2001 and accepted September 2003

Conventional Statistical Process Control (SPC) schemes fail to monitor nonlinear and finite-state processes that often result from feedback-controlled processes. SPC methods that are designed to monitor autocorrelated processes usually assume a known model (often an ARIMA) that might poorly describe the real process. In this paper, we present a novel SPC methodology based on context modeling of finite-state processes. The method utilizes a series of context-tree models to estimate the conditional distribution of the process output given the context of previous observations. The Kullback-Leibler divergence statistic is derived to indicate significant changes in the trees along the process. The method is implemented in a simulated flexible manufacturing system in order to detect significant changes in its production mix ratio output.

1. Introduction and literature review

In many industrial environments process outputs are often adjusted by applying feedback mechanisms and policies. As an example, the temperature in a chemical reactor is controlled by a thermostat; another example is when the allocation of parts to machines is based on machine queue information gathered by sensors. These feedback mechanisms, including closed-loop controllers, often create both linear and nonlinear autocorrelations (hereafter, termed interchangeably, as *interdependence* or *dependencies*) among the observations of the controlled output and increase its variation (Deming, 1986; Boardman and Boardman, 1990). In extreme cases, the structure of these autocorrelations may be established by the dynamics of the observed output trajectories. However, such an identification is not a simple task in noisy environments, or when the feedback mechanism depends on many past observations.

The Statistical Process Control (SPC) methodology has been developed independently of the Engineering Process Control (EPC) approach, which relies heavily on feedback mechanisms. Although both strategies aim for quality improvement, SPC concepts are in sharp contrast with EPC. Whereas EPC actively compensates for process disturbances by performing adjustments constantly, in SPC, a control action is taken only when there is statistical evidence that the process is out of control. This evidence is usually a point outside the limits on a control chart.

Since traditional SPC methods assume an independent process (whereas conventional SPC methods for autocorrelated process assume a known underlying model that is often used to transform the data), a major concern when integrating traditional SPC and EPC techniques is the dependencies among observations that are created by the feedback mechanisms. This point was recently stated in English *et al.* (2001): "when considering the integration of these two approaches, the application of a control chart assumes independence of the observed process observations. The independence assumption is dramatically violated in processes subjected to process control. The chemical and petroleum industries, for example, have traditionally employed classic proportional-integral-derivative (PID) control and have extended this effort in the past decade or two to implement optimal discrete time control systems. By the very nature of PID control, the observations of the process output are highly correlated."

In this paper, we propose a novel approach, termed Context-SPC (CSPC) to monitor a state-dependent process of varying dependence order. Thus, not only that the monitored process be interdependent (either linearly or nonlinearly), but also the nature and the order of this dependence might change in time according to the process state. The suggested method assumes a Markovian property of the process output without requiring a prior knowledge of its transition parameters. Moreover, the CSPC does not assume a closed-form time-series model, which is often required by conventional SPC approaches for dependent processes. The main disadvantages of the CSPC are its limitation to monitor processes with discrete measures over a

*Corresponding author

finite alphabet and the relatively large amount of data that it requires, restricting its on-line analysis capabilities. Albeit, there are many areas to which these limitations do not apply. In particular, as the frequent automated monitoring of processes becomes common, the amount of available observations increases, and at the same time, the interdependence of these observations cannot be ignored. Examples for potential areas for the CSPC approach are described in Ben-Gal, Morag and Shmilovici (2003) that focuses on the statistical properties of the proposed method. This paper, as opposed to Ben-Gal, Morag and Shmilovici (2003), concentrates on the applicability of the CSPC to dependent industrial processes that follow a simple feedback-loop policy. In particular, the applicability of the CSPC to a system with a simple feedback routing policy is analyzed in Section 4. The applicability of the CSPC to more advanced EPC devices, including industrial closed-loop controllers, such as PID, is left for future research.

The employment of SPC methods to autocorrelated processes has been extensively investigated in relation to EPC applications. Box and Kramer (1992) and Box *et al*. (1997) provided an excellent comparison between SPC (which they refer to as *statistical process monitoring*) and EPC (which they refer to as *automatic process control*). They claimed that an effective strategy for on-line quality improvement is to use SPC along with feedback control to monitor the controlled outputs. Montgomery *et al.* (1994) and Montgomery (2000) suggested integrating EPC and SPC to benefit from both approaches. In particular, they suggested using control charts for statistical process monitoring when the control actions are based on feedback control policies. The charts should be applied to the controlled output or to the difference between the controlled output and the target. Such a paradigm simultaneously monitors both the underlying process and the controller itself. Other studies by Gultekin *et al*. (2002) and English *et al.* (2001) suggest further investigation to support such integrations.

Conventional SPC approaches for dependent processes often filter or transform the observations before analyzing and charting their residuals. This requires, of course, a complete knowledge of the controlled process as indicated by Faltin *et al*. (1997) who proposed integrating SPC and EPC particularly when an adequate stochastic model of the process is known. Numerous publications have followed the same principle. Carmen *et al.* (1999) presented a case study on integrating SPC and EPC in an industrial polymerization process. Janakiram and Keats (1998) presented a case study on combining SPC and EPC in a hybrid industry. Vander Weil *et al.* (1992) introduced an Algorithmic Statistical Process Control (ASPC) that represents a proactive approach to quality improvement in a polymerization process. Tsung *et al.* (1999) proposed a strategy to jointly monitor the PID-controlled output and the manipulated inputs using bivariate SPC. Nembhard and Mastrangelo (1998) offered an effective control scheme for a dynamic system in a transient phase. In their study, they implemented a Proportional-Integral (PI) controller as the EPC, and a Moving Center-line Exponentially Weighted Moving Average (MCEWMA) chart as the SPC.

In all the above-mentioned papers, the controlled output was assumed to behave according to a known model, often a time-series model, representing a fixed-order linear dependence among the observations. This is the reason why conventional SPC methods, including those that were designed to handle autocorrelated data, are not suitable, in general, to monitor a varying-length state-dependent process that might result from feedback-controlled processes. In fact, state-dependent processes of varying dependence order have not been previously considered for SPC. Recalling the numerous modeling applications of fixed Markov models, it is hard to ignore the potential applicability of the more general varying-order Markov models that are considered here. Buhlmann and Wyner (1999) further motivate the use of the varying-order models, by noting that the achieved flexibility in parameter dimensionality (by obtaining a continuous parameter dimensionality instead of the 'jumps' in the parameter dimensions of fixed-order models) provides a better trade-off between model bias and model variance.

The rest of the paper is organized as follows. A general SPC scheme for both dependent and independent processes is presented in Section 2. The context-tree model and its construction algorithm, given a stream of observed data, are introduced in Section 3. In Section 4, the suggested method is applied to a simulated Flexible Manufacturing System (FMS). The CSPC reveals the dynamics in the process output and detects when the output is significantly affected by different system characteristics. Section 5 concludes the paper.

2. A general SPC scheme

Two main problems arise when applying SPC to a system output manipulated by a feedback control device. The first problem is how to identify that the process or the controllers deviate from their "expected" behavior. In independent processes, when the observed characteristic behaves according to a known probability distribution, it is relatively easy to identify a change in the process by applying SPC charts to estimated statistics. However, when observations are dependent and follow certain dynamics as a result of feedback control effects, it is much harder to identify a deviation in the process, since both probability type errors increase (Deming, 1986; Boardman and Boardman, 1990; Thomas and Lloyd, 1990). The second problem is how to model the behavior of the stochastic output resulting from the control rules. Alternatively, how to model the relations between time-sequenced observations and allow the order of dependency between current and past observations, to change according to the observed values. Naturally, simple statistical

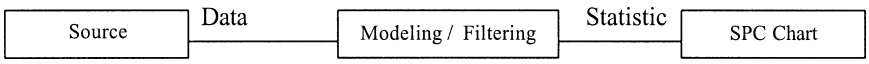


Fig. 1. The general SPC scheme.

measures such as the correlation coefficient are not helpful in understanding the process dynamics of a feedback-controlled system, which are often nonlinear.

A general scheme for the SPC method is shown in Fig. 1. In the first stage, data which may be either independent or dependent, is collected from a general process. In the second stage, statistics that are preferably independent and identically distributed (i.i.d.) having known distribution characteristics are gathered by modeling or filtering operations. In the third stage, these statistics are plotted in SPC control charts. For example, in the first stage of the traditional Shewhart charts, the collected data is i.i.d. In the second and third stages, the $\bar{X}$ statistic, which is assumed to be normally distributed according to the central limit theorem, is estimated and plotted. Similarly, for time-series sources, often an ARIMA model is used in the second stage to filter the data and generate a stream of residuals that are assumed i.i.d. and approximately Gaussian random variables, to which a SPC chart can be applied. For Page's CUSUM methods (Page, 1962) a known distribution is assumed to generate the data and the likelihood ratio statistics are estimated and depicted on the SPC chart.

Following the general SPC scheme, we propose a modeling stage that converts a state-dependent data stream into a statistic with known distribution that can be monitored on the SPC charts. In particular, we apply the *context-tree* model, which was originally proposed for data compression purposes (Rissanen, 1983; Weinberger *et al.*, 1995). The context tree can represent both fixed-order and varying-order Markov models. It has the same advantages as a regular Markov model of order k, where context (that are represented by branches in the tree) act as its states. When representing a varying-order Markov model, the lengths of various branches in the tree are not necessarily equal and one does not need to fix k such that it accounts for the maximum expected dependence in the data (Buhlmann and Wyner, 1999).

Following the second stage in Fig. 1, the Kullback-Leibler (KL) statistic is used in the CSPC to measure the relative distance between *monitored context trees* that represent the dynamics in the system at different monitoring periods, and the *reference context tree* that represents the in-control behaviour of the system. In the next section it is shown that the KL distance statistics are asymptotically i.i.d. and chi-square distributed and hence can be monitored by a SPC control chart.

3. The context tree and the KL measure

3.1. *The context tree*

The context-tree model is an irreducible collection of conditional probabilities that fits an observed sequence of symbols (Rissanen, 1983). We use the context tree to estimate the conditional probability of a symbol in a sequence, given the context of previously observed symbols. The tree is a graph entity that consists of branches and nodes. Branch links are labeled by the symbol types. A context, s, is represented by the path of branch links starting up at the tree root down to some node. The context order is reversed with respect to the order of observance, such that lower nodes represent previously observed symbols. Figure 2 presents the context tree that fits the observed sequence $\{A, A, B, A, B, C, A, B, C, A, B, C\} \times 5$, i.e., five repetitions of the sequence which is read from left to right; meaning that A is observed prior to A which is observed prior to B etc. Accordingly, the context of the last symbol C, is $B, A, C, B, A, C, B, A, B, A, A$. However, modeling such a long context is not necessary since the tree has only two levels, indicating that each symbol depends only on its preceding symbol. The tree consists of three *optimal contexts* that are the shortest significant contexts required by the representing conditional probability of symbols (the exact mathematical condition is given in the next section by Equation (1)).

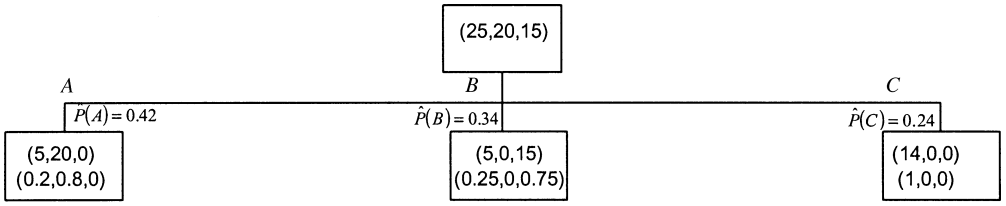


Fig. 2. The context tree that fits the observed sequence $[A, A, B, A, B, C, A, B, C, A, B, C] \times 5$.

In this example, the optimal contests $\{A, B, C\}$ are all the branches from the root up to the leaves. However, in general, the length (depth) of various contexts (branches) do not have to be equal. Each node in the tree contains a vector of conditional frequencies of all symbol types given the respective context. The root node contains the unconditional frequency vector of symbol types in the sequence, thus, given an empty context. Nodes that follow an optimal context contain another vector of the empirical conditional probabilities of symbols given the context. The estimated probabilities of the optimal contexts appear above the nodes. For example, the conditional frequency of symbol C, given the context B is 15, while the conditional frequencies of symbols A and B given that context are five and zero, respectively. Based on these frequencies, the estimated probability to obtain symbols A, B and C given the optimal context B are, $P(A \mid B) = 5/20 = 0.25$, $P(B \mid B) = 0/20 = 0$ and $P(C \mid B) = 15/20 = 0.75$. The estimated probability to obtain the optimal context B is $P(B) = 20/59 = 0.333$ (it is divided by 59 and not by 60 since the first symbol in the string does not have any context).

Note that in general the context tree is not necessarily *balanced* (i.e., not all the branches should have the same length) nor *complete* (i.e., not all the nodes in the tree should have all the d offsprings).

3.2. *The tree construction algorithm*

There exist several algorithms for the construction of a context tree (Rissanen, 1983; Weinberger *et al.*, 1995; Buhlmann and Wyner, 1999; Ben-Gal, Morag and Shmilovici, 2003). In the following, we briefly outline our version of the construction algorithm. For further details the reader is referred to the above literature.

Consider a sequence (string) of observations $x^N = x_1, \ldots, x_N$, with elements x_t $t = 1, \ldots, N$ defined over a finite symbol set, X, of size d. The context tree at time t is denoted by $T(t)$ and based upon all the observations, $x_1, x_2, \ldots, x_t$. Algorithmically, $T(t)$ is obtained iteratively by updating $T(t-1)$ following the last observation x_t. Given an observation string, a context, $s(x^t)$, of the next symbol in the string x_{t+1}, is defined as a reversed string, $s(x^t) = x_t, x_{t-1}, \ldots, x_{\max}\{0, t-k+1\}$ for some $k \geq 0$. The string is truncated since k will be chosen such that the symbols observed prior to x_{t-k+1} do not affect the conditional probability of x_{t+1}. To obtain an *optimal context* one selects the smallest k for which the conditional probability of a symbol given the context is practically equal to the conditional probability of that symbol given the whole sequence, i.e., nearly satisfying:

$$P(x_{t+1} \mid x^t) = P(x_{t+1} \mid s(x^t)). \tag{1}$$

All the optimal contexts belong to the set of optimal contexts, Γ, and defined as the states of the context-tree model.

There are two main stages in the tree construction: (I) a tree growing stage; and (II) a tree pruning stage. These stages can be performed either iteratively, following each new observation, or sequentially, by first constructing the tree and only then pruning it.

In the iterative approach a minimal tree is grown from its root by splitting nodes following each single observation until some goodness-of-split criterion fails to be met. This approach requires less memory space and potentially has a smaller computational complexity, since insignificant branches are never grown. A major problem with the iterative approach is that potentially significant splitting might exist on lower branch levels that are pruned at earlier stages. The sequential approach, which we apply here, constructs a context tree up to a predetermined maximal order (depth) L from all the data string and only then prunes it, upwards, by trimming leaves that are statistically similar to their parent nodes. Although this approach requires more memory space (the tree is represented by an $L \times d$ matrix), it keeps a significant branch even if the significant splitting is located towards the end of the branch. The complexity of this algorithm is $O(N \log N)$ given an input string of length N (Rissanen, 1999). In practical terms, the construction of a tree from a string with 2000–5000 data points took less than 5–7 seconds on a Pentium 3 PC. Further discussion on various construction versions can be found in Buhlmann and Wyner (1999) and Ben-Gal, Morag and Shmilovici (2003).

3.2.1. *Stage 1: Tree growing and updating*

1. Define the initial tree $T(0)$

Construct $T(0)$, the initial context tree at time $t = 0$, as a root node with all symbol counters equal to zero. The initial symbol counters are denoted by $n(x \mid s_0) = 0 \, \forall x \in X$, where s_0 denotes the initial empty context.

2. Constructing the tree $T(t)$

Recursively, having constructed the tree $T(t-1)$ from x^{t-1}, read the symbol x_t. Climb the tree according to the path defined by previously observed symbols $x_{t-1}, x_{t-2}, \ldots$, and increment the count of this symbol, $n(x_t \mid x_{t-1}, x_{t-2}, \ldots)$, by one for every node visited until the deepest node is reached. If the last updated count is at least one and its depth is less than L, a maximal predetermined tree depth, create a new node, with all symbol counts equal to zero, except for the count associated with x_t which is set to one.

Illustrative example: We demonstrate the construction of a context tree for an example string, $x^4 = A, A, B, A$ which represent the first four symbols of the observed sequence in Section 3.1. The string is composed of a symbol set of $d = 3$, and its length is $N = 4$. For the purpose of illustration, we assume no limitation on the maximal tree depth, i.e., $L \geq 4$. Figure 3 presents the tree growing and counter update process.

3.2.2. *Stage 2: Tree pruning*

Two pruning rules are applied during the tree construction. The first rule determines the maximal depth of the

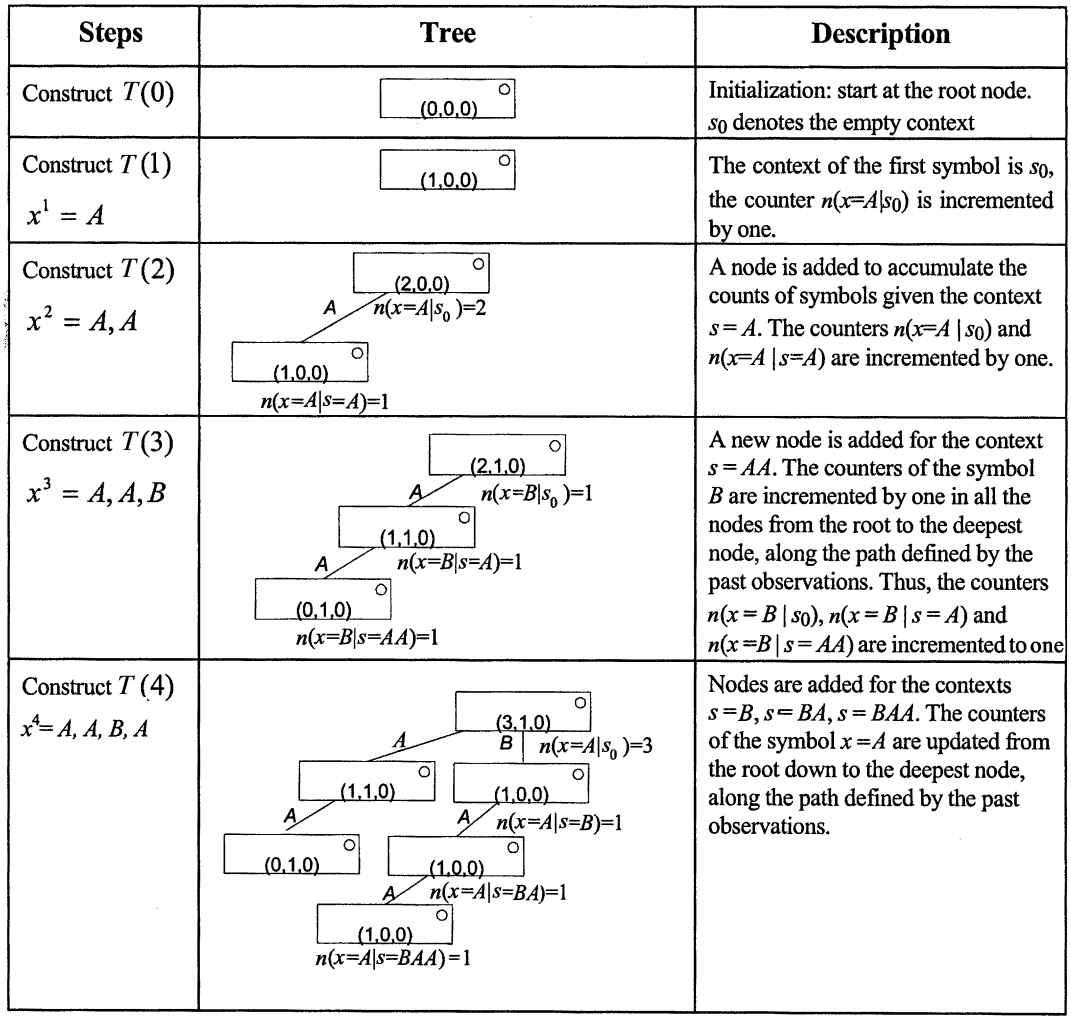


Fig. 3. Tree growing and counter updating stage in context algorithm for string $x^4 = A, A, B, A$.

tree according to the amount of available data. The second rule is based on a relative-entropy measure (Rissanen, 1983; Weinberger *et al.*, 1995) to prune insignificant nodes to obtain the shortest contexts that satisfy Equation (1).

Pruning rule 1: When following the sequential approach, as done here, this rule is applied at the beginning of stage 1 to determine the maximal depth of the tree, L. This rule guarantees that if N available data points are divided evenly among all d^L leaves of a complete and balanced tree, then each of the leaves would contain at least n_a data points, i.e., requiring that $n_a \times d^L \leq N$. Selecting the minimal value $n_a = 1$, we obtain $L \leq \log(n)/\log(d)$. When following the iterative approach, this rule is applied before a new node, w, is added to the tree. It bounds the depth of that node, denoted by $|w|$, by the logarithm (to the base d) of the length of the observed string until that point, i.e., $|w| \leq \log(t+1)/\log(d)$.

Pruning rule 2: This rule uses the relative entropy to measure the distance between the symbols' distributions at a descendant node, sb $\forall b \in X$, and its parent node s. The descendant node is pruned if this measure is larger than a penalty cost for growing the tree (i.e., of adding a node). The driving principle is to prune a descendant node having a symbol distribution which is similar to that of its parent node in terms of the KL divergence. In particular, we calculate $\Delta_N(sb)$, the (ideal) *code length difference* of the descendant node sb, $\forall b \in X$:

$$\Delta_N(sb) = \sum_{x \in X} n(x\,|\,sb) \log\left(\frac{\hat{P}(x\,|\,sb)}{\hat{P}(x\,|\,s)}\right), \tag{2}$$

and require that $\Delta_N(sb) > c(d+1)\log(t+1)$, where logarithms are taken to base 2, and c is the pruning constant tuned to process requirements (with default $c = 2$ as suggested in Weinberger *et al.* (1995)). This process is extended to the root node with $\Delta_N(x^0) = \infty$.

Illustrative example: Figure 4 presents the pruning stage for the counter context tree, which is constructed in Fig. 3.

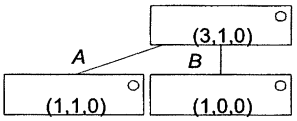
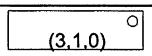
Rule	Context tree	Description
Rule 1		Rule 1: Maximum tree depth $\leq \log(N)/\log(d) = \log(4)/\log(3)=1.26$, i.e., the maximum tree depth is of level 1, all nodes of level 2 and below are trimmed.
Rule 2:		Rule 2: for the rest of the nodes in level 1 and the root, apply pruning rule 2. The threshold value is: $2\,(3+1)\log(4+1)=18.58$. For each of the descendant nodes: $\Delta_4(sb = s_0A) = 1\times\log\left(\frac{0.5}{0.75}\right)+1\times\log\left(\frac{0.5}{0.25}\right)+0 = 0.415$ $\Delta_4(sb = s_0B) = 1\times\log\left(\frac{1}{0.75}\right)+0+0 = 0.415$ Accordingly, both nodes are pruned.

Fig. 4. Pruning stage in context algorithm for the string $X^4 = A, A, B, A$.

3.2.3. *Stage 3: Estimating the probabilities of optimal contexts and symbols*

Once the tree is pruned, estimate the probabilities of the optimal contexts, $\hat{P}(s)$, $s \in \Gamma$ by their frequencies in the string:

$$\hat{p}(s) = \begin{cases} 0 & \text{if } sb \in T(t)\ \forall b \in X, \\ \dfrac{n(s)}{\sum_{s' \in T(t)} n(s')} = \dfrac{\sum_{x\in X}\left(n(x \mid s) - \sum_{b\in X} n(x \mid sb)\right)}{\sum_{s'\in T(t)}\sum_{x\in X}\left(n(x \mid s') - \sum_{b\in X} n(x \mid s'b)\right)} & \text{else, for } s \in T(t), \end{cases} \tag{3}$$

where $n(x \mid s)$ is a symbol counter representing the conditional frequency of the symbol $x \in X$ following the context s; $n(s)$ is the sum of the counters of symbols that belong to the context s but not to a longer context sb, $b \in X$ and $\{s' \in T(t)\}$ is the set of all the contexts in the tree. Any internal node in the tree satisfying the first line in Equation (3) represents a nonoptimal context, since it is a part of a longer optimal context. All the contexts with nonzero probabilities (second line in Equation (3)) belong to the set of optimal contexts, Γ, that has a finite state space, $|\Gamma| = S$. The second line in Equation (3) normalizes the probability estimates of the optimal contexts, s′, to compensate for the fact that the first symbols in the string do not have long enough contexts. Therefore, the summation of counters in descendent nodes is not necessarily equal to summation of counters in the parent node (as exemplified at the end of Section 3.1). Following Equation (3), an *optimal context* can be either a path to a leaf (a node with no descendants) or a *partial leaf*–a node which is not a leaf; however, for certain symbol(s) (but not for all the symbols) its path defines an optimal context satisfying Equation (3) (Ben-Gal, Morag and Shmilovici, 2003). This important concept decreases the statistical error rates and extends the algorithm Context proposed by Rissanen (1983) and Weinberger *et al.* (1995) that considered only the leaves in the tree as contexts.

Once Γ is determined, we estimate the conditional probabilities of symbol types $x \in X$ given an optimal context $s \in \Gamma$ by their respective frequencies:

$$\hat{P}(x \mid s) = \frac{n(x \mid s) - \sum_{b\in X} n(x \mid sb)}{\sum_{x\in X}\left(n(x \mid s) - \sum_{b\in X} n(s \mid sb)\right)}, \forall x \in X, \forall s \in \Gamma. \tag{4}$$

A predictive version of Equation (4) is given in Weinberger *et al.* (1995) and can be used for systems where there is a positive probability of obtaining any symbol given any context. Finally, following Equations (3) and (4), the context tree can be estimated as a collection of joint probability distributions of contexts and symbols, $\hat{P}(x, s) = \hat{P}(x \mid s) \times \hat{P}(s)$.

Illustrative example: Applying Equation (3) to the pruned counter context tree presented in Fig. 2 results in the estimated probabilities of three optimal contexts:

$$\{\hat{P}(A), \hat{P}(B), \hat{P}(C)\} = \left\{\frac{25}{59}, \frac{20}{59}, \frac{14}{59}\right\}.$$

All the contexts, in this case, are leaves. Note that the root node is not an optimal context since all its symbols are

contained in its descendants nodes. The estimated conditional probabilities of symbols given contexts are:

$$\{\hat{P}(A \mid A), \hat{P}(B \mid A), \hat{P}(C \mid A), \hat{P}(A \mid B), \hat{P}(B \mid B), \hat{P}(C \mid B), \hat{P}(A \mid C), \hat{P}(B \mid C), \hat{P}(C \mid C)\} = \left\{\frac{5}{25}, \frac{20}{25}, \frac{0}{25}, \frac{5}{20}, \frac{0}{20}, \frac{15}{20}, \frac{14}{14}, \frac{0}{14}, \frac{0}{14}\right\}.$$

These estimates are found by applying Equation (4) to the *counter context tree* presented in Fig. 2.

3.3. *Applying the KL statistic to the context-tree model*

Kullback (1959) proposed a measure for the "relative distance" or the discrimination between two probability mass functions with the same range, later known as the Kullback Leibler (KL) measure. The KL measure is applied at any monitoring point to detect the relative distance between the monitored distribution of symbols and contexts $\hat{P}_i(x, s)$, which is estimated from N observations, and the in-control reference distribution $\hat{P}_0(x, s)$. The reference distribution is either driven analytically (as done in Ben-Gal, Morag and Shmilovici, 2003) for the case of buffer monitoring in a manufacturing line with known production probabilities) or can be estimated by applying the context algorithm to a sequence of size N_0. It is then denoted by $\hat{P}_0(x, s)$. The sequence lengths in both stages, N and N_0, have to adhere to the chi-square sampling principle suggested by Cochran (1952). This principle requires that at least 80% of the sampling bins (corresponding in this case to the nonzero conditional probabilities of symbols given optimal contexts) contain at least four data points. Convergence conditions and further analysis for the reference distribution are given in Ben-Gal, Morag and Shmilovici (2003).

The KL measure between the two trees can be decoupled into two terms: one measuring the relative distance between the distribution of optimal contexts, and the other measuring the relative distance between the conditioned distribution of symbols given an optimal context. Moreover, it has been shown (Ben-Gal, Morag and Shmilovici, 2003; Ben-Gal, Shmilovici and Morag, 2001) that if the two trees are estimated from the same stationary process, their KL statistic is asymptotically (in N which is the length of the monitored string) chi-square distributed with $2(Sd - 1)$ degrees of freedom. Thus, the KL measure depends on the number of symbol types, d, and the number of optimal contexts, S:

$$K(\hat{P}_i(x, s), \hat{P}_0(x, s)) = \sum_{s\in\Gamma} \hat{P}_i(s) \log \frac{\hat{P}_i(s)}{\hat{P}_0(s)} + \sum_{s\in\Gamma} \hat{P}_i(s) \sum_{s\in X} \hat{P}_i(x|s) \log \frac{\hat{P}_i(x|s)}{\hat{P}_0(x|s)}, \to \frac{1}{2N}\left(\chi^2_{2(S-1)} + \chi^2_{2S(d-1)}\right) = \frac{1}{2N}\chi^2_{2(Sd-1)}. \quad (5)$$

A numerical analysis for the asymptotic convergence rate of the KL measure as a function of N for a given value of α was obtained in Ben-Gal, Morag and Shmilovici (2003) and can be used to determine the minimal string length which is required. In general, the minimal string length depends on the number and lengths of the contexts in the tree; processes that are less interdependent require a smaller string length. Evidently, a high value of the required string length undermines the effectiveness of the on-line monitoring.

Finally, for a predetermined type-I error probability, α, the control limits of the KL statistic (multiplied by twice the monitored string length) are:

$$0 \le 2N \;\; K(\hat{P}_i(x, s), \hat{P}_0(x, s)) \le \chi^2_{2(Sd-1),1-\alpha}. \quad (6)$$

Thus, the Upper Control Limit (UCL) is the $100(1 - \alpha)$ percentile of the chi-square distribution with $2(Sd - 1)$ degrees of freedom.

3.4. *The CSPC monitoring procedure*

Following the above results, a brief description of the CSPC steps is provided. Note that the CSPC method follows the general SPC scheme presented in Fig. 1.

Step 1. Obtain a reference tree, $P_0(x, s)$, either analytically or by estimating it from a long string of symbols.

Step 2. Collect a sequence of N observations from the monitored source in any monitoring point of time and construct a monitored tree $\hat{P}_i(x, s)$. Each such sequence is called a "run" and contributes one monitoring point to the CSPC chart. Following the minimum discrimination information (MDI) principle (Kullback, 1978; Alwan *et al.* 1998), use the monitored observations to update the values of the counters in the reference tree, and estimate the probability measures of context and symbols in that tree.

Step 3. Compute the KL statistic measuring the relative distance between the monitored tree, $\hat{P}_i(x, s)$, and the reference tree, $P_0(x, s)$. The KL measures are i.i.d. and chi-square distributed, if both trees are generated by the same stationary process.

Step 4. Plot the KL measures on a simple control chart, with respect to the UCL given in Equation (6). KL measures large than the UCL trigger an out-of-control signal. Once an out-of-control signal has been detected, the KL measures can be decomposed with respect to optimal contexts (see Equation (5)) to locate those contexts that contribute to the high value of the KL measure.

Step 5. For further monitoring, increment $i = i + 1$ and return to Step 2, otherwise stop.

4. Monitoring of a simulated FMS

4.1. *Purpose of the study*

Traditional SPC methods are often applied to monitor product characteristics and process attributes. These measures provide information on certain quality criteria but overlook general output criteria such as *throughput* or the *Production Mix Ratio* (PMR), which is the ratio among

produced quantities of different part types. A possible reason why SPC for such a criterion is missing can be the complex and nonlinear patterns of the PMR that often cannot be described by a known stochastic process (e.g., an ARIMA model). The PMR process cannot be controlled even by conventional SPC methods for autocorrelated data, as seen in Section 4.3. On the other hand, there are good reasons to monitor such a criterion. The PMR is often a direct measure for the suitability of the feedback-control policies that are implemented in the production system. For example, by monitoring the PMR, one can examine whether the underlying routing policy of parts to various machines is properly functioning. In real-life settings, many of the control rules are implemented (and are actually "hidden") by long codes of commercialized software packages that cannot be monitored directly. This is true also in cases where the software has been changed or misimplemented unintentionally. Monitoring the PMR provides an unbiased way to ensure that these rules are properly functioning, particularly if the process changes frequently. Moreover, since various machines and devices in the production line affect the PMR, it can be used as a direct measure for the interactions among them. These interactions are of particular importance in a FMS, as considered here. Note that several state-dependent processes might occur in systems where SPC is traditionally applied.

The purpose of this simulation study is to demonstrate the abilities of the CSPC (in comparison with conventional SPC methods) in monitoring the PMR patterns and indicating whether a significant change has occurred. The section is organized as follows. In Section 4.2 we describe the simulated FMS and our underlying assumptions. In Section 4.3 we show that conventional SPC methods for autocorrelated processes fail to monitor the PMR. In Section 4.4, we follow the suggested CSPC approach. In particular, we examine the KL distance statistics between a series of monitored context trees and a reference context tree. The trees are constructed from data strings that indicate the type and the order of the produced parts under various production conditions. The KL values are plotted in control charts against the UCL given in Equation (6). KL measures larger than the UCL can trigger an out-of-control signal that enables the inspector to search for the origin of the alteration.

4.2. *Description of the FMS simulation model*

The considered FMS cell is composed of three CNC machining centers, one universal loading station and one unloading station, all having an infinite capacity. Figure 5 shows the FMS layout.

Three part types are produced in the cell simultaneously. The processing times are assumed to be known deterministically, since all the machining operations are computer-numerically controlled. More than one machine is capable of performing the same operations, therefore, each part type has alternative routes. The alternative routes and processing times of operations of different part types are listed in Table 1. For example, notice from Table 1 that part type *A* has two alternative routes. In the first route, the part is processed for 3 minutes in the loading station, 10 minutes in MC1, 11 minutes in MC3 and finally 3 minutes in the downloading station. In the second route, operations 1, 2 and 4 are performed in the same machining centers as before with the same processing times. The difference occurs in the third operation, which is processed for 14 minutes in MC2 instead of MC3.

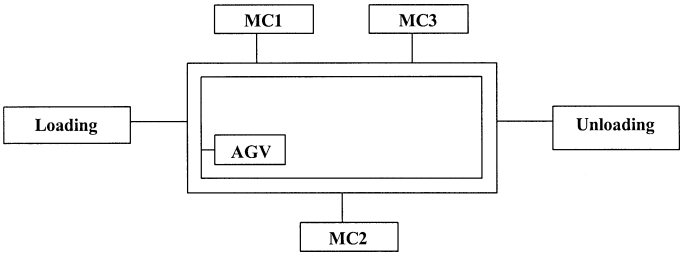


Fig. 5. Schematic illustration of the FMS.

Two different input probability ratios are considered and presented in Table 2. An input probability ratio is the ratio between the different part types that enter the system. In practice, the input ratio can represent the ratios among different raw materials or WIP, each of which is associated with a specific part type. Alternatively, the input probability ratio can represent the ratios of job orders that enter the system. For example, the first input probability ratio in Table 2 represents a stochastic process with a probability of 0.3 to obtain a job order for part type *A*, a probability of 0.2 to obtain a job order for part type *B*, a probability of 0.3 to obtain a job order for type *C* and a probability of 0.2 to obtain a job order for part type *D*.

We implement a routing control policy with a feedback-loop which is a simple version of the *Dynamic Alternative Routings* (DAR) proposed by Chan (1998). The implemented routing policy delivers the processed part to the next alternative machine with the lowest number of entities in its queue. If several buffers have the same number of entities in their queue (including empty buffers), the routing rule

Table 1. Alternative routes and processing times for different part types (cell entries indicate the processing time in minutes)

		Machine centers				
Part type	*Route*	*Load station*	*MC1*	*MC2*	*MC3*	*Unload station*
A	1	3	10		11	3
	2	3	10	14		3
B	1	5	10			5
	2	5		8		5
C	1	2	30			2
	2	2		34		2
D	1	3		7	13	3
	2	3	10		13	3

Table 2. Input ratios for different part types

	Part type				
Input probability ratio	*A*	*B*	*C*	*D*	*Sum*
1	0.3	0.2	0.3	0.2	1
2	0.4	0.1	0.3	0.2	1

delivers the part to the buffer of the machine on which the lowest processing time is required with respect to the allocated part type. Thus, the routing controller takes an action only after observing the state of the system, which is defined by the queue levels. This action, which is not expressed in closed-form equations, is performed at each part delivery; thus, it is driven by discrete events that are not necessarily equi-spread along the time horizon. Although this feedback loop is much simpler than most of the applied EPC methods (such as PID and model predictive control), it creates an interdependent PMR process that cannot be monitored by conventional SPC methods for autocorrelated processes, and therefore serve us for illustration purposes. For other details on the DAR characteristics in comparison to other routing methods the reader is referred to Chan (1998).

4.2.1. *Summary of assumptions and operational control rules*

The following list summarizes the assumptions implemented in the simulation model:

1. The sequence of the operations for each part type is known (see Table 1).
2. The processing times for each part type are known (see Table 1).
3. The input ratio of different part types is stochastic (see Table 2).
4. The modified DAR is implemented as the routing (feedback) policy.
5. All local buffers in the machine centers have the same FIFO dispatching rule.
6. The capacities of the universal loading station and the local buffers are infinite.
7. The traveling times of raw materials between stations are negligible.
8. A cycle is concluded when a total of $N > 930$ parts arrives to the unloading station.

4.3. *Implementation of conventional SPC methods*

The ARIMA family of models is widely applied for the representation and filtering of autocorrelated processes. If the process is well described by an ARIMA model, then the model filtering of a dependent process yields independent and approximate Gaussian residuals, to which traditional SPC can be applied. Moreover, it has been shown, for example in Box and Jenkins (1976) and in Apley and Shi (1999), that simple ARIMA models, such as AR(1) or IMA(1,1), can effectively filter a wide range of autocorrelated processes even if they do not fit the model exactly.

In this section it is shown that ARIMA models are generally inadequate for the monitoring of nonlinear state-dependent processes with discrete measures, such as the PMR process considered here, or the buffer levels that are considered in Ben-Gal, Morag and Shmilovici (2003). In particular, we checked the applicability of a wide range of ARIMA models to the monitoring of the above FMS example, by fitting models to the time series $z_t = 1$ (part type A observed), $z_t = 2$ (part type B observed), ..., $z_t = 4$ (part type D observed).

We started by simulating the in-control process, which is based on the routings and processing times number 1 (Table 1), and input probability ratio number 1 (Table 2). Table 3 presents the best found ARIMA models, in terms of their type-I statistical errors and the Average Run Length (ARL) as analyzed using the Statgraphics software package. The columns of the table are, respectively: (i) the ARIMA model; (ii) the subgroup size N, (i.e., fitting a model and then applying an $\bar{X}$ chart to a subgroup of N residuals); (iii) the Mean-Square Error (MSE); (iv)

Table 3. Analysis of different ARIMA methods for the in-control stage

ARIMA model	*N*	*MSE*	*UCL* $\alpha = 5\%$	*Beyond UCL*	$\hat{\alpha}$	$\hat{ARL}$	*Model parameters*
ARIMA(1,0,0)	5	0.28	0.46	75	0.38	2.5	$\hat{z}_t = 3.37 + 0.02z_{t-1}$
ARIMA(2,0,1)	5	0.28	0.46	82	0.41	2.09	$\hat{z}_t = 2.67 + 0.02z_{t-1} - 0.12z_{t-2} + 0.01\varepsilon_{t-1}$
ARIMA(2,0,2)	10	0.14	0.23	113	0.57	1.8	$\hat{z}_t = 2.39 - 0.03z_{t-1} + 0.05z_{t-2} - 0.04\varepsilon_{t-1} + 0.05\varepsilon_{t-2}$
ARIMA(0,0,2)	10	0.14	0.23	114	0.57	1.76	$\hat{z}_t = 2.41 - 0.07\varepsilon_{t-1} + 0.1\varepsilon_{t-2}$
ARIMA(1,0,2)	10	0.14	0.23	113	0.57	1.78	$\hat{z}_t = 2.51 - 0.04z_{t-1} - 0.03\varepsilon_{t-1} + 0.09\varepsilon_{t-2}$
ARIMA(1,0,4)	10	0.14	0.23	112	0.56	1.76	$\hat{z}_t = 2.53 - 0.05z_{t-1} - 0.02\varepsilon_{t-1} + 0.11\varepsilon_{t-2} - 0.01\varepsilon_{t-3} - 0.02\varepsilon_{t-4}$
ARIMA(1,2,1)	10	0.19	0.27	112	0.56	2.05	$\hat{z}_t - 2z_{t-1} + z_{t-2} = z_{t-1} - 0.66\varepsilon_{t-1}$
ARIMA(2,0,1)	20	0.07	0.12	134	0.67	1.44	$\hat{z}_t = 2.67 + 0.01z_{t-1} - 0.12z_{t-2} + 0.05\varepsilon_{t-1}$
ARIMA(1,0,0)	25	0.06	0.09	153	0.77	1.31	$\hat{z}_t = 2.44 - 0.02z_{t-1}$
ARIMA(1,0,2)	25	0.06	0.09	153	0.77	1.31	$\hat{z}_t = 2.42 - 0.01z_{t-1} - 0.01\varepsilon_{t-1} - 0.004\varepsilon_{t-2}$
ARIMA(1,0,4)	25	0.06	0.09	154	0.77	1.28	$\hat{z}_t = 2.37 + 0.01z_{t-1} - 0.03\varepsilon_{t-1} + 0.001\varepsilon_{t-2} + 0.02\varepsilon_{t-3} - 0.03\varepsilon_{t-4}$
ARIMA(1,2,1)	25	0.08	0.11	139	0.7	1.56	$\hat{z}_t - 2z_{t-1} + z_{t-2} = z_{t-1} - 0.51\varepsilon_{t-1}$

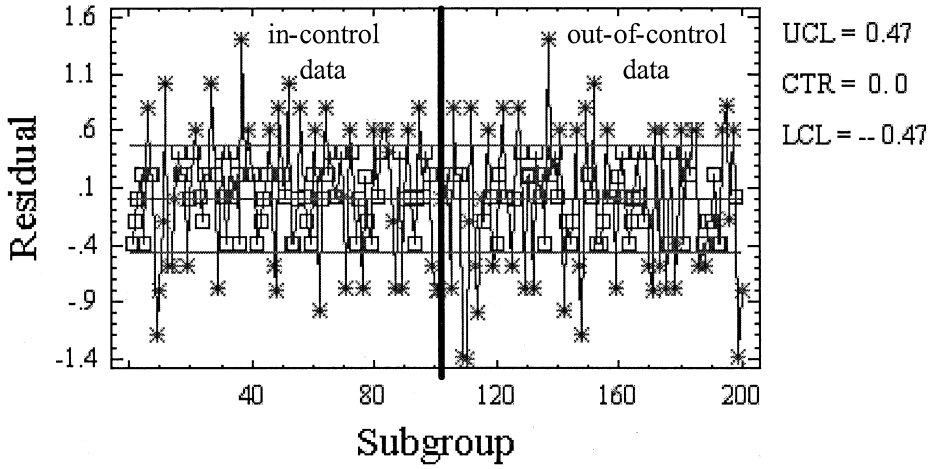


Fig. 6. AR(1) chart analysis with subgroup 5 applied to 500 in-control data points and 500 out-of-control data points. The limits are based on $\pm 1.96 \times \sqrt{MSE/5}$.

the UCL which is determined as $Z_{0.975}\sqrt{MSE/N}$, where $Z_{0.975}$ denotes the 0.975 percentile of the standard normal distribution and equals 1.96; (v) the actual number of runs that fell beyond the UCL; (vi) the estimated type-I statistical error, calculated by dividing the number of runs that fell beyond the UCL by 200 which is the number of analyzed subgroups; (vii) the estimated ARL which was computed directly from the control charts; and (viii) the estimated model parameters. In general, in all the ARIMA charts, including ones that are not presented here, we obtained high values of type-I statistical errors and low ARL values. As seen in the next section, the CSPC method resulted in a much better statistical performance.

Figure 6 presents the control chart of the residual AR(1) with $N = 5$; the best found model in terms of its ARL and the type-I statistical error. The first 100 points in the chart are generated by the in-control process, while the second 100 points are generated by the out-of-control process, which is based on input probability number 2 in Table 2. Note that more than 30% of the first 100 points are marked erroneously (by the star signs) as out of control. Moreover, it is impossible to distinguish between the two processes and indicate a change point.

Since the suggested CSPC requires a large subgroup size, a justified question is whether the performance of the ARIMA models can be improved by increasing the subgroup size. Note from Table 3 that regardless of the model, an increase of the subgroup size resulted in a poorer ARL. In general, it is evident that better estimation of the model parameters, as a result of a larger sample size, does not improve the performance of a wrong model. This can be illustrated by examining the simple AR(1) model, which is fitted for the in-control process as follows:

$$\hat{z}_t = \mu(1-\theta_1) + \theta_1 z_{t-1} = 2.42(1-0.02) + 0.02 z_{t-1} = 2.37 + 0.02 z_{t-1}, \quad (7)$$

where $\mu = 2.42$ is the mean of the process and $\theta_1 = 0.02$ is its first-order autocorrelation. Evidently, since the process is not linear, the autocorrelation parameter tends to zero as N increases and the expected response depends only on the process mean. The residuals of an AR(1) filtering fluctuate around the mean value. As one increases the sample size, the sample mean converges faster to the process mean. However, the nonlinear dependencies in the process do not play any role in the monitoring of such a filtered process and accordingly the ARL decreases. In fact, any modification of the in-control process that maintains the same mean value cannot be identified by the AR(1) control chart, regardless of the subgroup size. For the purpose of illustration, we generated a deterministic out-of-control process by replicating the string {2,1,4,1,4,1,4,1,4,1,4,2} over and over again. Since the mean of the deterministic process ($\mu = 2.417$) is almost identical to the mean of the in-control process but derived by a shorter string, a high type-II probability error is obtained with a small sample size.

Figure 7 presents the average AR(1) estimation error, $\bar{\varepsilon}(N) = (\hat{z}_t - z_t)/N$, around the mean value for both the in-control process (dotted line) and the deterministic out-of-control process (bold line). As seen, the average estimation error for the out-of-control process converges (in the sample size) to zero much faster than the in-control process, leading to a high type-II probability error. Note, for comparison, that the type-II probability error for the deterministic out-of-control process is effectively zero when applying the suggested CSPC; the obtained KL statistic between the tree based on the deterministic tree and the reference tree is equal to 4800, whereas the UCL ($\alpha = 0.05$) is equal to 62.82. A similar phenomenon occurs when applying other ARIMA models to monitor nonlinear state-dependent processes; the average error actually reflects the convergence rate of the model parameters regardless of the data interdependencies and the model correctness.

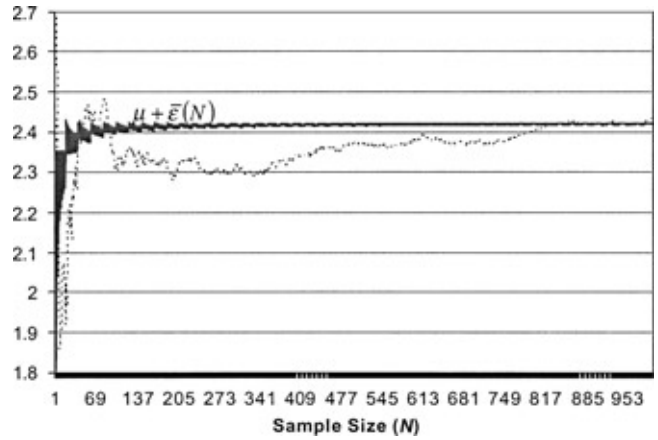


Fig. 7. The average AR(1) estimation error (around the process mean) as a function of the subgroup size for both the in-control process (dotted line) and the deterministic out-of-control process (bold line).

4.4. *Implementation of the CSPC approach*

The purpose of this section is to follow the suggested CSPC and illustrate that it can efficiently identify changes in a state-dependent stochastic environment. The simulated study was performed in four steps. At each step we simulated the FMS cell under different production conditions. Each step consisted of 100 simulation runs generated from different seeds under the same conditions. In the first step, the FMS model was based on the routings and processing times number 1 (Table 1), and input probability ratio number 1 (Table 2). In the second step, we maintained the same routings and processing times but used input probability ratio number 2 (Table 2). In the third step, we simulated the same system as in the second step, however, we updated the reference tree to illustrate that the proposed monitoring method has a "learning" ability once a change has been identified and a new reference tree represents the modified process. In the fourth and last step, we changed the routings of the part types by using routings number 2 (Table 1). The simulated results are presented in what follows.

4.4.1. *First step: Monitoring the reference in-control system*

Figure 8 shows the initial reference context tree $\hat{P}_0(x, s)$. The tree is multileveled with $d = 4$ symbols, $X = \{A, B, C, D\}$, each of which represents a different part type. It was obtained by applying the context algorithm to an initial sequence with $N_0 = 931$ observations of parts that were produced by the simulated system. A string length of 931 observations adheres to the chi-square sampling principle suggested by Cochran (1952): note from Fig. 8 that 22 bins out of $S \times d = 6 \times 4 = 24$ bins (around 92% of the bins) contain at least four data poins. When possible, we suggest using a larger sample size than the one defined by Cochran (1952) in order to better estimate the reference distribution. Some numerical methods to determine N_0 are suggested in Ben-Gal, Morag and Shmilovici

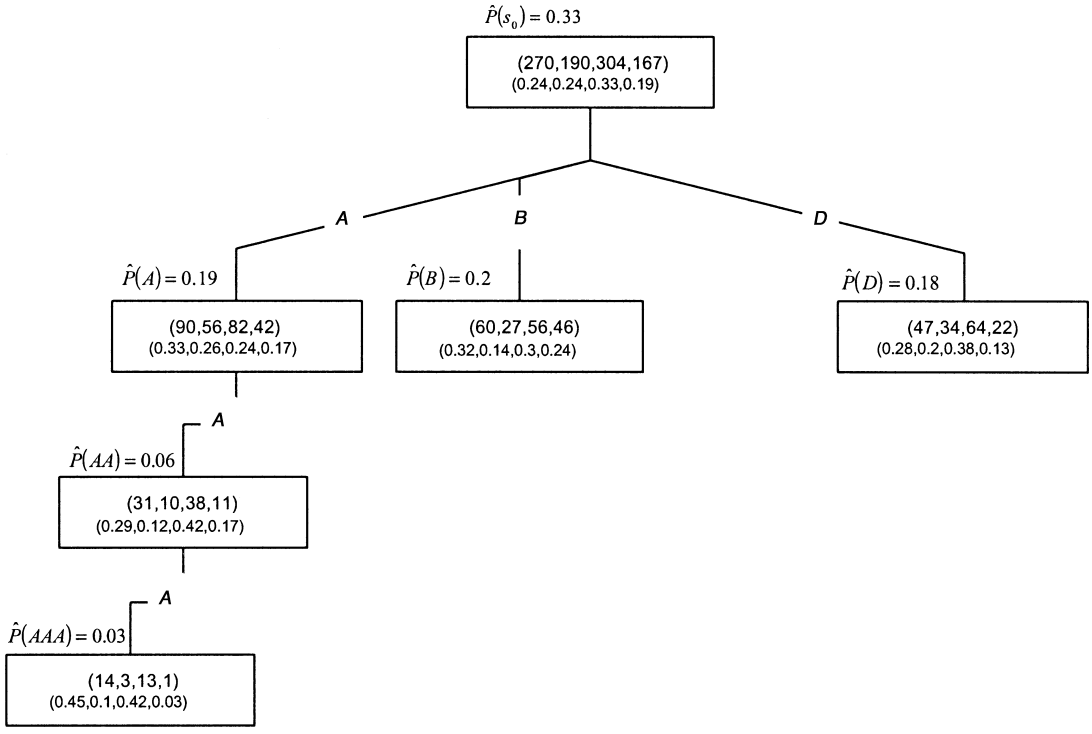


Fig. 8. The initial reference context tree $\hat{P}_0(x, s)$ with four symbols and six optimal contexts.

(2003). The structure of the tree in Fig. 8 represents the varying length dependencies among the observations. As seen, the empirical conditional distribution of symbols, following the contexts *A, AA* and *AAA* are significantly different from each other, whereas the empirical distribution of symbols following *B* or *D* are conditioned only on that single symbol. The tree has $S = 6$ optimal contexts, $\{s_0, A, B, D, AA, AAA\}$, where s_0 denotes the empty context represented by the tree root. All the nodes in the tree represent optimal contexts and contain both the conditional frequencies of symbols given optimal contexts and the empirical conditional distribution of the symbols. For example, the empirical conditional distribution of symbols given the optimal context *AA* is, respectively, $P_0(A, B, C, D \mid s = AA) = (0.29, 0.12, 0.42, 0.17)$. The root corresponds to the empty context and presents the unconditional frequencies of part types. The probabilities of the optimal contexts are listed above the corresponding nodes.

At this stage, we checked whether the suggested CSPC procedure would indicate that the system is stable, thus, would result with a small type-I statistical error. We used different seeds to generate 100 simulation runs, each of which contain a sequence of 931 produced parts. The context algorithm was then applied to each sequence to construct a series of monitored trees, $\hat{P}_i(x, s), i = 1, \ldots, 100$. Following the MDI principle, these sequences updated the counter values in the monitored trees to evaluate the probability estimates for symbols and contexts. Next, a series of KL statistics (multiplied by $2N$, where $N = 931$ is the size of the sequence), measuring the distance between the monitored trees and the reference tree, were computed and plotted against the UCL. The UCL was calibrated to a type-I error probability of $\alpha = 0.05$, resulting in $\chi^2_{2(Sd-1),1-\alpha} = \chi^2_{46,0.95} = 62.83$.

The first 100 KL values are plotted in the left-hand side of Fig. 9 against the UCL to monitor the in-control system prior to any change. As seen, all but one point lie below the UCL with an estimated ARL close to 50.

4.4.2. *Second step: Changing the input probability ratio*

At this step, we used input probability ratio number 2 (Table 2), while keeping all the other system parameters unchanged. The context algorithm was applied to the next 100 runs to generate the monitored trees, $\hat{P}_i(x, s), i = 101, \ldots, 200$, each of which are based on a sequence of 931 parts. The KL statistics (multiplied by $2N$) were computed and compared to the same UCL, as done in the previous step. The last 100 points in Fig. 9. represent the monitoring of the out-of-control process that resulted from the modification in the input probability ratio. As seen, all the values lie above the UCL.

4.4.3. *Third step: Modifying the reference tree*

In this step we illustrated the learning ability of the proposed CSPC after it has indicated that a significant change has occurred in the system. This is achieved by updating the reference tree following the indicated change in the input probability ratio.

Figure 10 shows the updated reference tree, $\hat{P}^*_0(x, s)$, which is based on input probability number 2. The tree was obtained by applying the context algorithm to a sequence of 931 observations of the parts that were produced by the modified simulated system. Defining this tree as the new in-control reference tree emphasized the capability of the CSPC to capture the new dynamics in the system following a change. One hundred simulation runs that were based on different seeds generated the monitored trees, $\hat{P}_i(x, s), i = 201, \ldots, 300$. The CSPC

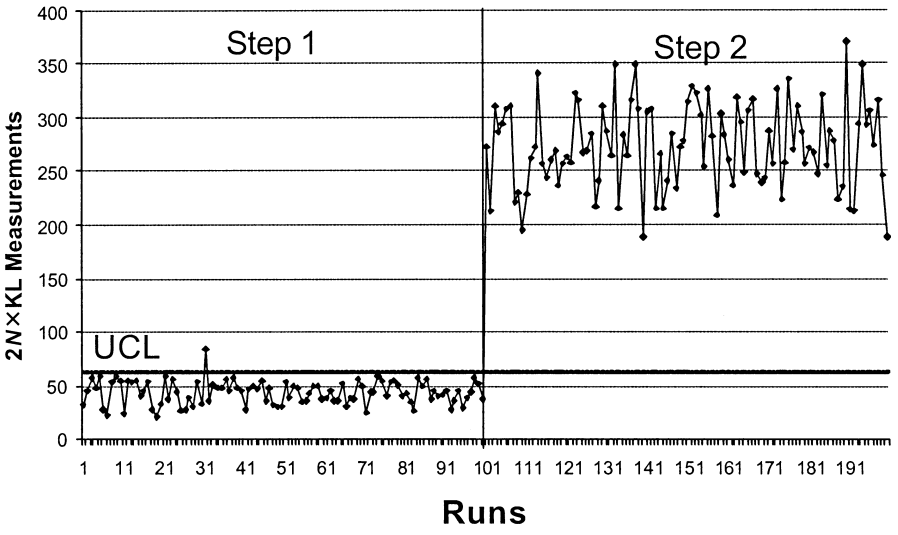


Fig. 9. The CSPC control chart indicating a change in production ratio in Step 2 after 100 runs.

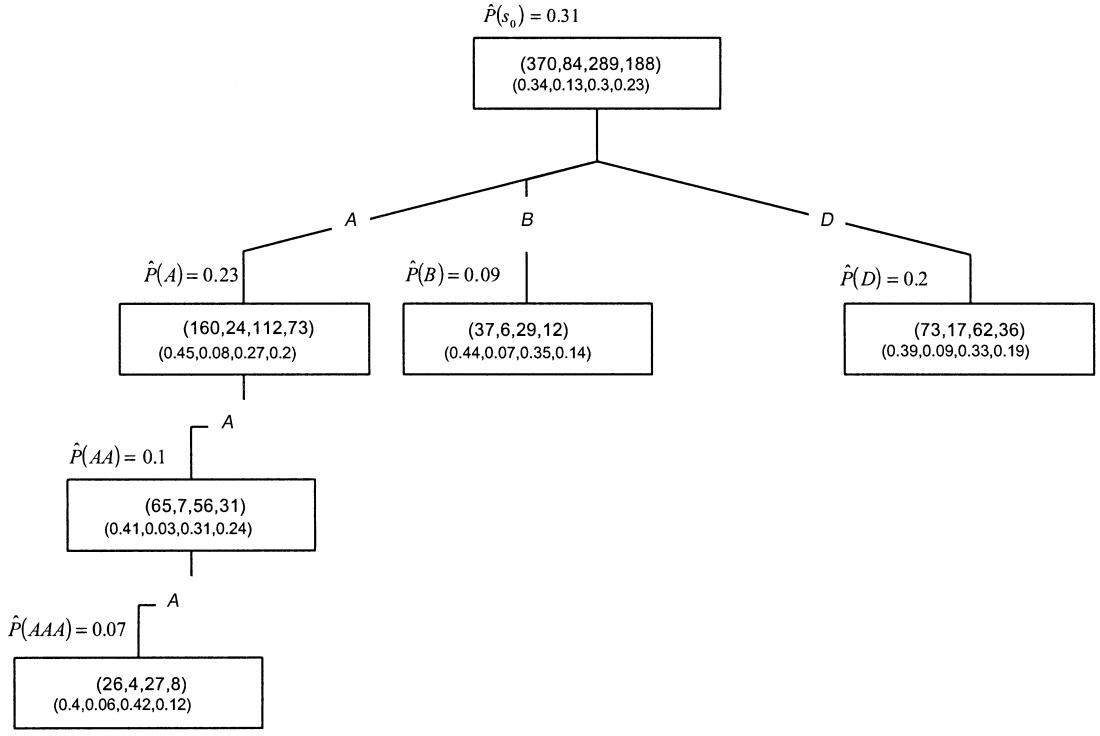


Fig. 10. The updated reference tree $\hat{P}_0^*(x, s)$ with four symbols and six optimal contexts that resulted from the modified input probability ratio.

procedure returned the KL statistics that are plotted in the left-hand side of Fig. 11. As seen, all values are below the UCL (with an ARL $\rightarrow \infty$), confirming that the modified tree has captured the dependencies in the modified system.

4.4.4. *Fourth step: Changing the routings of the different products*

In this step we fixed the part routings, originally presented in Table 1, by selecting route number 2 for all part types. All the other parameters remained unchanged at their values in the third step (i.e., using input probability number 2). Once again, 100 simulation runs were generated based on different seeds and the new routings. We applied the CSPC procedure to construct 100 monitored trees, $\hat{P}_i(x, s)$, $i = 301, \ldots, 400$ and to obtain the KL measures that are plotted in the right-hand side of Fig. 11. As seen, all the values lie above the UCL. At this stage the simulation study was terminated.

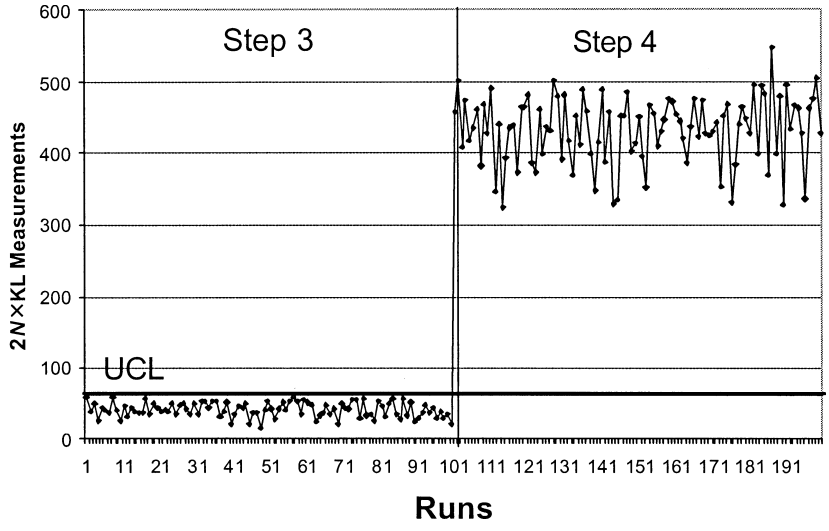


Fig. 11. The CSPC control chart indicating a change in the parts' routings in Step 4 after 100 runs.

It is seen that the proposed CSPC framework performed well throughout the whole simulation study with respect to both statistical errors. 99.5% of the in-control points fell below the upper control limit and 100% of the out-of-control points were clearly identified as being generated from a modified system.

5. Conclusions

Feedback control mechanisms often create autocorrelations (which we also term here as dependencies) among various system characteristics (Box and Luceno, 1997). These dependencies might be state-dependent and nonlinear, thus, unidentifiable by traditional monitoring methods. The CSPC, a new monitoring and statistical control methodology, is proposed in this paper for the analysis of finite-state processes. In particular, the data dependencies are modeled by the context tree that was originally suggested by Rissanen (1983). The KL measures are implemented to calculate the distance between the *monitored context trees* and the *reference context tree*, in order to indicate significant changes in the stochastic process.

The proposed methodology can be applied to monitor discrete processes generated by a finite-state machine source. The varying context lengths in the context-tree model result in fewer parameters that have to be estimated with respect to a fixed-order Markov model and consequently require less data to model the process. The proposed CSPC has some additional appealing characteristics. First, it is *parameter-free* and does not assume, in contrast to conventional SPC methods for autocorrelated processes, a particular time-series model. Second, the proposed approach can "capture" both linear as well as nonlinear trends in the data by using the same procedure. Third, the context tree has a learning ability for the underlying dynamics and distribution during the monitoring stage, as seen in the numerical example presented in Section 4.4. Finally, the CSPC enables the use of a single control chart that can be further decomposed, if necessary, for analysis purposes (Ben-Gal, Morag and Shmilovici, 2003; Ben-Gal, Shmilovici and Morag, 2001).

Despite the above-mentioned advantages of the SCPC, if the true input-output relation can be reasonably approximated by a continuous linear model, then conventional SPC methods for autocorrelated processes should be used. These methods are, in general, more efficient than the CSPC in terms of parameter estimation, complexity of model construction and the required sample sizes. The large sample sizes required by the CSPC might limit its implementation to off-line monitoring applications, or to high-frequency automated monitors that collect large amounts of data, which is often autocorrelated.

Further research in this area has to deal with the statistical properties of the proposed approach. The development of a CSPC version that would require smaller sample sizes could be essential for its utilization as an on-line monitoring method. Another potential direction is the integration between CSPC and different types of popular EPC controllers, such as PID or model-predictive controllers. These endeavors can lead to a better monitoring of production systems and towards a real integration between statistical control and engineering control approaches.

Acknowledgement

This research was partially supported by the MAGNET/CONSIST Grant.

CSPC website: for available CSPC demo software see http://www/eng.tau.ac.il/∼bengal/

References

Alwan, L.C., Ebrahimi, N. and Soofi, E.S. (1998) Theory and methodology—information theoretic framework for process control. *European Journal of Operational Research*, **111**, 526–542.

Apley, D.W. and Shi, J. (1999) The GRLT for statistical process control of autocorrelated processes. *IIE Transactions*, **31**, 1123–1134.

Ben-Gal, I., Morag, G. and Shmilovici, A. (2003) CSPC: a monitoring procedure for state dependent processes. *Technometrics*, **45**(4), 293–311.

Ben-Gal, I., Shmilovici, A. and Morag, G. (2001) Design of control and monitoring rules for state dependent processes. *Journal of Manufacturing Science and Production*, **3**(2/4), 85–93.

Boardman, T.J. and Boardman, E.C. (1990) Don't touch that funnel. *Quality Progress*, **23**, 65–69.

Box, G.E.P., Coleman, D.E. and Baxley, R.V. (1997) A comparison of statistical process control and engineering process control. *Journal of Quality Technology*, **29**, 128–130.

Box, G.E.P. and Jenkins, G.M. (1976) *Times Series Analysis, Forecasting and Control*, Holden Day, Oakland, CA.

Box, G.E.P. and Kramer, T. (1992) Statistical process monitoring and feedback adjustment—a discussion. *Technometrics*, **34**, 251–267.

Box, G.E.P. and Luceno, A. (1997) *Statistical Control by Monitoring and Feedback Adjustment*, Wiley, New York, NY.

Buhlman, P. and Wyner, A.J. (1999) Variable length Markov chains. *The Annals of Statistics*, **27**(2), 480–513.

Carmen, C., Alberto, F. and Rafael, R. (1999) Integration of statistical and engineering process control in a continuous polymerization process. *Technometrics*, **41**, 14–28.

Chan, F.T.S. (1998) Evaluations of operational control rules in scheduling a flexible manufacturing system. *Robotics and Computer-Integrated Manufacturing*, **15**, 121–132.

Cochran, W.G. (1952) The chi-square test of goodness of fit. *The Annals of Mathematical Statistics*, **23**, 315–345.

Deming, W.E. (1986) Out of the crisis. MIT Center for Advanced Engineering Study, Cambridge, MA.

English, J.R., Martin, T., Yaz, E. and Elsayed, E. (2001) Change point detection and control using statistical process control and automatic process control. *Present at the IIE Annual Conference*, 2001, Dallas, TX.

Faltin, F.W., Mastrangelo, C.M., Runger, G.C. and Ryan, T.P. (1997) Considerations in the monitoring of autocorrelated and independent data. *Journal of Quality Technology*, **29**, 131–153.

Gultekin, M., Elsayed, E.A., English, J.R. and Hauksdottir, A.S. (2002) Monitoring automatically controlled processes using statistical control charts. *International Journal of Production Research*, **40**, 2303–2320.

Janakiram, M. and Keats, J.B. (1998) Combining SPC and EPC in a hybrid industry. *Journal of Quality Technology*, **30**, 189–200.

Kullback, S. (1959) *Information Theory and Statistics*, Wiley, New York, NY.

Montgomery, D. (2000) *Introduction to Statistical Quality Control*, 4th edn., Wiley, New York, NY.

Montgomery, D., Keats, J.B., Runger, G.C. and Messina, W.S. (1994) Integrating statistical process control and engineering process control. *Journal of Quality Technology*, **26**, 79–87.

Nembhard, H.B. and Mastrangelo, C.M. (1998) Integrated process control for startup operations. *Journal of Quality Technology*, **30**, 201–210.

Page, E.S. (1962) Cumulative sum schemes using gauging. *Technometrics*, **4**, 97–109.

Rissanen, J.J. (1983) A universal data compression system. *IEEE Transactions on Information Theory*, **29**, 656–664.

Rissanen, J. (1999) Fast universal coding with context models. *IEEE Transactions on Information Theory*, **45**, 1065–1071.

Thomas, W.N. and Lloyd, P.P. (1990) Understanding variation. *Quality Progress*, **23**, 70–78.

Tsung, F., Shi, J. and Wu, C.F.J. (1999) Joint monitoring of PID-controlled processes. *Journal of Quality Technology*, **31**, 275–285.

Vander Weil, S.A., Tucker, W.T., Faltin, F.W. and Doganaksoy, N. (1992) Algorithmic statistical process control: concepts and an application. *Technometrics*, **34**, 286–297.

Weinberger, M., Rissanen, J.J. and Feder, M. (1995) A universal finite memory source. *IEEE Transactions on Information Theory*, **41**, 643–652.

Biographies

Irad Ben-Gal is an Assistant Professor in the Department of Industrial Engineering at Tel Aviv University. He holds a B.Sc. (1992) degree from Tel Aviv University, M.Sc. (1996) and Ph.D. (1998) degrees from Boston University. He is a member of the Institute for Operations Research and Management Sciences (INFORMS) and the Institute of Industrial Engineers (IIE). He is the head of the Computer Integrated Manufacturing (CIM) lab at Tel Aviv University. For several years he has worked for various industrial organizations as a consultant and a project manager. His research interests include quality control, design-of-experiments, testing procedures, and application of information theory to industrial and bioinformatics problems.

Gonen Singer is a Ph.D. student at the Department of Industrial Engineering in Tel Aviv University. He holds B.Sc. (2001) and M.Sc. (2002) degrees from Tel Aviv University. He teaches in the Open University at Tel Aviv. His research interests include quality control, stochastic processes and graphical user interfaces.

Contributed by the On-Line Quality Engineering Department.