A New Algorithm for Supervisor Reduction/Localization

Yingying Liu [a], Lihua Wu [b], Renyuan Zhang [c], Zhaojian Cai [d], Kai Cai [d]

[a]*School of Mechanical and Electronic Engineering, Northwest A & F University, Xi'an, China*

[b]*College of Mathematics and Informatics, South China Agricultural University, Guangzhou, China*

[c]*School of Automation, Northwestern Polytechnical University, Xi'an, China*

[d]*Department of Core Informatics, Osaka Metropolitan University, Osaka, Japan*

Abstract

In this paper we propose a fast algorithm for supervisor reduction/localization of discrete-event systems (DES). Supervisor reduction/localization is based on merging pairs of states of the supervisor that are control consistent. Our proposed algorithm employs two new lists – a *mergeable* list and a *non-mergeable* list – which store state pairs that have been confirmed to be control consistent or inconsistent, respectively. With these two lists, our algorithm eliminates any repeated control consistency checks, and guarantees that every state pair will be checked exactly once. We prove that the time complexity of our new algorithm is $O(n^2)$, where n is the state number of the supervisor; this improves all previously known results on supervisor reduction/localization algorithms. Moreover, we provide the space complexity of the new algorithm. We employ numerical examples to empirically compare the computing time and local controller's state number obtained by our proposed algorithm and the original supervisor reduction/localization algorithms.

1 Introduction

The supervisory control theory [11], [19], [18], [6] is a powerful framework for the synthesis of control for discrete-event systems (DES). With a single centralized supervisor, the controlled behavior of the plant, with respect to (w.r.t.) an imposed specification, can be made optimal and nonblocking. The goal of making the supervisor's control logic more comprehensible is achieved by *supervisor reduction*.

The supervisor reduction problem was first studied in [15]. Subsequently, the authors of [12] proposed a polynomial-time supervisor reduction algorithm, and proved that computing a minimal supervisor is NP-hard. This work was further extended in [13]. Supervisor reduction is applied in many scenarios. A attack model reduction problem is reduced to the supervisor reduction problem to provide a simplified attack logic, which discloses the key observation sequences resulting in the damage infliction, and thus to guide system designers to improve security level [14],[8]. The supervisor reduction algorithm in [12] is used to synthesize the supremal networked supervisor with a reduced size in [9].

To synthesize a distributed control architecture, *supervisor localization* explored in [3][5] has a similar procedure to supervisor reduction in [12]. As a generalization of supervisor reduction, supervisor localization has a new feature: it is conducted based solely on local (agent wise) control information. The authors of [2] developed a general framework, called "consistent reduction", for formalizing and solving a large class of minimization/reduction problems in DES. An algorithm is proposed to solve the formulated consistent reduction problem based on splitting covers. In the state tree structure (STS) framework, the authors of [4] developed a counterpart state-based localization theory. Compared to the language-based theory in [3][5], they also presented a more efficient symbolic localization algorithm, by exploiting binary decision diagram computation. Reduction [12] and localization [3][5] procedures applied to a monolithic supervisor that is synchronizing w.r.t. the initial state, respectively, reduced and localized supervisors that are also synchronizing w.r.t. the initial state, if the premises are fulfilled in [1]. Instead of reducing the number of states in [3][5][12], the authors of [10] improved supervisor reduction for DES by reducing the number of events to make supervisor or controller more manageable.

The crucial operation in the localization (or reduction) algorithm is to check control consistency (precise definition given in Section 2) for every state pair of the supervisor, and two states may be merged together if they are control consistent as well as all their future states (reachable by the same strings) are also control consistent. The latter is re-

Email addresses: yingyingliu611@163.com (Yingying Liu), lhwu@outlook.com (Lihua Wu), ryzhang@nwpu.edu.cn (Renyuan Zhang), zhaojian.cai@c.info.eng.osaka-cu.ac.jp (Zhaojian Cai), cai@omu.ac.jp (Kai Cai).

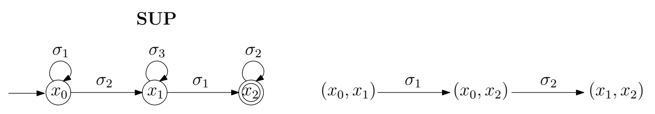

Fig. 1. Supervisor **SUP** and its corresponding state pair transition diagram. Consider the situation that in **SUP**, σ_3 is the only controllable event which is 'don't care' (neither enabled nor disabled) at x_0 and is enabled at state x_1 but is disabled at state x_2. We say that states x and x' are control consistent if there is no controllable event that is enabled at x but is disabled at x', or vice versa. Hence, only x_1 and x_2 are not control consistent. In the state pair transition diagram, $(x_0,x_1) \xrightarrow{\sigma_1} (x_0,x_2)$ means that x_0 and x_2 are the corresponding future states of x_0 and x_1, respectively, by a one-step transition σ_1 defined in **SUP**.

ferred to as the *mergeability* condition, and is verified by a function called CHECK_MERGE. As one of the arguments of the function CHECK_MERGE, a waiting list stores all the state pairs whose future states are waiting to be checked for control consistency. Unfortunately, this waiting list is initialized afresh at each iteration and does not 'memorize' which state pairs have already been checked. As a result, control consistency of some state pairs may be checked repeatedly in [3][5][12]. The reduction and localization algorithms have (worst-case) time complexity $O(n^4)$, where n denotes the state number of the given supervisor.

An illustrative example is given in Fig. 1. Suppose that the original reduction algorithm in [12] (or localization algorithm in [3][5]) is adopted for the example. Since state pair (x_0,x_2) is the corresponding future state pair of (x_0,x_1) via event σ_1, and (x_1,x_2) is the corresponding future state pair of (x_0,x_2) via event σ_2, control consistency of the above three pairs will all be checked in the first iteration of the algorithm to determine the mergeability of x_0 and x_1. In the second iteration for mergeability of x_0 and x_2, control consistency of (x_0,x_2) and (x_1,x_2) will be checked again. Similarly, in the third and last iteration for mergeability of x_1 and x_2, control consistency of (x_1,x_2) will be checked again. In this paper, we propose a generalization of the localization algorithm in [3][5] by inserting new filters to avoid repeated checking for control consistency of each state pair. If we adopt the new algorithm proposed in this paper for the above example, then all the repeated checking operations for (x_0,x_2) and (x_1,x_2) will not take place in the last two iterations.

In this paper, we develop a new supervisor localization algorithm by employing two new lists – a *mergeable* list and a *non-mergeable* list – which store state pairs that have been confirmed to satisfy or dissatisfy the mergeability condition, respectively. With these two lists serving as filters, our proposed algorithm eliminates any repeated control consistency checks, and guarantees that every state pair will be checked exactly once. In comparison with the reduction algorithm in [12] and the localization algorithm in [3][5], the time complexity of the algorithm in this paper is reduced from $O(n^4)$ to $O(n^2)$. Therefore, the algorithm we present is more efficient than those in [3][5][12]. Notice that, the symbolic localization algorithm based on STS in [4] has time complexity $O(n^3)$. In the worst case, $n(n-1)/2$ calls are made to the function CHECK_MERGE, which then calls itself at most $n-2$ times. Since the former $n(n-1)/2$ calls of CHECK_MERGE are unavoidable, even if the idea of using mergeable and non-mergeable lists is adopted, the STS algorithm would have at least $O(n^2)$ time complexity,

With our developed new algorithm, it becomes more efficient to compute a reduced supervisor for improving comprehensibility of control logic, as well as more efficient to synthesize local controllers for component agents to establish distributed control architecture. For the latter, in cases where agent number and/or specifications change such that new local controllers must be computed, our more efficient algorithm will be more useful.

Compared to the conference predecessor [20], this paper provides (i) proof of the time complexity result; (ii) a new result of space complexity and its proof; (iii) new empirical studies for numerical comparisons.

The remainder of the paper is organized as follows. Section 2 presents preliminaries. A new supervisor localization algorithm is presented, and the time complexity and space complexity of the algorithm are analyzed in Section 3. In this section the computational efficiency of the algorithm is demonstrated with an example. We compare the computing times of our proposed algorithm and the original supervisor reduction/localization algorithm by numberical examples in Section 4. We finally draw our conclusions in Section 5.

2 PRELIMINARIES

The DES plant to be controlled is modeled by a generator $\mathbf{G} = (Q,\Sigma,\delta,q_0,Q_m)$, where Q is the finite state set, $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ is the finite event set which is partitioned into two subsets – the controllable event subset Σ_c and the uncontrollable event subset Σ_u, $\delta : Q \times \Sigma \to Q$ is the (partial) state transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marker states. In the usual way, we extend δ such that $\delta : Q \times \Sigma^* \to Q$, and write $\delta(q,s)!$ to mean that $\delta(q,s)$ is defined, where $q \in Q$ and $s \in \Sigma^*$. A string $s_1 \in \Sigma^*$ is a *prefix* of another string $s \in \Sigma^*$, written $s_1 \leq s$, if there exists $s_2 \in \Sigma^*$ such that $s_1 s_2 = s$. The *prefix closure* of L, written $\overline{L}$, is $\overline{L} := \{s_1 \in \Sigma^* \mid (\exists s \in L) s_1 \leq s\}$. A language L is *closed* if $L = \overline{L}$. The *closed behavior* of $\mathbf{G}$ is the language $L(\mathbf{G}) := \{s \in \Sigma^* \mid \delta(q_0,s)!\}$. The *marked behavior* of $\mathbf{G}$ is the sublanguage $L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) \mid \delta(q_0,s) \in Q_m\} \subseteq L(\mathbf{G})$. The generator $\mathbf{G}$ is *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$.

Consider that the plant $\mathbf{G}$ consists of N (> 1) component agents $\mathbf{G}_k = (Q_k,\Sigma_k,\delta_k,q_{0,k},Q_{m,k})$, $k = 1,2,\ldots,N$. Write $\underline{N}$ for the set of integers $\{1,2,\ldots,N\}$. Then the closed and marked behaviors of $\mathbf{G}$ are $L(\mathbf{G}) = \|\{L(\mathbf{G}_k) | k \in \underline{N}\}$ and $L_m(\mathbf{G}) = \|\{L_m(\mathbf{G}_k) | k \in \underline{N}\}$, respectively, where $\|$ denotes *synchronous product* ([18]). In this paper, we assume that for every $k \in \underline{N}$, $\mathbf{G}_k$ is nonblocking. Each agent's event set Σ_k is partitioned into two subsets – a controllable subset $\Sigma_{c,k}$ and an uncontrollable subset $\Sigma_{u,k}$, i.e., $\Sigma_k = \Sigma_{c,k} \dot{\cup} \Sigma_{u,k}$. Hence the plant $\mathbf{G}$ is defined over $\Sigma := \dot{\cup}\{\Sigma_k | k \in \underline{N}\}$, with controllable subset $\Sigma_c := \dot{\cup}\{\Sigma_{c,k} | k \in \underline{N}\}$ and uncontrollable subset $\Sigma_u := \dot\cup\{\Sigma_{u,k}|k \in \underline{N}\}$. The component agents $\mathbf{G}_k$ ($k = 1,2,\dots,N$) are implicitly coupled through a specification language $E \subseteq \Sigma^*$ that imposes a constraint on the global behavior of $\mathbf{G}$, where E may be the synchronous product of multiple component specifications (see [7] and subsequent literature, e.g., [16]).

We say that E is *controllable* w.r.t. $\mathbf{G}$ if $\overline{E}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{E}$ [18]. Whether or not a specification language E is controllable, we denote by $\mathscr{C}(E)$ the family of all sublanguages of E, i.e., $\mathscr{C}(E) := \{K \subseteq E \mid \overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}\}$. The empty language $\emptyset$ belongs to $\mathscr{C}(E)$. Furthermore, $\mathscr{C}(E)$ has a (unique) supremal element $\sup\mathscr{C}(E) := \bigcup\{K|K \in \mathscr{C}(E)\}$ [18]. For the plant $\mathbf{G}$ and the imposed specification E, the optimal and nonblocking monolithic supervisor $\mathbf{SUP} = (X,\Sigma,\xi,x_0,X_m)$ is such that $L_m(\mathbf{SUP}) := \sup\mathscr{C}(E \cap L_m(\mathbf{G}))$ [5]. In the following, we assume that $L_m(\mathbf{SUP}) \neq \emptyset$, and $\mathbf{SUP}$ is feasibly computable.

The key concept of supervisor localization is control cover, adapted from [12] for supervisor reduction. Note that the control information in $\Sigma_{c,k}$ (concerning only the controllable events in $\Sigma_{c,k}$) is characterized by the following four functions. Let $\mathbf{G} = (Q,\Sigma,\delta,q_0,Q_m)$, $\mathbf{SUP} = (X,\Sigma,\xi,x_0,X_m)$ and define $E : X \to Pwr(\Sigma)$ as $E(x) = \{\sigma \in \Sigma \mid \xi(x,\sigma)!\}$, where $Pwr(\cdot)$ denotes powerset. Thus $E(x)$ is the subset of events that are enabled at state x in $\mathbf{SUP}$. Next define $D_k : X \to Pwr(\Sigma_{c,k})$ according to $D_k(x) = \{\sigma \in \Sigma_{c,k} \mid \neg\xi(x,\sigma)!\ \&\ (\exists s \in \Sigma^*)(\xi(x_0,s) = x\ \&\ \delta(q_0,s\sigma)!)\}$. Thus $D_k(x)$ is the subset of controllable events in $\Sigma_{c,k}$ that must be disabled at x in $\mathbf{SUP}$. We say that x is a 'don't care' state for event σ if $\sigma \notin E(x)$ and $\sigma \notin D_k(x)$. Then define $M : X \to \{0,1\}$ according to $M(x) = 1$ if and only if $x \in X_m$. So $M(x) = 1$ means that state x is marked in $\mathbf{SUP}$. Finally define $T : X \to \{0,1\}$ according to $T(x) = 1$ if and only if $(\exists s \in \Sigma^*)\xi(x_0,s) = x\ \&\ \delta(q_0,s) \in Q_m$. Hence $T(x) = 1$ means that there exists a string that reaches x and also reaches some marker state in $\mathbf{G}$. Based on the above four definitions, a binary relation $\mathscr{R}_k$ on X is defined [5].

Definition 1 *Let $\mathscr{R}_k \subseteq X \times X$. We say that $\mathscr{R}_k$ is a* control consistency relation *(w.r.t. $\Sigma_{c,k}$) if for every $x,x' \in X$, $(x,x') \in \mathscr{R}_k$ if and only if*

(1) $E(x) \cap D_k(x') = \emptyset = E(x') \cap D_k(x)$
(2) $T(x) = T(x') \Rightarrow M(x) = M(x')$.

Obviously, $\mathscr{R}_k$ is reflexive and symmetric, but need not be transitive. Thus it is not an equivalence relation on X. This fact leads to the following definition of control cover [5]. Recall that a cover of a set X is a family of nonempty subsets (or cells) of X whose union is X.

Definition 2 *Let $\mathscr{C}_k = \{X_i \subseteq X \mid i \in I_k\}$ be a cover of X, with I_k a suitable index set. We say that $\mathscr{C}_k$ is a* control cover *of X (w.r.t. $\Sigma_{c,k}$) if*

(1) $(\forall i \in I_k, \forall x,x' \in X_i)(x,x') \in \mathscr{R}_k$
(2) $(\forall i \in I_k, \forall\sigma \in \Sigma)\big[\big((\exists x \in X_i)\xi(x,\sigma)!\big) \Rightarrow \big((\exists j \in I_k)(\forall x' \in X_i)\xi(x',\sigma)! \Rightarrow \xi(x',\sigma) \in X_j\big)\big]$.

The subsets X_i ($i \in I_k$) are the cells of $\mathscr{C}_k$. A control cover $\mathscr{C}_k$ is a *control congruence* if $\mathscr{C}_k$ happens to be a partition of X, namely its cells X_i are pairwise disjoint.

With the control cover $\mathscr{C}_k$ of X, a nonblocking local controller $\mathbf{LOC}_k$ for agent $\mathbf{G}_k$ can be constructed by the localization procedure in [3][5]. Since the cells of a control cover may overlap, the constructed local controller may not be unique. One would like to obtain local controllers of minimal number of states; the minimal state problem is, however, NP-hard [5, Appendix B]. As a generalization of supervisor reduction algorithm, a polynomial localization algorithm, which generates control congruences instead of covers, is developed in [3][5]. The localization algorithm is implemented in TCT [17] by computing $\{\mathbf{LOC}_1,\mathbf{LOC}_2,\dots,\mathbf{LOC}_N\} = Localize(\mathbf{G},\{\mathbf{G}_1,\mathbf{G}_2,\dots,\mathbf{G}_N\},\mathbf{SUP})$, where plant $\mathbf{G}$, agents $\mathbf{G}_k$ ($k \in \underline{N}$), and the monolithic supervisor $\mathbf{SUP}$ are input generators, and local controllers $\mathbf{LOC}_k$ ($k \in \underline{N}$) are output generators. Thus a set of local controllers $\{\mathbf{LOC}_k \mid k \in \underline{N}\}$, one for each agent, with $L(\mathbf{LOC}) := \|\{L(\mathbf{LOC}_k)|k \in \underline{N}\}$ and $L_m(\mathbf{LOC}) := \|\{L_m(\mathbf{LOC}_k)|k \in \underline{N}\}$ such that $L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP})$ and $L_m(\mathbf{G}) \cap L(\mathbf{LOC}) = L_m(\mathbf{SUP})$ hold.

The time complexity of the localization algorithm is $O(n^4)$, where n is the state number of $\mathbf{SUP}$. Since control consistency of some state pairs may be checked repeatedly in the algorithm (as illustrated in Fig. 1), we are motivated to improve its complexity by avoiding these repeated checks.

3 A NEW LOCALIZATION ALGORITHM

In this section we first present a new and more efficient polynomial algorithm to compute a control congruence from which a local controller is generated. We call this algorithm Fast Localization Algorithm (FLA); it is a variation of the localization algorithm in [3][5] and the reduction algorithm in [12]. Then we analyze the time complexity and space complexity of FLA.

3.1 Algorithm Description

First, we sketch the idea of FLA. Given a supervisor $\mathbf{SUP} = (X,\Sigma_c\dot\cup\Sigma_u,\xi,x_0,X_m)$, let the states in X be labeled as $X = \{x_0,x_1,\dots, x_{n-1}\}$ and $\Sigma_{c,k} \subseteq \Sigma_c$ be the set of controllable events of agent $\mathbf{G}_k$ (for a fixed $k \in \underline{N}$). The proposed FLA will generate a control congruence $\mathscr{C}_k$ of X w.r.t. $\Sigma_{c,k}$.

Initially, $\mathscr{C}_k$ is set to be the singleton partition of X, i.e.,

$$\mathscr{C}_k := \{[x_i] = \{x_i\}|i = 0,1,\dots,n-1\} = \{[x_0],[x_1],\dots,[x_{n-1}]\}$$

where $[x]$ denotes the cell in $\mathscr{C}_k$ to which x belongs. For a state x, we call other states $x' \in [x]$ the *equivalent states* of x; for a state pair (x'_i,x'_j), we call other state pairs $(x_i,x_j) \in [x'_i] \times [x'_j]$ the *equivalent state pairs* of (x'_i,x'_j). Then FLA 'merges' two cells $[x'_i]$ and $[x'_j]$ into one if for every $x_i \in [x'_i]$ and $x_j \in [x'_j]$, x_i and x_j, as well as all their corresponding future states reachable by identical strings, are control consistent in terms of $\mathscr{R}_k$ (Definition 1). This 'mergeability' condition is checked by a function called CHECK_MERGE in the pseudocode below. This CHECK_MERGE function checks control consistency of the current state pair (x_i,x_j) at line 21 and recursively checks for all their corresponding future states at lines 25–33. The arguments of the CHECK_MERGE function are a state pair, a 'waiting list' W, and the index i (with current value from line 4). Here the waiting list $W \subseteq X \times X$ stores all the state pairs whose future states are waiting to be checked for control consistency. So far the elements introduced above are inherited from [3][5][12]. The following are five new elements in our proposed FLA.

(1) A 'mergeable list' $\surd\_list \subseteq X \times X$ stores all the state pairs that can be merged, that is, they satisfy the mergeability condition.
(2) A 'non-mergeable list' $\times\_list \subseteq X \times X$ stores all the state pairs that cannot be merged, that is, they dissatisfy the mergeability condition.
(3) An 'index' $pos(x_i,x_j) \in \{1,2,\dots,n(n-1)/2\}$ records the position number of (x_i,x_j) stored in the waiting list W. For instance, if $pos(x_i,x_j) = 3$, this means that (x_i,x_j) is the third state pair in W.
(4) A 'dynamic list' $dynamic(x_i,x_j)$, initialized at line 24, stores all the state pairs in the waiting list W that determine the mergeability of x_i and x_j. Note that, if there exist two state pairs $(x'_i,x'_j),(x''_i,x''_j)$ such that $(x'_i,x'_j) \in dynamic(x_i,x_j)$ and $(x''_i,x''_j) \in dynamic(x'_i,x'_j)$, then by transitivity we do not need to have $(x''_i,x''_j) \in dynamic(x_i,x_j)$.
(5) $root(x_i,x_j)$ records the minimal position number of the state pairs in the waiting list W that determine the mergeability of x_i and x_j.

Now we present the pseudocode of FLA. The MAIN procedure consists of lines 1–13, and it calls the function CHECK_MERGE at line 10. CHECK_MERGE consists of lines 14–38, and it calls at line 37 another function DRAWN_CHECK_MARK (lines 39–43).

```
1: procedure MAIN()
2: $\surd\_list := \emptyset$;
3: $\times\_list := \emptyset$;
4: for $i := 0$ to $n-2$ do
5:     if $i > \min\{m | x_m \in [x_i]\}$ then continue;
6:     for $j := i+1$ to $n-1$ do
7:         if $j > \min\{m | x_m \in [x_j]\}$ then continue;
8:         if $\{(x_i,x_j),(x_j,x_i)\} \cap \times\_list \neq \emptyset$ then continue;
9:         $W := \emptyset$;
10:        CHECK_MERGE($x_i,x_j,W,i$);
11:        for each $(x_p,x_q) \in W$ do
12:            if $\{(x_p,x_q),(x_q,x_p)\} \cap \surd\_list = \emptyset$ then
                   $\times\_list := \times\_list \cup \{(x_p,x_q)\}$;
13:            else $[x_p] = [x_q] := [x_p] \cup [x_q]$;
14: function CHECK_MERGE($x_i,x_j,W,i$)
15: for each $x_p \in [x_i] \cup \bigcup\{[x] | \{(x,x_i),(x_i,x)\} \cap W \neq \emptyset\}$ do
16:     for each $x_q \in [x_j] \cup \bigcup\{[x] | \{(x,x_j),(x_j,x)\} \cap W \neq \emptyset\}$
            do
17:         if $\{(x_p,x_q),(x_q,x_p)\} \cap W \neq \emptyset$ then
18:             $dynamic(x_i,x_j) := dynamic(x_i,x_j) \cup \{(x_p,x_q)\}$;
19:             continue;
20:         if $\{(x_p,x_q),(x_q,x_p)\} \cap \times\_list \neq \emptyset$ then return false;
21:         if $(x_p,x_q) \notin \mathscr{R}_k$ then $\times\_list := \times\_list \cup \{(x_p,x_q)\}$;
                return false;
22:         $W := W \cup \{(x_p,x_q)\}$;
23:         $root(x_p,x_q) = root(x_q,x_p) := pos(x_p,x_q)$;
24:         $dynamic(x_p,x_q) := \{(x_p,x_q)\}$;
25:         for each $\sigma \in \Sigma$ with $\xi(x_p,\sigma)!$ & $\xi(x_q,\sigma)!$ do
26:             if $[\xi(x_p,\sigma)] = [\xi(x_q,\sigma)]$ then continue;
27:             if $\{(\xi(x_p,\sigma),\xi(x_q,\sigma)),(\xi(x_q,\sigma),\xi(x_p,\sigma))\} \cap$
                    $W \neq \emptyset$ then
28:                 $dynamic(x_p,x_q) := dynamic(x_p,x_q) \cup$
                        $\{(\xi(x_p,\sigma),\xi(x_q,\sigma))\}$;
29:                 continue;
30:             if $\{(\xi(x_p,\sigma),\xi(x_q,\sigma)),(\xi(x_q,\sigma),\xi(x_p,\sigma))\} \cap$
                    $\times\_list \neq \emptyset$ then return false;
31:             if $\min\{m | x_m \in [\xi(x_p,\sigma)]\} < i$ or $\min\{m | x_m \in$
                    $[\xi(x_q,\sigma)]\} < i$ then return false;
32:             if CHECK_MERGE($\xi(x_p,\sigma)$, $\xi(x_q,\sigma)$, $W$, $i$) =
                    false then return false;
33:             $dynamic(x_p,x_q) := dynamic(x_p,x_q) \cup$
                    $\{(\xi(x_p,\sigma),\xi(x_q,\sigma))\}$;
34:         $root(x_p,x_q) := \min\{root(x'_p,x'_q) | (x'_p,x'_q) \in$
                $dynamic(x_p,x_q)\}$;
35:         $dynamic(x_i,x_j) := dynamic(x_i,x_j) \cup \{(x_p,x_q)\}$;
36: $root(x_i,x_j) := \min\{root(x_p,x_q) | (x_p,x_q) \in$
        $dynamic(x_i,x_j)\}$;
37: if $root(x_i,x_j) \geq pos(x_i,x_j)$ then
        DRAWN_CHECK_MARK $(x_i,x_j)$;
38: return true;
39: function DRAWN_CHECK_MARK($x_i$, $x_j$)
40: if $\{(x_i,x_j),(x_j,x_i)\} \cap \surd\_list \neq \emptyset$ then return;
41: $\surd\_list := \surd\_list \cup \{(x_i,x_j)\}$;
42: for each $(x_p,x_q) \in dynamic(x_i,x_j)$ do
43:     DRAWN_CHECK_MARK($x_p$, $x_q$);
```

Throughout the merging process, in order to generate a control congruence, it is crucial to prevent states from being shared by more than one cell. It is achieved by inserting three filters at lines 5, 7, and 31 in FLA (which are the same as [3][5][12]). The proposed algorithm loops until all the state pairs $(x_i, x_j) \in X \times X$ are checked by lines 4–13. For convenience, the "(i, j)th iteration" denotes the iteration that determines the mergeability of x_i and x_j.

Furthermore, and this is new in FLA, to eliminate any repeated control consistency checks and guarantee that every state pair will be checked exactly once, several additional filters are inserted at lines 8, 17–20, and 26–30. For every state pair, there is only one chance to be checked in FLA; that is, only state pairs that have not been checked can pass through the above newly added filters. Therefore, the role of these filters is to exclude repeated control consistency checks and thereby speed up the localization process.

Comparing with the original CHECK_MERGE function in [3][5][12], we have added the following lines.

(1) Lines 18 and 35 are used to record the state pairs that directly determine the mergeability of x_i and x_j; lines 28 and 33 are used to record the state pairs that directly determine the mergeability of x_p and x_q.
(2) Line 36 is added to obtain the smallest position number of the state pairs that determine the mergeability of x_i and x_j; line 34 is added to obtain the smallest position number of the state pairs that determine the mergeability of x_p and x_q.
(3) Line 23 records the position number of state pair (x_i, x_j) in W.
(4) A new function DRAWN_CHECK_MARK is called by CHECK_MERGE at line 37, where DRAWN_CHECK_MARK consists of lines 39–43. The condition $root(x_i, x_j) \geq pos(x_i, x_j)$ in line 37 is critical, which means that the mergeability of x_i and x_j is determined by only those state pairs having no smaller position number than (x_i, x_j) in W.

In what follows, we briefly describe how FLA updates the two new lists: $\surd$_*list* and $\times$_*list*.

(1) When checking mergeability of states x_i and x_j, if all their corresponding future state pairs (if they exist) satisfy the mergeability condition according to the procedure, and the position numbers of these state pairs are no smaller than that of (x_i, x_j), i.e., (x_i, x_j) can pass line 37, then the mergeability of x_i and x_j is confirmed, and we can add all the related state pairs to $\surd$_*list* by lines 39–43; otherwise the mergeability of x_i and x_j is still pending ((x_i, x_j) cannot pass line 37), and we do not add these related state pairs to $\surd$_*list*.
(2) When checking mergeability of cells $[x_i]$ and $[x_j]$, if all the related state pairs are confirmed to be mergeable, then all the related state pairs will be added to $\surd$_*list* by lines 39–43; if there exits one pair of states that cannot be merged, i.e., the state pair fails at a line of lines 20, 21, or 30–32, then the state pair will be added to $\times$_*list* by line 21, and all the state pairs that are in W, but not in $\surd$_*list*, will be added to $\times$_*list* by line 12.

Based on the above outline, we present in the following the detailed updates of $\surd$_*list* and $\times$_*list* according to the steps of the proposed algorithm.

(1) Suppose that $(x_p, x_q) \notin \mathscr{R}_k$. By line 21, (x_p, x_q) will be added to $\times$_*list*, and CHECK_MERGE $(x_i, x_j, W, i) = false$. It follows from line 12 that all the state pairs that are in W, but not in $\surd$_*list*, will be added to $\times$_*list*, and the checking operation for mergeability of x_i and x_j will be terminated.
(2) Suppose that $(x_p, x_q) \in \mathscr{R}_k$, and (x_p, x_q) has no corresponding future states. Thus $root(x_p, x_q) = pos(x_p, x_q) = 1$, and (x_p, x_q) passes line 37. It follows from line 41 that $\surd\_list := \surd\_list \cup \{(x_p, x_q)\}$. Hence the checking operation for mergeability of x_p and x_q will be terminated, and then the mergeability of other state pairs x'_p and x'_q that have not been checked will continue to be verified.
(3) Suppose that $(x_p, x_q) \in \mathscr{R}_k$, and there exist some events $\sigma_1, \sigma_2, \ldots, \sigma_l \in \Sigma$ such that $\xi(x_p, \sigma)!$ and $\xi(x_q, \sigma)!$, where $\sigma \in \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$.
    (3a) If $\xi(x_p, \sigma)$ and $\xi(x_q, \sigma)$ have no corresponding future states, where $\sigma \in \{\sigma_1, \sigma_2, \ldots, \sigma_l\}$, then the mergeability of $\xi(x_p, \sigma)$ and $\xi(x_q, \sigma)$ is checked as follows.
        (i) If they satisfy the condition of line 26, then go to Step (v) below; otherwise go to Step (ii).
        (ii) If they fail at line 27, i.e., $(\xi(x_p, \sigma), \xi(x_q, \sigma))$ or $(\xi(x_q, \sigma), \xi(x_p, \sigma))$ is in W, by lines 27–29, then $(\xi(x_p, \sigma), \xi(x_q, \sigma))$ is added to *dynamic*(x_p, x_q), and go to Step (v) below; otherwise go to Step (iii).
        (iii) If they fail at line 30 or line 31, by line 12, then all the state pairs that are in W, but not in $\surd$_*list*, will be added to $\times$_*list*, and the checking operation for mergeability of x_i and x_j will be terminated.
        (iv) If they fail at line 32, then CHECK_MERGE $(x_i, x_j, W, i) = false$. It follows from line 12 that all the state pairs that are in W, but not in $\surd$_*list*, will be added to $\times$_*list*, and the checking operation for mergeability of x_i and x_j will be terminated.
        (v) The mergeability of the other corresponding future states that have not been checked will continue to be verified by repeating Steps (i)–(v). Assume that all the future state pairs $(\xi(x_p, \sigma), \xi(x_q, \sigma))$ pass lines 26–33. Then we calculate the smallest position number of (x_p, x_q) in W by the dynamic list of (x_p, x_q). If (x_p, x_q) satisfies the condition of line 37, then all the related state pairs will be added to $\surd$_*list*; otherwise the mergibility of x_p and x_q is still pending, and (x_p, x_q) will not be added to $\surd$_*list*. Hence the checking operation for mergeability of x_p and x_q is terminated, and the mergeability of other states x'_p and x'_q that have not been checked will continue to be checked.

(3b) If there exists an event $\sigma' \in \Sigma$ such that $\xi(\xi(x_i,\sigma),\sigma')!$ and $\xi(\xi(x_j,\sigma),\sigma')!$, then the mergeability of $\xi(\xi(x_i,\sigma),\sigma')$ and $\xi(\xi(x_j,\sigma),\sigma')$ will be checked recursively in the same way as Step (3a).

An example that illustrates the steps of FLA in detail can be found in the conference precursor [20].

3.2 Complexity Analysis

In this subsection, we study the time complexity and space complexity of the proposed FLA. First, we analyze the time complexity.

Theorem 1 *Given a supervisor **SUP** with state set X and $|X| = n$, FLA terminates in finite steps, has (worst-case) time complexity $O(n^2)$, and the generated $\mathscr{C}_k$ is a control congruence of X.*

Proof. According to lines 4 and 6, the MAIN procedure can call the function CHECK_MERGE (at line 10) at most $n(n-1)/2$ times. In each call of CHECK_MERGE, the main operation is to check whether or not $(x_i,x_j) \in \mathscr{R}_k$ for fixed i and j involved in that call (the same as [3][5][12]). The key here is to show that by lines 5, 7, 8, 17–19, and 26–31 that control consistency of (x_i,x_j) will be checked exactly once in the entire process (from start to termination of FLA). To prove this statement, we need to consider two cases as follows.

(1) Assume that control consistency of (x_i,x_j) has been checked in a previous (i',j')th iteration (i.e., $i' < i$, or $i' = i$ and $j' < j$). Thus (x_i,x_j) must be in either $\surd$*_list* or $\times$*_list*. (Case i) If $(x_i,x_j) \in \surd$*_list*, by line 13, then states x_i and x_j had been merged into one cell in the (i',j')th iteration. According to lines 5, 7, 26, and 31, the (i,j)th iteration will be terminated, and thus control consistency of (x_i,x_j) will not be checked again in the remaining iterations. (Case ii) If $(x_i,x_j) \in \times$*_list*, by lines 8, 20, and 30, then control consistency of (x_i,x_j) will not be checked again in the remaining iterations.

(2) Assume that control consistency of (x_i,x_j) has not been checked in any previous iterations and will be checked in the current (i,j)th iteration. (Case i) If $(x_i,x_j) \notin \mathscr{R}_k$, by line 21, then (x_i,x_j) will be added to $\times$*_list*, and the (i,j)th iteration will be terminated. By lines 8, 20, and 30, control consistency of (x_i,x_j) will not be checked again in the remaining iterations. (Case ii) If $(x_i,x_j) \in \mathscr{R}_k$, by line 22, then (x_i,x_j) will be added to W, by lines 17–19 and lines 27–29, control consistency of (x_i,x_j) will not be checked again in the (i,j)th iteration. When the (i,j)th iteration is finished, by (1) above, control consistency of (x_i,x_j) will not be checked again in the remaining iterations.

Thus, control consistency of each state pair will be checked exactly once. So the total number of checking operations (calls from the MAIN procedure and the recursive calls from the functions CHECK_MERGE) is at most $|\Sigma|n(n-1)/2$, where $|\Sigma|$ is due to line 25.

Therefore FLA terminates in finite steps, and has (worst-case) time complexity $O(n^2)$.

Finally, since lines 5, 7, and 31 ensure no element overlapping among cells in $\mathscr{C}_k$, the resulting $\mathscr{C}_k$ is a partition of X (the same as [3][5][12]). It then suffices to show that $\mathscr{C}_k$ is a control cover; namely the two conditions (1) and (2) of Definition 2 in Section 2 are satisfied. That condition (1) holds is guaranteed by line 21: no two states in the same cell can be control inconsistent; and (2) holds by lines 25–33: recursively check control consistency of their future states reached by identical strings. Therefore we derive that the resulting $\mathscr{C}_k$ is a control congruence, as required. □

Theorem 1 shows that control consistency of each state pair will be checked exactly once in FLA. Therefore in comparison with the reduction algorithm in [12] and the localization algorithm in [3][5], the time complexity of FLA is reduced from $O(n^4)$ to $O(n^2)$.

Example 1 *To illustrate the reduction in time complexity, consider an example similar to that in Fig. 1. Given a supervisor **SUP** $= (X,\Sigma,\xi,x_0,X_m)$ which has the same structure as the one in Fig. 1, but with $X = \{x_0,x_1,\ldots, x_{n-1}\}$. The number of state pairs is $n(n-1)/2$. The state pair transition diagram of **SUP** is displayed in Fig. 2. Suppose that all the state pairs but (x_{n-2},x_{n-1}) are control consistent. If we adopt the algorithm in [3][5][12], then control consistency of every state pair (except for the first (x_0,x_1)) needs to be checked repeatedly (for the same reason as the example in Fig. 1). In fact the total number of checking operations is $n(n-1)/2 + (n(n-1)/2 - 1) + \cdots + 2 + 1 = (n^4 - 2n^3 + 3n^2 - 2n)/8$. If we use FLA, then all the checking operations only take place once and only in the first iteration for determining the mergeability of states x_0 and x_1, that is, control consistency of all the state pairs will not be checked again in the remaining iterations. Hence the total number of checking operations is $n(n-1)/2$. Therefore, this example demonstrates that the proposed algorithm improves the computational efficiency.*

Remark 1 *To avoid any repeated control consistency checks, $\surd$_list, $\times$_list, pos(x_i,x_j), dynamic(x_i,x_j) and root(x_i,x_j) are used in FLA. Using these new elements, on the other hand, makes the order of merging states in FLA* different *from the order used in the reduction algorithm in [12] and the localization algorithm in [3][5]. As a result, the control congruence $\mathscr{C}_k$ generated by FLA is generally different from those generated in [3,5,12]. In general, the state sizes of the resulting reduced/local controllers are* incomparable. *An example showing that FLA generates a local controller with a smaller state size than that generated by algorithms in [3][5][12] can be found in the conference precursor [20]. The reverse case is also possible, as can be seen in the examples given in Section IV below.*

The improvement of time complexity achieved by FLA is by means of properly using a number of new elements (e.g. $\surd$*_list*, $\times$*_list*, *pos*(x_i,x_j), *dynamic*(x_i,x_j) and *root*(x_i,x_j)). Thus intuitively, these new elements increase FLA's (worst-case) space complexity. The following is the space complex-

$$\begin{tikzcd}(x_0,x_1) \arrow[r, "\sigma_1"] & (x_0,x_2) \arrow[r, "\sigma_2"] & (x_0,x_3) \arrow[r, "\sigma_3"] & \cdots \arrow[r, "\sigma_{n(n-1)/2-1}"] & (x_{n-2},x_{n-1})\end{tikzcd}$$

Fig. 2. State pair transition diagram.

ity of FLA.

Theorem 2 *Given a supervisor **SUP** with state set X and $|X| = n$, FLA has (worst-case) space complexity $O(n^4)$.*

Proof. The storage space of FLA includes W, $\surd$_*list*, $\times$_*list*, $pos(x_i,x_j)$, $dynamic(x_i,x_j)$ and $root(x_i,x_j)$. For the state pair (x_i,x_j), in the worst case W has $n(n-1)/2$ state pairs, that is, all the state pairs are included in W by line 22. Note that W is initialized as an empty set after each state pair is checked by line 9, so the storage space for storing W is then $O(n^2)$. Since all state pairs are uniquely stored in either $\surd$_list by line 12 or $\times$_list by line 41, the storage spaces of $\surd$_list and $\times$_list are both $O(n^2)$. $dynamic(x_i,x_j)$, initialized at line 24, stores all the state pairs in the waiting list W. To determine the mergeability of x_i and x_j, all their corresponding future states are checked recursively at lines 25-33. Hence for each state pair (x_i,x_j), there are at most $n(n-1)/2$ state pairs need to be checked, which implies that the storage space for storing $dynamic(x_p,x_q)$ (lines 28 and 35) is then $O(n^2)$. There are $n(n-1)/2$ state pairs (lines 4 and 6), so the storage spaces of $dynamic(x_i,x_j)$ is $O(n^4)$. In addition, $pos(x_i,x_j) \in \{1,2,\ldots,n(n-1)/2\}$ and $root(x_i,x_j) \in \{1,2,\ldots,n(n-1)/2\}$ record the position number and the minimal position number of (x_i,x_j) stored in the waiting list W by line 23 and line 36, respectively. There exist at most $n(n-1)/2$ state pairs (x_p,x_q) to be checked by the function CHECK_MERGE (lines 15-16), so the storage spaces for storing $pos(x_p,x_q)$ and $root(x_p,x_q)$ are both $O(n^2)$. There are $n(n-1)/2$ state pairs totally (lines 4 and 6), thus the storage spaces for storing $pos(x_i,x_j)$ and $root(x_i,x_j)$ are $O(n^4)$. Hence the total storage space for storing all the associated lists is $O(n^4)$. □

Remark 2 *The function* CHECK_MERGE *in FLA has the same arguments x_i, x_j, W, and i as those in [3][5][12]. On the other hand, $pos(x_i,x_j)$, $dynamic(x_i,x_j)$, $root(x_i,x_j)$, and two global variables $\surd$_list and $\times$_list are newly added in FLA. From the above discussion, we know that the space complexities of W, $\surd$_list, and $\times$_list are $O(n^2)$. However, the space complexities of $pos(x_i,x_j)$, $dynamic(x_i,x_j)$, and $root(x_i,x_j)$ are $O(n^4)$ in the worst case. Our proposed algorithm has higher space complexity than those in [3][5][12]. Although FLA consumes extra storage space, it substantially improves the time complexity. It is an open problem if it is possible to design a reduction/localization algorithm whose time complexity and space complexity are both $O(n^2)$.*

To end this section, we provide two examples to illustrate the space complexity of FLA. The first example shows a case in which the worst-case space complexity $O(n^4)$ is needed. On the other hand, the second example shows a case where only an $O(n^2)$ space complexity is required.

Example 2 *Given a supervisor **SUP** $= (X,\Sigma,\xi,x_0,X_m)$, where $X = \{x_0,x_1,\ldots, x_{n-1}\}$. We consider the state pair transition diagram of **SUP** given in Fig. 2. Suppose that all the state pairs in Fig. 2 are control consistent. Initially, we have $\surd$_list $= \emptyset$, $\times$_list $= \emptyset$, and $dynamic(x_i,x_j) = \{(x_i,x_j)\}$, where $i \in \{0,1,\ldots,n-2\}$, $j \in \{1,2,\ldots,n-1\}$, and $i < j$. Notice that control consistency of all the state pairs will be checked in the $(0,1)$th iteration. After the $(0,1)$th iteration, all the state pairs will be added to $\surd$_list. Then $\times$_list $= \emptyset$. So, the storage space for storing $\surd$_list and $\times$_list is $O(n^2)$. It follows from Theorem 1 that control consistency of all the state pairs will not be checked again in the remaining iterations. Since $dynamic(x_0,x_1) = \{(x_0,x_1)\} \cup dynamic(x_0,x_2)$, $dynamic(x_0,x_2) = \{(x_0,x_2)\} \cup dynamic(x_0,x_3)$, $\ldots$, $dynamic(x_{n-2},x_{n-1}) = \{(x_{n-2},x_{n-1})\}$, we have $dynamic(x_0,x_1) = \{(x_0,x_1),(x_0,x_2),\ldots,(x_{n-2},x_{n-1})\}$, $dynamic(x_0,x_2) = \{(x_0,x_2),(x_0,x_3),\ldots,(x_{n-2},x_{n-1})\}$, $\ldots$, $dynamic(x_{n-2},x_{n-1}) = \{(x_{n-2},x_{n-1})\}$. Thus the total number of state pairs in all the associated lists is $n(n-1)/2 + (n(n-1)/2-1) + \cdots + 2 + 1 = (n^4 - 2n^3 + 3n^2 - 2n)/8$. Consequently, the storage space for storing all the associated lists is $O(n^4)$. Furthermore, by Theorem 2, the storage space for storing W is $O(n^2)$. Therefore, the space complexity of FLA is $O(n^4)$.*

Example 3 *We consider the same supervisor **SUP** and the same state pair transition diagram as those in Example 2. Suppose that all the state pairs but (x_{n-2},x_{n-1}) are control consistent. Similarly, control consistency of all the state pair will be checked only in the $(0,1)$th iteration. After the $(0,1)$th iteration, all the state pairs will be added to $\times$_list. Thus $\surd$_list $= \emptyset$. So the storage space for storing $\times$_list and $\surd$_list is $O(n^2)$. We also have $dynamic(x_i,x_j) = \{(x_i,x_j)\}$ for all $i \in \{0,1,\ldots,n-2\}$, $j \in \{1,2,\ldots,n-1\}$, and $i < j$. Hence the storage space for storing all the associated lists is $O(n^2)$. The storage space for storing W is $O(n^2)$. Based on the above discussion, FLA has space complexity $O(n^2)$ in this case.*

4 Empirical Study

In this section, we empirically compare the computing times of FLA and the localization algorithm in [3][5] with different examples, where the computing time is obtained by a software tool TCT [5].

4.1 Example 1 Continued

In Example 1, it was shown that FLA improves the worst-case time complexity by the order $O(n^2)$, as compared with the algorithms in [3][5][12]. Here n is the number of states in the supervisor. To empirically validate this improvement, we have tested this example for n ranging from 100 to 300 (with interval 20). The result is displayed in Table 1; the

Table 1
Comparison of computing times (in seconds) by FLA and the algorithm in [5] for Example 1

n	100	120	140	160	180	200	220	240	260	280	300
[5]	1	2	3	8	15	31	45	56	81	117	168
FLA	<1	<1	<1	<1	1	1	1	2	2	2	4

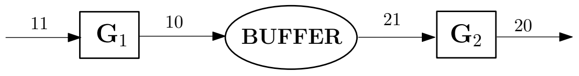

Fig. 3. Small Factory.

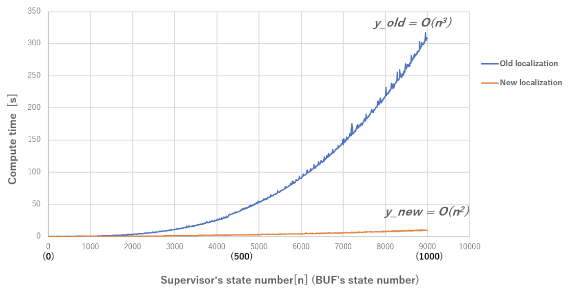

Fig. 4. Comparison of the computing time of FLA and the localization algorithm in [3][5] for Small Factory. The horizontal axis represents the number of supervisor's states. The horizontal brackets (0), (500), and (100) represent the number of buffer's state. The vertical axis is the computing time in seconds.

unit is second for rows 2 and 3. As one can observe, the increase of computing time by FLA is much slower than that by the algorithm in [5] (similarly by those in [3,12]). This test empirically confirms our theoretic analysis of improved efficiency.

4.2 *Small Factory*

We consider a version of Small Factory in [18] consisting of one input machine $\mathbf{G}_1$ and one output machine $\mathbf{G}_2$ as arranged in Fig. 3. We take a buffer with different slots and consider only the buffer specification (protecting buffer against overflow and underflow). We consider the case that the capacity of buffer increases from 2 to 1000, and show the computing time of FLA and the localization algorithm in [3][5], respectively, in Fig. 4. The horizontal axis represents the number of supervisor's state. The horizontal brackets represent the number of buffer's state. The vertical axis is the computing time in seconds. The blue curve represents the computing time using the localization algorithm in [3][5]. The red curve represents the computing time using FLA.

As shown in Fig. 4, in this example the computing time of the localization algorithm in [3][5] is of the order $O(n^3)$, while it is $O(n^2)$ for FLA. Moreover, we have confirmed that all the local controllers computed by both FLA and the localization algorithm in [3][5] have exactly the same state numbers.

4.3 *Transfer Line*

As displayed in Fig. 5, Transfer Line consists of two machines **M1**, **M2** followed by a test unit **TU**; these three agents are linked by two buffers **B1**, **B2**. A workpiece entering the system is first processed by **M1** and stored in **B1**, then processed by **M2** and stored in **B2**. A processed workpiece tested by **TU** may be accepted or rejected; if accepted, it is released from the system; if rejected, it is returned to **B1** for reprocessing by **M2**.

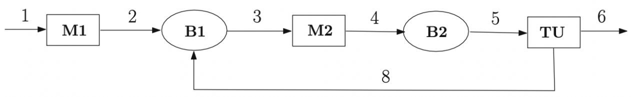

Fig. 5. Transfer Line.

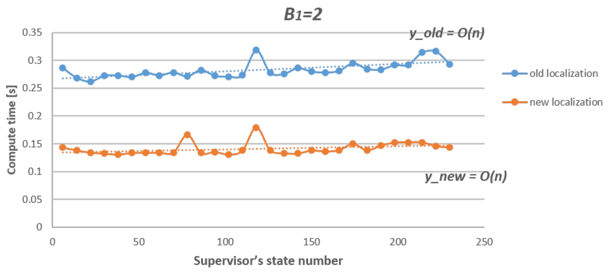

Fig. 6. Comparison of the computing time for the case that the capacity of **B1** is fixed at 2 and the capacity of **B2** increases from 2 to 30.

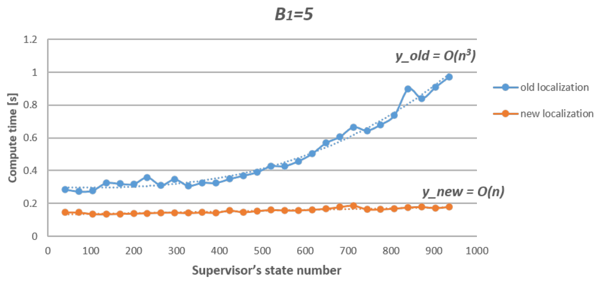

Fig. 7. Comparison of the computing time for the case that the capacity of **B1** is fixed at 5 and the capacity of **B2** increases from 2 to 30.

Here we show the computing times of FLA and the localization algorithm in [3][5], respectively for different cases. Fig. 6 and Fig. 7 give the computation results for the cases that the capacity of buffer **B1** is fixed at 2 and 5 respectively, while the capacity of **B2** increases from 2 to 30. Fig. 8 and Fig. 9 show the results for the cases that the capacity of buffer **B2** is fixed at 2 and 5 respectively, while the capacity of **B1** increases from 2 to 30. Same as Fig. 4, the horizontal axis represents the number of supervisor's state. The vertical axis is the computing time in seconds. The blue curve represents the time using the localization algorithm in [3][5]. The red curve represents the time using FLA.

Observe that in all these cases the computing time of FLA and the localization algorithm in [3] [5] are close (within a fraction of a second) due to fairly small number of supervisor's state. In all cases, FLA has smaller or the same trend of increase in computing time as compared to localization algorithm in [3][5].

Tables 2, 3, 4 present the state numbers of local controllers computed in the experiments corresponding to Fig. 6–9. There are three controllable events $\{1,\ 3,\ 5\}$ in Transfer Line, thus three corresponding local controllers whose average state numbers are reported in Tables 2, 3, 4 respectively.

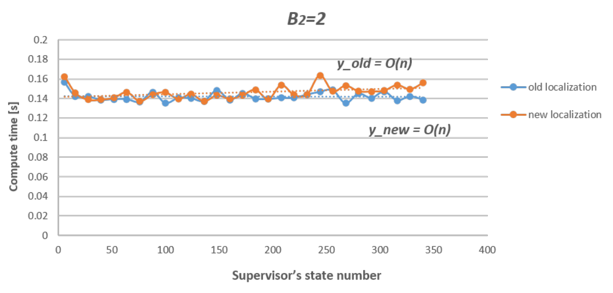

Fig. 8. Comparison of the computing time for the case that the capacity of **B2** is fixed at 2 and the capacity of **B1** increases from 2 to 30.

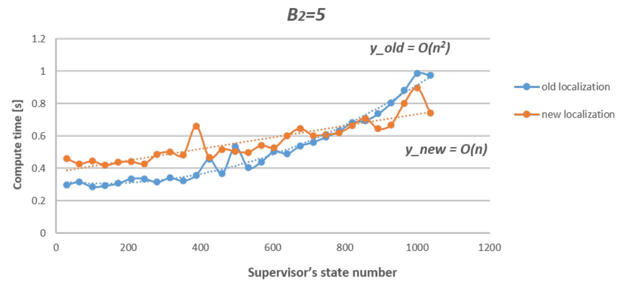

Fig. 9. Comparison of the computing time for the case that the capacity of **B2** is fixed at 5 and the capacity of **B1** increases from 2 to 30.

Table 2
Average state comparison of the local controller for event 1.

Transfer Line	$\lvert B1\rvert=2$ $\lvert B2\rvert=(2,3,\ldots,30)$	$\lvert B1\rvert=5$ $\lvert B2\rvert=(2,3,\ldots,30)$	$\lvert B2\rvert=2$ $\lvert B1\rvert=(2,3,\ldots,30)$	$\lvert B2\rvert=5$ $\lvert B1\rvert=(2,3,\ldots,30)$
Old Loc. Alg.	56	79	16	79
Fast Loc. Alg.	56	194	16	148

Table 3
Average state comparison of the local controller for event 3.

Transfer Line	$\lvert B1\rvert=2$ $\lvert B2\rvert=(2,3,\ldots,30)$	$\lvert B1\rvert=5$ $\lvert B2\rvert=(2,3,\ldots,30)$	$\lvert B2\rvert=2$ $\lvert B1\rvert=(2,3,\ldots,30)$	$\lvert B2\rvert=5$ $\lvert B1\rvert=(2,3,\ldots,30)$
Old Loc. Alg.	31	79	31	78
Fast Loc. Alg.	2	78	30	79

Table 4
Average state comparison of the local controller for event 5.

Transfer Line	$\lvert B1\rvert=2$ $\lvert B2\rvert=(2,3,\ldots,30)$	$\lvert B1\rvert=5$ $\lvert B2\rvert=(2,3,\ldots,30)$	$\lvert B2\rvert=2$ $\lvert B1\rvert=(2,3,\ldots,30)$	$\lvert B2\rvert=5$ $\lvert B1\rvert=(2,3,\ldots,30)$
Old Loc. Alg.	46	159	31	239
Fast Loc. Alg.	47	158	2	238

It is observed that the state numbers are generally incomparable between local controllers computed by FLA and the localization algorithm in [3][5]. In many cases the state numbers are close, but there also exist situations where one is much smaller than the other. Finally due to the current implementation of these algorithms in TCT [5], in which it is challenging to record consumed memory space, we leave an empirical study of space complexity in our future work by a new implementation in a possibly different software tool.

5 CONCLUSIONS

In this paper, we have extended the previous work on supervisor reduction and supervisor localization for DES. We have presented a fast algorithm for supervisor reduction/localization of DES by employing a mergeable list and a non-mergeable list to store the checked state pairs. We have shown that every state pair will be checked exactly once in the whole process, and analyzed the time complexity and space complexity of the proposed algorithm. We have verified the time and space complexities of the fast algorithm by several examples. In future work, we aim to further evaluate space complexity of the proposed algorithm, and inquire a design of a localization algorithm which has both time and space complexity $O(n^2)$.

References

[1] Lucas VR Alves and Patrícia N Pena. On the reduction and localization of synchronizing supervisors. In *2021 American Control Conference (ACC)*, pages 4491–4496. IEEE, 2021.

[2] Kai Cai, Alessandro Giua, and Carla Seatzu. Consistent reduction in discrete-event systems. *Automatica*, 142:110333, 2022.

[3] Kai Cai and Walter Murray Wonham. Supervisor localization: a top-down approach to distributed control of discrete-event systems. *IEEE Transactions on Automatic Control*, 55(3):605–618, 2010.

[4] Kai Cai and Walter Murray Wonham. Supervisor localization of discrete-event systems based on state tree structures. *IEEE Transactions on Automatic Control*, 59(5):1329–1335, 2013.

[5] Kai Cai and Walter Murray Wonham. Supervisor localization: A top-down approach to distributed control of discrete-event systems, 2016.

[6] C. G. Cassandras and S. Lafortune. Introduction to discrete event systems (3rd ed.), 2021.

[7] Feng Lin and Walter Murray Wonham. Decentralized supervisory control of discrete-event systems. *Information sciences*, 44(3):199–224, 1988.

[8] Liyong Lin, Yuting Zhu, and Rong Su. Synthesis of covert actuator attackers for free. *Discrete Event Dynamic Systems*, 30:561–577, 2020.

[9] Liyong Lin, Yuting Zhu, Ruochen Tai, Simon Ware, and Rong Su. Networked supervisor synthesis against lossy channels with bounded network delays as non-networked synthesis. *Automatica*, 142:110279, 2022.

[10] Robi Malik. Supervisor reduction by hiding events. *IFAC-PapersOnLine*, 53(6):1–6, 2020.

[11] Peter J Ramadge and W Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[12] Su Rong and Walter Murray Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, 14:31–53, 2004.

[13] Rong Su and Walter Murray Wonham. What information really matters in supervisor reduction? *Automatica*, 95:368–377, 2018.

[14] Ruochen Tai, Liyong Lin, and Rong Su. Identification of system vulnerability under a smart sensor attack via attack model reduction. *IEEE Control Systems Letters*, 6:2948–2953, 2022.

[15] Anthony Faria Vaz and William Murray Wonham. On supervisor reduction in discrete-event systems. *International Journal of Control*, 44(2):475–491, 1986.

[16] Yosef Willner and Michael Heymann. Supervisory control of concurrent discrete-event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[17] W. M. Wonham. Design software: XPTCT. *Systems Control Group, ECE Dept., University of Toronto, Toronto, ON, Canada*, 2017.

[18] W Murray Wonham, Kai Cai, et al. Supervisory control of discrete-event systems, 2019.

[19] W Murray Wonham and Peter J Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

[20] Lihua Wu, Kai Cai, Renyuan Zhang, and Yingying Liu. An $o(n^2)$ algorithm for supervisor reduction/localization of discrete-event systems. *IFAC-PapersOnLine*, 53(4):211–216, 2020.