Least-Cost Planning Sequence Estimation in Labeled Petri Nets

Lingxi Li and Christoforos N. Hadjicostis

Abstract: This paper develops a recursive algorithm for estimating the least-cost planning sequence in a manufacturing system that is modeled by a labeled Petri net. We consider a setting where we are given a sequence of labels that represents a sequence of tasks that need to be executed during a manufacturing process, and we assume that each label (task) can potentially be accomplished by a number of different transitions which represent alternative ways of accomplishing a specific task. The processes via which individual tasks can be accomplished and the interactions among these processes in the given manufacturing system are captured by the structure of the labeled Petri net. Moreover, each transition in this net is associated with a nonnegative cost that captures its execution cost (e.g., in terms of the amount of workload or power required to execute the transition). Given the sequence of labels (i.e., the sequence of tasks that has to be accomplished), we need to identify the transition firing sequence(s) (i.e., the sequence(s) of activities) that has (have) the least total cost and accomplishes (accomplish) the desired sequence of tasks while, of course, obeying the constraints imposed by the manufacturing system (i.e., the dynamics and structure of the Petri net). We develop a recursive algorithm that finds the least-cost transition firing sequence(s) with complexity that is polynomial in the length of the given sequence of labels (tasks). An example of two parallel working machines is also provided to illustrate how the algorithm can be used to estimate least-cost planning sequences.

Key words: labeled Petri nets; planning sequences estimation; least-cost firing sequences.

The authors are with the Coordinated Science Laboratory and the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, USA. Address for correspondence: C. N. Hadjicostis, 357 Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, USA. Email: chadjc@uiuc.edu.
1. Introduction

Petri nets (PNs) are widely used to model and analyze dynamical systems. Petri net models can compactly represent system behavior, and the graphical representation of a plant as a Petri net model can have advantages when trying to design a monitor, supervise a system or plan sequences of operations in a given plant. As the size and complexity of practical systems increase, significant attention is paid to problems of planning, scheduling, operation, and control of manufacturing systems. In particular, planning has emerged as one of the most important aspects in manufacturing systems and has led to the study of several different types of planning problems in the literature, including assembly and task planning (Rosell, 2004), disassembly planning (Tang et al., 2001), and process planning (Kiritsis et al., 1999). More generally, assembly and process planning can be treated as sequence planning problems where different sequences of activities can accomplish identical tasks (e.g., the assembly of a product); the goal in such settings is to determine a (feasible and optimal) sequence of activities based on particular criteria of interest (Kiritsis et al., 1999; Rosell, 2004).

In this paper we consider the sequence planning problem in manufacturing systems in the context of labeled Petri nets. In our setup, a given sequence of labels represents a sequence of (possibly different) tasks, each of which may be accomplished via a set of different transitions (different alternatives for accomplishing a specific task). The structure of a given labeled Petri net represents the ways in which different tasks can be accomplished and the interactions among them as imposed by the underlying manufacturing system; we assume that each transition in the given net is associated with a nonnegative cost which could represent its viability or process cost (e.g., in terms of the amount of workload or power required to start a machine or assemble a part). Given a sequence of labels (i.e., a sequence of tasks) that have to be accomplished, we aim at finding the transition firing sequence(s) (i.e., the sequence(s) of activities) that accomplishes (accomplish) the specified sequence of tasks and has (have) the least total cost, while adhering with the constraints imposed by the given Petri net. We develop a recursive algorithm that is able to find the least-cost transition firing sequence(s) with complexity that is polynomial in the length of the given sequence of labels (tasks). This implies that we are able to efficiently plan a sequence of activities that agrees with the structure (and dynamics) of the underlying manufacturing system and accomplishes the desirable sequence of tasks.

Note that in our setting, transitions in the net are associated with costs that represent their viability or likelihood. The problem of estimating the least-cost planning sequence(s) has applications in the calculation and prediction of the cost of activities before they are actually performed; this can be particularly useful during early design phases when deciding how to make a product (Kiritsis et al., 1999). In the literature, researchers have also associated transitions with costs for a variety of other applications that could also benefit from the planning algorithm we develop here. In (Sampath et al., 2008), the
authors consider the control reconfiguration problem and associate the firing of each transition with some cost that captures the dependency level for reconfiguration. In (Zussman and Zhou, 1999), the authors consider the problem of adaptive planning of a disassembly process and propose a planning algorithm to find the remanufacturing value; they show that when transitions are associated with nonnegative real numbers the solution they propose is optimal. The authors of (Moore et al., 1998) study optimal disassembly process planning using the reachability tree of a disassembly Petri net, while allowing markings to be associated with costs that vary as a function of time. The authors of (Tang et al., 2006) consider labor cost as a fuzzy variable and study disassembly planning in fuzzy Petri nets.

Related to our work is the work on event estimation in Petri nets which is a problem that has been studied rather extensively. For instance, a survey on the problem of finding legal firing sequences (LFS) for a given Petri net is provided in (Watanabe, 2000). The LFS problem can be described as follows: given a Petri net $N$ with an initial marking $M_0$ and a firing vector $\bar{\sigma} = [\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_m)]^T$ where $\sigma(t_i)$ captures how many times transition $t_i$ in a net with $m$ transitions has occurred, find a sequence of transitions that can be fired one by one, starting from the initial marking $M_0$, such that each transition $t_i$ appears exactly $\sigma(t_i)$ times in the sequence. It was shown in (Watanabe, 2000) that the LFS problem is $NP$-hard for general Petri nets but can be solved for some sub-structures (e.g., unweighted state machines (Morita and Watanabe, 1996) and a class of edge weighted cactuses (Taoka and Watanabe, 1999)). Note that in the planning problem we consider here, the given sequence of labels defines the sequence of tasks, thus providing partial information about the sets of transitions that can fire at each step; an additional challenge, however, arises from the fact that each label (task) can correspond to a group of transitions. Also note that in the LFS problem, when the structure of the Petri net and its initial marking are known, the final marking given a firing vector is unique and can be obtained easily; however, in our setup, many final markings are possible.

One approach to obtain the least-cost planning sequence(s) under the constraints imposed by the sequence of labels (tasks) is to exhaustively evaluate the total costs of all sequences of transitions that are consistent with both the net structure and the sequence of labels (tasks). An alternative approach that we propose and analyze in this paper is to start by first obtaining all consistent markings (i.e., operational states of the underlying manufacturing system) that correspond to each label (task). We then calculate the least-cost firing transition(s) that could lead to each of these markings. From the observation that finding the least-cost planning sequence(s) at time epoch $k$ (i.e., the completion of $k$ tasks) only depends on the $k^{th}$ task and the least-cost planning sequences that lead to each of the consistent markings at time epoch $k-1$, we are able to develop an efficient recursive algorithm that returns the transition firing sequence(s) with the least-cost. Crucial to our algorithmic complexity analysis is the fact that the number of markings that are consistent with a given sequence of labels is upper bounded by a function that is polynomial in the length of the given label sequence (Ru and Hadjicostis, 2006). Note
that our algorithm resembles the Viterbi algorithm (Viterbi, 1967; Forney, 1973) which is able to find the sequence of states that best matches a sequence of events; the main difference is that the set (and number) of consistent markings changes at each time epoch (each time a new label is considered).

2. Petri net notation

In this section, we provide basic definitions and terminology that will be used throughout the paper. More details about Petri nets and their modeling capability can be found in (Murata, 1989; Cassandras and Laforune, 1999).

A Petri net structure is a weighted bipartite graph \( N = (P, T, A, W) \) where 
\[ P = \{ p_1, p_2, \ldots, p_n \} \] is a finite set of places (drawn as circles), 
\[ T = \{ t_1, t_2, \ldots, t_m \} \] is a finite set of transitions (drawn as bars), 
\[ A \subseteq (P \times T) \cup (T \times P) \] is a set of arcs (from places to transitions and from transitions to places), and 
\[ W : A \rightarrow \{1,2,3,\ldots\} \] is the weight function on the arcs.

Let \( b_{ij}^- \) denote the integer weight of the arc from place \( p_i \) to transition \( t_j \), and \( b_{ij}^+ \) denote the integer weight of the arc from transition \( t_j \) to place \( p_i \) (\( 1 \leq i \leq n, 1 \leq j \leq m \)). Note that \( b_{ij}^- (b_{ij}^+) \) is taken to be zero if there is no arc from place \( p_i \) to transition \( t_j \) (or vice versa). We define the input incident matrix \( B^- = [b_{ij}^-] \) (respectively, the output incident matrix \( B^+ = [b_{ij}^+] \)) to be the \( n \times m \) matrix with \( b_{ij}^- \) (respectively, \( b_{ij}^+ \)) at its \( i^{th} \) row, \( j^{th} \) column position. The incident matrix of the Petri net is defined to be \( B = B^+ - B^- \).

A marking is a vector \( M : P \rightarrow \mathbb{Z}_0^+ \) that assigns to each place in the Petri net a nonnegative integer number of tokens (drawn as black dots). We use \( M_0 \) to denote the initial marking of the Petri net. A transition \( t \) is said to be enabled at a marking \( M \) if each of its input places has at least \( B^- (p,t) \) tokens, where \( B^- (p,t) \) is the weight of the arc from place \( p \) to transition \( t \); this is denoted by \( M[t] \). An enabled transition \( t \) may fire. When it fires, it removes \( B^- (p,t) \) tokens from each input place of \( t \) and deposits \( B^+ (p,t) \) tokens to each output place of \( t \) to yield a new marking \( M' = M + B(\cdot,t) \), where \( B(\cdot,t) \) denotes the column of \( B \) that corresponds to \( t \). This is also denoted by \( M[t]M' \).
Let \( \sigma = t_{i_1}t_{i_2}\ldots t_{i_k} (t_{i_j} \in T, j \in \{1,2,\ldots,k\}) \) be a transition firing sequence. We say \( \sigma \) is enabled with respect to \( M \) if \( M(t_{i_1})M(t_{i_2})\ldots M(t_{i_k}) \); this is denoted by \( M(\sigma) \). Let \( M(\sigma)M' \) denote that the firing of \( \sigma \) from \( M \) yields \( M' \) and let \( \overline{\sigma}(t) \) be the total number of occurrences of transition \( t \) in \( \sigma \). More specifically, \( \overline{\sigma} = [\overline{\sigma}(t_1)\ldots\overline{\sigma}(t_m)]^T \) is the firing vector that corresponds to \( \sigma \).

A labeling function \( L: T \rightarrow \Sigma \) assigns to each transition in the net a label (task) from a given alphabet \( \Sigma \). Note that two or more transitions may correspond to a same label (i.e., there may exist different alternatives for accomplishing a specified task). For a label (task) \( l \in \Sigma \), we use \( T_l \) to denote the set of transitions with label \( l \) (i.e., the set of transitions that accomplish task \( l \)). Thus, given a transition firing sequence \( \sigma = t_{i_1}t_{i_2}\ldots t_{i_k} \), the corresponding label (task) sequence is \( \omega = L(\sigma) = L(t_{i_1})L(t_{i_2})\ldots L(t_{i_k}) \), i.e., a string in \( \Sigma^k \).

A cost function \( C: T \rightarrow \mathbb{Z}_0^+ \) assigns to each transition a nonnegative integer cost. Given a transition firing sequence \( \sigma = t_{i_1}t_{i_2}\ldots t_{i_k} \), its total cost is given by \( C(\sigma) = \sum_{j=1}^{k} C(t_{i_j}) \).

**Definition 1** Given an initial marking \( M_0 \) and a label (task) sequence \( \omega \), the set of consistent markings with respect to \( \omega \) is \( Z(\omega) = \{ M \mid \exists \sigma : M_0(\sigma)M \land L(\sigma) = \omega \} \).

**Definition 2** Given a label (task) sequence \( \omega = l_1l_2\ldots l_k (l_j \in \Sigma, j \in \{1,2,\ldots,k\}) \), \( \omega_{k-1} = l_1l_2\ldots l_{k-1} \) is the prefix of \( \omega \) of length \( k-1 \). Similarly, given a transition firing sequence \( \sigma = t_{i_1}t_{i_2}\ldots t_{i_k} \), the prefix of \( \sigma \) of length \( k-1 \) is given by \( \sigma_{k-1} = t_{i_1}t_{i_2}\ldots t_{i_{(k-1)}} \).

**Example 1** Consider the disassembly and assembly operation in the manufacturing system (Desrochers and Al-Jaar, 1995) modeled by the labeled Petri net shown in Figure 1. Transition \( t_1 \) (associated with label \( a \)) represents a disassembly operation where one workpiece is removed from the pallet and placed in \( p_2 \) and the other workpiece along with the pallet are placed in \( p_3 \) concurrently. Then, each workpiece receives subsequent processing (modeled by transitions \( t_2, t_3, t_4 \) and the labels associated with them) before the final assembly operation (modeled by transition \( t_5 \) with label \( d \)) is accomplished. No assembly operation can occur until both workpieces finish their processing.
The Petri net model of the disassembly and assembly operation has places \( P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\} \); transitions \( T = \{t_1, t_2, t_3, t_4, t_5\} \); labels (tasks) \( \sum = \{a, b, c, d\} \); labeling function defined as \( L(t_1) = a, L(t_2) = L(t_3) = b, L(t_4) = c, L(t_5) = d \); and transition costs given by \( C(t_1) = 1, C(t_2) = 2, C(t_3) = 3, C(t_4) = 3 \) and \( C(t_5) = 1 \). Given that our goal is to accomplish the sequence of labels (tasks) \( \omega = abc \), we see that the only possible firing sequence \( \sigma \) is \( t_1t_2t_4 \) and its cost is \( C(t_1) + C(t_2) + C(t_4) = 6 \). The set of consistent markings with respect to the given label sequence \( \omega \) is given by \( Z(\omega) = \{0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \} \).

Figure 1. Disassembly and assembly operation modeled by a labeled Petri net.

3. Problem formulation

The problem we deal with in this paper is the following. We are given a labeled Petri net where different labels in the net represent different tasks, each of which may be accomplished via a set of different transitions that can be considered as different alternatives for accomplishing a specific task (these alternatives share the same label in the Petri net model). Each transition in the net is associated with a nonnegative integer cost that represents its viability or process cost. Given a sequence of labels \( \omega = l_1l_2 \ldots l_k \) (where \( l_j \in \sum, j \in \{1,2,\ldots,k\} \)) that specifies the sequence of tasks to be performed, we aim at finding the underlying (unknown) transition firing sequence(s) that is (are) consistent with both \( \omega \) and the structure of the net, and has (have) the least total cost.

Given the sequence of labels (tasks) \( \omega \), the set of least-cost firing sequence(s) \( \{\sigma_{\min}\} \) is the solution to the following problem:
The problem in (1) could be solved by (i) enumerating each possible length \( k \) sequence \( \sigma \), (ii) evaluating whether it satisfies \( L(\sigma) = \omega \) and \( M_0(\sigma) \), and (iii) obtaining the valid one(s) with the least cost. The problem with this approach is that, in the worst case, the number of transition sequences that satisfy (1) is exponential in the length of the given sequence of labels (tasks). However, by looking at this problem in a different way, i.e., in terms of a trellis diagram (Lin et al., 1998) which describes the evolution of consistent markings over time epochs as shown in Figure 2, we will argue that one can use a dynamic programming approach (Bellman, 1957) to obtain the solution more efficiently in a recursive manner.

\[
\arg \min_{\sigma} C(\sigma) \quad \text{s.t.} \quad L(\sigma) = \omega \quad \& \quad M_0(\sigma)
\]

Figure 2. Trellis diagram in our setup.

In Figure 2, \( \omega = l_1 l_2 \ldots l_k \) denotes the given sequence of labels (tasks) with time epochs (stages) \( 1,2,\ldots,k \) corresponding to the order of labels (tasks) in the sequence. Each node in the trellis diagram (drawn as a big black dot) denotes a marking that is consistent with the net structure and the given sequence of labels (tasks), i.e., \( M_{ji} \in Z(l_1 l_2 \ldots l_j) \) where \( j \in \{1,2,\ldots,k\} \) and \( i \) is the index of a given marking within the set \( Z \). Arcs between nodes represent transitions whose firing will lead from one marking to another. Given the sequence of labels (tasks) \( \omega \), we need to find the set of transition firing sequences that have the least cost from the initial marking \( M_0 \) to any of the consistent markings \( M_{ki} \) that are possible in the last stage.

Note that we do not consider a particular final marking (a target to be reached) which implies that every marking that is consistent with the given sequence of labels (tasks) and
the net structure can appear as a final node in the trellis diagram. Also, in our setup, the number of nodes at each time epoch is not fixed: it can increase or decrease with each time epoch (i.e., the trellis diagram is not regular) but, as we will argue, the number of nodes (consistent markings) at the \( k^{th} \) stage of the trellis diagram is upper bounded by a function that is polynomial in \( k \).

**Definition 3** Let \( \omega_j = l_1 l_2 \ldots l_j \) denote the prefix of the given sequence of labels (tasks) \( \omega \) that has length \( j \) \((j < k)\). The set of markings consistent with \( \omega_j \) is given by \( Z(\omega_j) = \{ M_{j_1}, M_{j_2}, \ldots, M_{j_i}, \ldots \} \) where \( i \) denotes the index of marking \( M_{j_i} \) within the set \( Z(\omega_j) \) in the trellis diagram.

**Definition 4** The set of least-cost firing sequences that lead to the \( i^{th} \) consistent marking \( M_{j_i} \) with respect to \( \omega_j \) is given by \( LC_{i}^{(j)} = \{ \sigma_i \mid \sigma_i = \arg\min_{\sigma} C(\sigma) \text{ for } \sigma \text{ such that } M_{0}[\sigma]M_{j_i} \text{ and } L(\sigma) = \omega_j \} \).

By formulating the problem in terms of a trellis diagram, it is clear that each consistent marking in the set \( Z(\omega_{j+1}) \) has to be reached via at least one consistent marking in the set \( Z(\omega_j) \). Moreover, a dynamic programming approach can be used to compute the least-cost firing sequence(s) recursively. The basic observation is that the transition sequences which have the least cost at time epoch \( j + 1 \) only depend on the least-cost transition sequences up to time epoch \( j \) (specifically, at the sequences \( LC_{i}^{(j)} \) for all indices \( i \) in the set \( Z(\omega_j) \)) and the \((j + 1)^{th}\) label (task). By taking advantage of this observation, one can search for the sequence that has the least cost, one stage in the trellis diagram at a time. In our setup, given each consistent marking \( M_{j_i} \in Z(\omega_j) \) and their associated least-cost firing sequence set \( LC_{i}^{(j)} \), the set of least-cost firing sequences \( LC_{i}^{(j+1)} \) associated with each consistent marking \( M_{(j+1)i} \in Z(\omega_{j+1}) \) after \( j + 1 \) labels (i.e., after \( \omega_{j+1} = l_1 l_2 \ldots l_{j+1} \) has been considered) can be computed as follows:

\[
LC_{i}^{(j+1)} = \{ \sigma_f t_p \mid [\sigma_f, t_p] = \arg\min_{\sigma_f, t_p} (C(\sigma_f) + C(t_p)) \}
\]

where \( \sigma_f \) and \( t_p \) are such that
\[
L(t_p) = l_{j+1} \text{ and } \exists i' \text{ s.t. } \sigma_f \in LC_{i}^{(j)} \text{ and } M_{j_i}[t_p]M_{(j+1)i}.
\]

**Remark 1** In the above recursion, the firing sequence that has least total cost after considering \( j \) labels (tasks) is not necessarily the prefix of the firing sequence that will
give us the least total cost after considering \( j + 1 \) labels (tasks). Therefore, we must keep track of all markings consistent with \( j \) labels (tasks) and their corresponding least-cost firing sequence(s) in order to be able to find those firing sequences that have least total cost after considering \( j + 1 \) labels (tasks). This is due to the fact that we consider the label (task) at each time epoch (which may correspond to a set of consistent markings) instead of only considering a particular consistent marking (the principle of optimality (Bellman, 1957) still holds in our approach if we consider a particular consistent marking instead of a label (task) at each stage).

By calculating (2) recursively in the number of labels (tasks), we can efficiently find the firing sequence(s) that has (have) the least cost. Here we want to point out that the total number of consistent markings (number of nodes in Figure 2) is upper bounded at each stage by a polynomial function in the length of the given sequence of labels (tasks). As we will show, this means that the computational complexity for finding the least-cost firing sequence is polynomial in the length of the given sequence of labels (tasks). In contrast, if we start with the set of transitions that can fire at each step, the number of sequences that will be investigated is exponential in the length of given sequence of labels (tasks).

4. Obtaining the least-cost firing sequence(s)

A. Algorithm

In this section, we propose a recursive algorithm to find the least-cost firing sequence(s) given a label (task) sequence \( w \) of length \( k \). We use a data structure \( \zeta = (M_{current}, LC, (t_in, M_{previous})) \) to capture the information we need to store for each node in the trellis diagram. More specifically, at time epoch \( k \), \( M_{current} \) denotes the marking that is associated with the node (and is, of course, consistent with the given label sequence so far); \( LC \) is the least cost among all valid firing sequences from \( M_1 \) to \( M_{current} \); \( (t_in, M_{previous}) \) denotes that the least cost firing sequence goes through \( M_{previous} \) at time epoch \( k - 1 \) and leads to \( M_{current} \) via the firing of transition \( t_in \) (which is, of course, enabled at \( M_{previous} \)).

We describe the algorithm in detail below.

Algorithm 1
**Input:** A labeled Petri net $N$ with transition costs $C(t_i) \geq 0$ for $t_i \in T$, and a sequence of labels (tasks) $\omega = l_1l_2\cdots l_k$ of length $k$.

1. $\omega_0 = \lambda, \zeta(\omega_0) = \{(M_0, 0, (\emptyset, \emptyset))\}$.
2. Let $j = 1$.
3. Consider the event $\omega_j$.
4. Set $\zeta(\omega_j) = \emptyset$.
5. For all $R \in \zeta(\omega_{j-1})$ do
   
   For all $t$ such that $L(t) = l_j$ and $R.M_{current}[t]$
   
   compute $M' = R.M_{current} + B(\cdot, t)$

   If $M'$ is a new marking that has not appeared in $\zeta(\omega_j)$
   
   $\zeta(\omega_j) = \zeta(\omega_j) \cup \{(M', R.LC + C(t), (t, R.M_{current}))\}$

   Else
   
   $M'$ has appeared in $R' \in \zeta(\omega_j)$

   If $R.LC + C(t) < R'.LC$
   
   $R' = (M', R.LC + C(t), (t, R.M_{current}))$

   Else If $R.LC + C(t) = R'.LC$
   
   $R' = (M', R'.LC, R'.(t_{in}, M_{previous}) \cup (t, R.M_{current}))$

   End IF

   End If

   End For

   End For

6. $j = j + 1$.

7. If $j = k + 1$, Goto 8; else Goto 3.

8. Recover all least-cost firing sequences using consistent markings with the information stored.

Given a labeled Petri net structure $N$ (with $n$ places and $m$ transitions) and a sequence of labels (tasks) $\omega = l_1l_2\cdots l_k$ of length $k$, Algorithm 1 recursively computes markings that are consistent with the sequence of labels (tasks) by looking, at each time epoch $j$, at the set of transitions $T_{l_j}$ corresponding to label $l_j$ and by associating with each marking the transition whose firing (from consistent markings at stage $j - 1$) could lead to it and has the least cost. The algorithm then stores all consistent markings and their corresponding least-cost transition(s), and goes to the next step (when the next label is considered). At any given time epoch, the algorithm can recover the transition sequences that have the least total cost corresponding to all consistent markings (each consistent marking has one (or more) least-cost firing sequence(s) that leads (lead) to it from $M_0$).
Remark 2 For the special case where the incident matrix \( B \) has full column rank, i.e., \( \text{rank}(B) = m \), we know that there is a unique firing vector for each valid final marking in the given net. Furthermore, all transition sequences that get us to this final marking will have the same cost. Depending on the underlying objective, the recursive algorithm can be modified in this case to simply keep track of the number of times each transition appears in a least-cost firing sequence. In addition, given an observed sequence of labels \( \omega \) that is infeasible (i.e., the set of consistent markings \( Z(\omega) \) is an empty set), Algorithm 1 will not return any outputs. Note that deadlock is avoided at intermediate stages \( j \) \((j \in \{1,2,\cdots,k-1\})\) because Algorithm 1 by default will eliminate such markings from further consideration. Deadlocks at the last stage (stage \( k \)) can be checked and prevented using existing deadlock avoidance techniques (Sreenivas, 1997; Park and Reveliotis, 2001; Wu and Zhou, 2005) in literature.

Example 2 Recall the disassembly and assembly operation modeled by the labeled Petri net with transition costs in Figure 1. If the given sequence of labels (tasks) is \( \omega = abbcd \), the corresponding trellis diagram is shown in Figure 3. We treat each consistent marking as a node in the diagram and each time epoch corresponds to the time a label is considered. The set of least-cost firing sequences is given by \( \sigma_{\min} = \{t_1\}, \{t_2\}, \{t_3\}, \{t_4\} \), both with least total cost 10. In terms of our data structure, node \( M_{22} \) would be \( \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \).
B. Upper bound on the number of consistent markings

Before we discuss the complexity of our algorithm, we would like to explain why the total number of consistent markings at each time epoch is upper bounded by a polynomial function in the length of the given sequence of labels (Ru and Hadjicostis, 2006). Note that each consistent marking is associated with at least one firing vector regardless of the ordering of transitions in the corresponding firing sequences. Therefore, if we find the maximum number of possible firing vectors for the observed label sequence, we will get an upper bound on the number of consistent markings.

Consider a label sequence \( \omega \) with \( d \) labels, i.e., \( l_1, l_2, \ldots, l_d \) where \( 1 \leq d \leq m \). The number of transitions for label \( l_i \) is \( h_i = |T_{l_i}| \) (where \( T_{l_i} \) denotes the set of transitions with label \( l_i \) and \( |T_{l_i}| \) is the cardinality of this set) and the labels are renamed (for convenience) such that \( h_i = |T_{l_i}| \geq 2 \) for \( i = 1, 2, \ldots, j \) while \( h_i = |T_{l_i}| = 1 \) for \( i = j + 1, j + 2, \ldots, d \). Then, (Ru and Hadjicostis, 2006) proves the following.

**Proposition 1** (Ru and Hadjicostis, 2006) If the sequence \( \omega \) has length \( k \) with \( l_i \) appearing \( k_i \) times and \( k_1 + \cdots + k_d = k \) (where \( k_i \geq 0 \)), then an upper bound on the number of consistent markings is

\[
\prod_{i=1}^{j} C_{k_i + h_i - 1}^{h_i - 1}, \quad \text{where} \quad C_{k_i + h_i - 1}^{h_i - 1} = \frac{(k_i + h_i - 1)!}{(h_i - 1)!(k_i)!}.
\]

It was also shown in (Ru and Hadjicostis, 2006) that the above function can be bounded by a polynomial function of \( k \) as \( k^b \) where \( b \) is a constant associated with structural parameters of the labeled Petri net. We use this fact in our complexity analysis in the next section.

C. Complexity analysis

The complexity of Algorithm 1 can be obtained as follows. First, regarding space complexity, the storage needed is proportional to the number of consistent markings (nodes). For each consistent marking (apart from the marking information itself and the least cost to get to it), we need to store the valid pairs of transitions and consistent markings in the previous time epoch that could lead to this marking and have the least
cost. If $\overline{L} = \max |T_{ij}|$ denotes the largest possible number of transitions corresponding to a label in the Petri net, then the number of transitions that could lead from distinct (consistent) markings in the previous stage to the current (consistent) marking is bounded by $\overline{L}$. Thus, for each consistent marking, the information to be stored is a constant (proportional to $\overline{L}$). Clearly, since the number of consistent markings associated with a given label sequence of length $k$ is upper bounded by $k^b$ (as argued earlier), the total space needed to store all consistent markings in each stage in the trellis diagram is

$$\sum_{j=1}^{k} O(j^b)$$

which can be simplified as $O(k \cdot k^b) = O(k^{b+1})$, i.e., the storage required is polynomial in the length $k$ of the given sequence of labels (tasks).

We now proceed to analyze computational complexity. We use $n_{k-1}$ to denote the number of consistent markings at the $(k-1)^{st}$ stage in the trellis ($n_{k-1} = O((k-1)^b)$) and $n_k$ to denote the number of consistent markings at the $k^{th}$ stage in the trellis ($n_k = O(k^b)$).

Given a label $l_k$, the number of possible transitions enabled from the $(k-1)^{st}$ stage is upper bounded by $n_{k-1} \cdot \overline{L}$ where $\overline{L}$ is the largest number of transitions corresponding to a certain label. For each marking (in the $k^{th}$ stage) yielding from these transitions, we need at most $n_k$ comparisons to decide if it has appeared or not (by searching through the set of existing markings). Thus, the computational complexity for finding the sequence that has least-cost for all markings at the $k^{th}$ stage is bounded by $n_{k-1} \cdot \overline{L} \cdot n_k$, which has complexity $O(\overline{L} \cdot k^{2b})$ where $\overline{L}$ is a constant. Thus, for consistent markings over all stages, the total computational complexity is given by $\sum_{j=1}^{k} O(j^{2b})$, which can be simplified as $O(k^{2b+1})$. Therefore, the algorithm has complexity that is polynomial in the length $k$ of the given sequence of labels (tasks).

5. An illustrative example

In this section, we illustrate the algorithm via a more complicated example. In Figure 4, consider two parallel part producing machines (Proth and Xie, 1996) that are modeled by a labeled Petri net. The Petri net has 10 places $P = \{p_1, p_2, \ldots, p_{10}\}$, 12 transitions $T = \{t_1, t_2, \ldots, t_{12}\}$, and initial marking $M_0 = \begin{bmatrix} 1 & 0 & 2 & 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$. The labeling function is given by $L(t_1) = L(t_2) = a$, $L(t_3) = L(t_6) = b$, $L(t_7) = L(t_9) = c$, $L(t_4) = L(t_{10}) = d$, $L(t_5) = e$, $L(t_8) = f$, $L(t_{11}) = g$ and $L(t_{12}) = h$. The cost of each transition is given by the cost vector
Our goal is to estimate least-cost planning sequence(s) based on different sequences of labels (tasks).

Figure 4. Petri net model for two parallel machines.

We consider a sequence of labels \(\omega = eeffaabcdeeffabcdabcdgghhabcd\) of length 30 to illustrate our algorithm.

Due to space limitations, we do not provide the trellis diagram associated with this example. Instead, we provide the following table where Label denotes the label (task) in the given sequence, Num of Markings gives number of consistent markings with the given label (task), LC captures the least total cost of sequence(s) that is (are) consistent with the labels given up to the current time epoch and \(\{\sigma_{\text{min}}\}\) gives the set of planning sequence(s) that has (have) least total cost up to the current time epoch.

**Table 1** Complete results of our example

<table>
<thead>
<tr>
<th>Label</th>
<th>Num of Markings</th>
<th>LC</th>
<th>({\sigma_{\text{min}}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>1</td>
<td>5</td>
<td>(t_1)</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>10</td>
<td>(t_1t_1)</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>15</td>
<td>(t_1t_1t_2)</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>20</td>
<td>(t_1t_1t_2t_2)</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>30</td>
<td>(t_1t_1t_2t_3t_3)</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>40</td>
<td>(t_1t_1t_2t_3t_3)</td>
</tr>
</tbody>
</table>
From Table 1 we see that, if the given sequence of labels (tasks) is $\omega^{(1)} = eeff aa$, Algorithm 1 finds the least-cost planning sequence to be $\sigma_{\min} = t_1t_2t_3t_4t_6$ with total cost 50. In addition, if the given sequence of labels (tasks) is $\omega^{(2)} = eeff aabcc$, Algorithm 1 finds the least-cost planning sequence to be $\sigma_{\min} = t_1t_2t_3t_4t_5t_6$ with total cost 110. Note that the two sequences of labels (tasks) share the first seven labels (tasks) and differ only in the last three labels (tasks); however, the least-cost planning sequence is different in the first seven transitions, i.e., the planning sequence with least-cost may vary as we are given more labels (tasks). More generally, the least-cost planning sequence after considering $k-1$ labels (tasks) is not necessarily a prefix of the sequence that will give us the least-cost after considering $k$ labels (tasks). The one(s) with least-cost can only be found by capturing *all* consistent markings with least-cost planning sequence information of the $(k-1)^{th}$ stage, as we have done in our algorithm.

### 6. Conclusions and future work

In this paper we considered the sequence planning problem in manufacturing systems that can be modeled as labeled Petri nets. In particular, a given sequence of labels represents a sequence of tasks that have to be accomplished during the entire manufacturing process, with each label (task) potentially corresponding to a number of different transitions (which represent different alternatives for accomplishing a specific task). We assume that each transition in the given net is associated with a nonnegative cost which could represent its viability and execution cost. Given the structure of a labeled Petri net (i.e., the ways in which different tasks can be accomplished and interactions among them) and given a sequence of labels (i.e., a sequence of tasks that have to be accomplished), we

<table>
<thead>
<tr>
<th>$b$</th>
<th>4</th>
<th>50</th>
<th>$t_1t_2t_3t_4t_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c$</td>
<td>4</td>
<td>70</td>
<td>$t_1t_2t_3t_5t_6t_9$</td>
</tr>
<tr>
<td>$c$</td>
<td>2</td>
<td>100</td>
<td>$t_1t_2t_3t_5t_6t_7t_9$</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>110</td>
<td>$t_1t_2t_3t_4t_5t_9t_8$</td>
</tr>
<tr>
<td>$b$</td>
<td>4</td>
<td>290</td>
<td>$t_1t_2t_3t_4t_5t_6t_9t_8t_9$</td>
</tr>
<tr>
<td>$c$</td>
<td>2</td>
<td>310</td>
<td>$t_1t_2t_3t_4t_5t_6t_9t_8t_9t_10$</td>
</tr>
<tr>
<td>$d$</td>
<td>1</td>
<td>330</td>
<td>$t_1t_2t_3t_4t_5t_6t_9t_8t_9t_10t_11t_12t_13t_14$</td>
</tr>
</tbody>
</table>
developed a recursive algorithm that finds the planning sequence(s) that has (have) the least total cost (i.e., the sequence(s) of activities that accomplishes (accomplish) the specified sequence of tasks and has (have) the least total cost). We also showed that the algorithm has complexity that is polynomial in the length of the given sequence of labels (tasks).

One possible direction for future work is to find classes of nets for which the complexity of the algorithm can be further reduced. Another interesting question is to study the problem in situations where the initial marking of the net is only partially known.

References


Figure 1. Disassembly and assembly operation modeled by a labeled Petri net.
Figure 2. Trellis diagram in our setup.
Figure 3. Trellis diagram for the net in Figure 1 with sequence of labels (tasks) $\omega = abbcdd$
Figure 4. Petri net model for two parallel machines.
Table 1: Complete results of our example

<table>
<thead>
<tr>
<th>Label</th>
<th>Num of Markings</th>
<th>LC</th>
<th>( { \sigma_{\text{min}} } )</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>1</td>
<td>5</td>
<td>( t_1 )</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>10</td>
<td>( t_1 t_1 )</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>15</td>
<td>( t_1 t_1 t_2 )</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>20</td>
<td>( t_1 t_1 t_2 t_2 )</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>30</td>
<td>( t_1 t_2 t_3 t_3 )</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
<td>40</td>
<td>( t_1 t_2 t_3 t_3 t_3 )</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>50</td>
<td>( t_1 t_2 t_3 t_6 )</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>70</td>
<td>( t_1 t_2 t_3 t_4 t_6 t_9 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( t_1 t_1 t_2 t_3 t_3 t_3 t_9 )</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>100</td>
<td>( t_1 t_2 t_3 t_5 t_6 t_7 t_9 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( t_1 t_1 t_2 t_3 t_5 t_6 t_7 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>( t_1 t_1 t_2 t_3 t_5 t_6 t_9 )</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>110</td>
<td>( t_1 t_1 t_2 t_3 t_4 t_9 t_8 )</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>290</td>
<td>( t_1 t_1 t_2 t_2 t_3 t_4 t_5 t_6 )</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>310</td>
<td>( t_1 t_1 t_2 t_3 t_4 t_5 t_6 t_9 )</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>330</td>
<td>( t_1 t_1 t_2 t_2 t_5 t_4 t_8 t_9 t_8 t_11 t_12 t_12 t_5 t_4 t_8 )</td>
</tr>
</tbody>
</table>