ITC 4/50

Information Technology and Control
Vol. 50 / No. 4 / 2021
pp. 769-785
DOI 10.5755/j01.itc.50.4.29031

Analyzing Isolation in Mobile Systems

Received 2021/05/01 | Accepted after revision 2021/10/01

http://dx.doi.org/10.5755/j01.itc.50.4.29031

HOW TO CITE: Hong, Z., Jiang, J.-M., Shu, H. (2021). Analyzing Isolation in Mobile Systems. *Information Technology and Control*, 50(4), 769-785. https://doi.org/10.5755/j01.itc.50.4.29031

Analyzing Isolation in Mobile Systems

Zhong Hong
College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350007, China;
e-mail: fjfzhz@fjnu.edu.cn

Jian-Min Jiang, Hongping Shu
College of Software Engineering, Chengdu University of Information Technology, Chengdu 610103, China;
e-mails: jjm@cuit.edu.cn, shp@cuit.edu.cn

Corresponding authors: fjfzhz@fjnu.edu.cn, jjm@cuit.edu.cn

As a safety-critical issue in complex mobile systems, isolation requires two or more mobile objects not to appear in the same place simultaneously. To ensure such isolation, a scheduling policy is needed to control and coordinate the movement of mobile objects. Unfortunately, existing task scheduling theories fails in providing effective solutions, because it is hardly possible to decompose a complex mobile system into multiple independent tasks. To solve this problem, a more fine-grained event scheduling is proposed in this paper to generate scheduling policies which can ensure the isolation of mobile objects. After defining event scheduling based on event-based formal models called dependency structures, a new event scheduling theory for mobile systems is developed accordingly. Then an algorithm for generating an event scheduling policy is proposed to implement the required isolation. Simulation experiments are conducted to prove the result of our theoretical analysis and show the effectiveness and scalability of the approach.

KEYWORDS: Mobility, Isolation, Scheduling Policy, Ambient, Collision Deadlock.

1. Introduction

In safety-critical mobile systems like intelligent transportation systems (ITS) and unmanned transportation systems, there is a compelling need to guarantee the safety and security of such systems, which has become the subject of intense research worldwide. In these systems, isolation requires two or more mobile objects not to appear in the same place simultaneously. For example, consider a scenario where several driverless vehicles need to cross a road inter-section without traffic lights. To prevent from potential collisions, these vehicles must pass through the intersection in some order, under the control of certain scheduling policy. In other words, they must be isolated from each other during the whole process. Thus an effective scheduling theory is needed to generate a scheduling policy so as to ensure the isolation relationships among mobile objects in such systems.

Task scheduling theories, e.g, [25], [6], [31], [38], [23], [24], [33], have been studied for decades under the strong assumption that each scheduled task is independent of the others. The later research on scheduling theories, e.g., [38], [23], [24], [33], [27], [22], dives deep into task scheduling and schedulability analysis, especially in cyber-physical systems [20]. Task scheduling mainly considers how to generate the optimal scheduling policies while schedulability analysis aims to check if there exists the possibility of constraint violations during the system's evolvement process. However, due to the inherent complexity of scheduling, existing work is far from enough to solve the isolation problems. Since mobile objects and environments are closely connected and intensively interact in a mobile system, it is hardly possible to separate a complex mobile system into independent tasks. Thus we cannot directly obtain a scheduling policy by using existing methods and techniques. Therefore, developing a new scheduling theory for complex mobile systems has become the most emergent matter.

To solve the complex scheduling problems, Jiang et al. [18] have proposed event scheduling, which is more fine-grained than task scheduling. An event is generally the occurrence of an action or activity, and a task (usually in the forms of threads, processes, or complex activities) may consist of several events. A complex scheduling problem can be decomposed into several event scheduling sub-problems, while it is unable to be divided into independent tasks. An event-based formal model called a *dependency structure* [18], [17], [15], [14] has been chosen to model complex mobile systems because it can not only conveniently express the concurrency, synchronization and loop, but also directly model mobility without the need for further refinement [18]. Inspired by the ambient calculus [10], two kinds of special moving events, namely, *entering* and *exiting* an ambient, have been proposed in [18] for modeling mobility. So whether these events occur simultaneously in the same place (ambient) is employed to express the isolation relations among mobile objects.

Contribution. This paper makes the following contributions:

- A novel notion of a schedule is introduced based on the execution process of a mobile system, which is different from that based on the priority in [17]. It shortens the length of scheduling sequences and improves the efficiency of obtaining such a scheduling sequence.
- The isolation of mobile objects in a complex mobile system is formally defined and the avoidance of collision deadlocks during the isolation control process is discussed in detail.
- A new algorithm is proposed for automatically generating a schedule, providing support for the implementation of safety isolation in a mobile system.
- The approach is especially applied to solving the problems of intersection isolation in real systems, so as to prove its correctness and effectiveness.

Structure. The remainder of this article is structured as follows. Section 2 gives an overview of the related work. Section 3 presents a typical intersection scenario as a running example. Section 4 introduces the dependency structure used as a system model. Section 5 defines the notion of a schedule and Section 6 discusses the decomposition and composition of scheduling. Section 7 defines the notion of isolation. Section 8 derives an algorithm to automatically generate a schedule. Section 9 gives a case study and some experiments, followed by conclusion in Section 10.

2. Related Work

In this section, the related work is introduced and discussed, from the aspects of scheduling and intersection isolation respectively.

2.1. Related Work on Scheduling

Since the first introduction by Liu and Layland [25] in 1973 on the Rate Monotonic (RM) Scheduling Algorithm, scheduling has been widely studied and applied in many different areas. Specifically, the real-time scheduling theory has been divided into several important categories: fixed priority scheduling (e.g. [35]), dynamic priority scheduling (e.g. [9]), soft real-time scheduling (e.g. [2]) and feedback scheduling (e.g. [28]). Unlike traditional computing systems, cyber-physical systems (CPS) are integrations of computation, networking, and physical processes, which have become a hot topic of researches in recent years. Due to limited space, only the most related work on the scheduling of CPS will be discussed below.

For a class of CPS whose behaviors are regulated by feedback control laws, Zhang et al. [38] co-design the control law and the task scheduling algorithm for predictable performance and power consumption for both the computing and the physical systems. To support scheduling in real-time CPS, Kim et al. [19] extend the fork-join parallel task model by applying the task stretch transformation. In the literature [30], Schneider et al. present a multi-layered schedule synthesis scheme for mixed-criticality cyber-physical systems (MCCPS) so as to jointly schedule deadline-critical, and QoC-critical tasks at different scheduling layers. Liu et al. [26] dive deep into the characteristics of temporal data dissemination in vehicular CPS and propose a scheduling algorithm to analyze the time bound for serving requests.

Besides the above work, there also exist some researches, which firstly provide a metric model, then propose a scheduling algorithm and finally improve the algorithm using different methods. Li and Negi [23], [24] contribute to the scheduling for CPS by identifying and addressing a general class of scheduling-type applications for physical networks. He considers the class of optimal scheduling algorithms in the quasi-static regime. Tang et al. [33] propose an abstract heat flow model to describe thermal interference in a distributed CPS and derive a unified scheduling and verification methodology based on it.

As mentioned before, these existing work always investigates the scheduling problems based on traditional task scheduling, and the efforts hardly reason about scheduling. Based on more fine-grained event scheduling, Jiang et al. [16] attempt to model and analyze isolation from the perspective of mobility, which is the most related to this work. However, their work focuses on fixed priority event scheduling and generates a priority scheduling sequence by computing an execution sequence of a system, which do not consider event scheduling with a loop. In this work, we explore dynamic event scheduling and generate a scheduling sequence based on the states of a system. The generated scheduling sequence may contain a loop. Thus this work is actually an extension of [16] aiming to solve the safety isolation problems in complex mobile systems, which is original and different from all these related work.

2.2. Related Work on Intersection Isolation

To avoid potential collisions and other faults at intersections while maximizing the throughput, researchers from worldwide have developed various intersection isolation approaches for vehicles so as to ensure their safety passage.

The first category of protocols are agent-based and reservation-based. In the literature [12], Dresner and Stone present a protocol named Autonomous Intersection Management (AIM) for fully autonomous vehicles so that they could reserve conflict-free trajectories in advance by calling ahead to a reservation manager agent at the intersection, which can effectively reduce the delay of vehicles caused by intersections but might introduce a single point of failure for its heavy reliance on the communications between the vehicles and the manager agent. This work is then extended to be compatible with partially automated vehicles in [32]. Lee and Park [21] also provide an agent-based approach but modeling such problem as a Constrained Nonlinear Optimization Problem, which is solved by using parallel computations of three optimization methods. Besides, Perronnet et al. [29] focus on the deadlock prevention under real-time conditions in a network of intersections and propose a reservation-based hierarchical approach to improve the overall throughput without deadlock.

The second category of protocols are priority-based, in which vehicles with higher priority are allowed to cross the conflicting areas prior to those with lower priority. Several spatio-temporal intersection protocols (STIP) are proposed by Azimi et al. [7], which assign priorities to vehicles according to their arrival times at the intersection. Such priority-based intersection protocols are extended by Aoki and Rajkumar

[4] to merge protocol by using both V2V communications and in-vehicle perception systems, which also realize the coordination between autonomous vehicles and human-driven vehicles.

The third category of protocols are derived from the Ballroom Intersection Protocol (BRIP) [8], in which, a specific spatio-temporal pattern is tailored to each particular intersection so that all vehicles crossing the intersection must follow it. Based on BRIP, Aoki and Rajkumar [3] present a more general and fault-tolerant version named the Configurable Synchronous Intersection Protocol (CSIP), which is more robust even in the case of GPS inaccuracies and other failures. Moreover, the introduction of inter-vehicular distances makes CSIP much more acceptable and comfortable to human passengers.

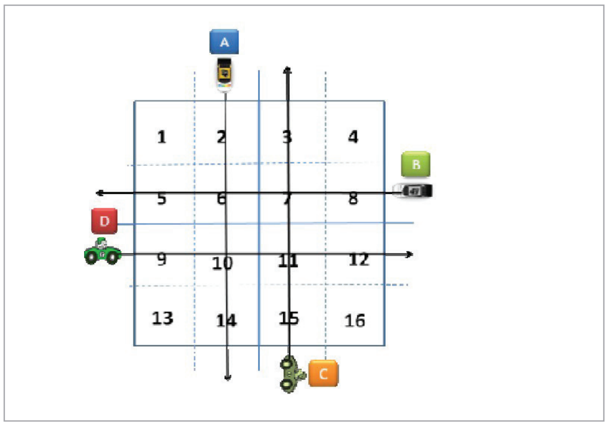
In addition, Wu et al. [36] regard the problem of intersection control as a new variant of the classic mutual exclusion problem rather than an optimization problem, and design both centralized and distributed algorithms to solve such a problem. This method is easy and simple because it avoids the involvement of traditional optimization process. Besides traditional static road intersections, Aoki and Rajkumar [5] study Dynamic Intersections (DI) and present a cooperative dynamic intersection protocol to avoid vehicle collisions at DI.

Compared with these existing intersection control approaches, our work offers a totally different solution based on formal methods, which provides a solid theoretic foundation and supports analyzing and reasoning at a higher abstract level. Moreover, our approach can be more general and less dependent on communication, which can be widely applied for nearly any sort of complicated driving contexts that may be encountered in the real world.

3. Running Example

Here we present a typical autonomous driving scenario (see Figure 1) as a running example, where four autonomous vehicles *A*, *B*, *C* and *D* need to pass through an intersection, which is separated into a number of small cells. Each cell is marked with a unique identifier as shown in Figure 1. At the current time, the four vehicles *A*, *B*, *C* and *D* are preparing to cross the intersection along the routes c2 → c6 → c10 → c14, c8 → c7 → c6 → c5, c15 → c11 → c7 → c3 and c9 → c10 → c11 → c12, respectively.

Figure 1
A running example with four vehicles crossing one intersection (from the literature [7])



Similar to the concept introduced by the ambient calculus [10], here we consider the cells as ambients where mobile objects can get in and out at their will. Obviously, in this running example system, each cell can be regarded as an ambient and the vehicles *A*, *B*, *C* and *D* are four mobile objects which can enter or exit these grid cells when moving along their routes.

The notations $\mathcal{A}$ and $\mathcal{M}$ are used here to denote the set of ambients and the set of mobile objects respectively. Thus the event of a mobile object M ($M \in \mathcal{M}$) entering an ambient A ($A \in \mathcal{A}$) is denoted by en^M_A. In fact, it is enough to only use the two movement events (*entering* and *exiting* events) for specifying the mobility in such a mobile system [18], [17]. To further simplify modeling specification, in this work we only consider the entering events because the event of entering one ambient naturally means the event of exiting the previously occupied ambient. Thus, we have the following movement events for the running example system: $en^A_{c2}, en^A_{c6}, en^A_{c10}, en^A_{c14}, en^B_{c8}, en^B_{c7}, en^B_{c6}, en^B_{c5}, en^C_{c15}, en^C_{c11}, en^C_{c7}, en^C_{c3}, en^D_{c9}, en^D_{c10}, en^D_{c11}, en^D_{c12}$.

4. System Model

In this section, a fine-grained event-based formal model called *dependency structure* [18], [17], [16] is briefly introduced as a system model for modeling complex mobile systems. An event set (a set of events) is used as an essential element of dependency structure model. Here the notations 2^X and $|X|$ are used to respectively denote the power set and the size of an event set X. Next comes the definition of a dependency structure.

4.1. Definition

Definition 4.1. A dependency structure (DS) is a tuple $<\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ with

- ξ, a finite set of events,
- $\mathbb{I} \subseteq 2^{\xi}$, the set of initially available event sets,
- $\mathbb{T} \subseteq 2^{\xi}\backslash\{\phi\} \times 2^{\xi}\backslash\{\phi\}$, the *transformation* relation,
- $\mathbb{S} \subseteq 2^{\xi}$, the *synchronism* relation such that $\forall X \in \mathbb{S}$: $|X| > 1$,
- $\mathbb{C} \subseteq 2^{\xi}$, the *choice* relation such that $\forall X \in \mathbb{C}$: $|X| > 1$,
- $\mathbb{P} \subseteq 2^{\xi}\backslash\{\phi\} \times 2^{\xi}\backslash\{\phi\}$, the *priority* relation,
- $\mathbb{W}: \xi \rightarrow \{1,2,3,...\}$, the capacity function, and
- $\mathbb{F} \subseteq 2^{\xi}$, the set of finally available event sets.

A dependency structure model can be graphically expressed by using a dependency structure diagram. For example, the dependency structure of the above running example is illustrated in Figure 2.

4.2. Execution Semantics

In a dependency structure, a system only runs according to its transformation dependencies, and the execution of each transformation dependency may lead to the change in state. Thus a state of a system is defined as follows:

Definition 4.2.1. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure. A *state* S of DS is a tuple

$\langle \Delta, F, \Gamma \rangle$ where $\Delta \subseteq \xi$ is the set of currently available events, F is the *availability function* from Δ to the set $\mathbb{Z}^*$ of nonnegative integers and $\Gamma \subseteq \mathbb{T}$ is the set of activated transformation dependencies satisfying for all dependencies $(X,Y) \in \mathbb{T} \Rightarrow X \subseteq \Delta$. The initial state of DS is defined as $S_0 = \langle \Delta_0, F_0, \Gamma_0 \rangle$ such that $\Delta_0 = \cup_{X \in \mathbb{I}} X$, $\forall e \in \Delta_0 : F_0(e) = |\{(X,Y) \in \mathbb{T} \mid e \in X\}|$ and $\Gamma_0 = \{(X,Y) \mid X \in \mathbb{I}, (X,Y) \in \mathbb{T}\}$.

Except for transformation dependency, other constraints (synchronism, choice and priority) can only control the execution of such a system. The synchronism sets specify the control of synchronization waiting and data merge, while the choice sets express the exclusive choice control and priorities control the occurrence order of the events. Thus, the execution semantics of a dependency structure can be defined as follows.

Definition 4.2.2. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure and $S_1 = \langle \Delta_1, F_1, \Gamma_1 \rangle$, $S_2 = \langle \Delta_2, F_2, \Gamma_2 \rangle$ be two of its states.

S_1 can evolve into S_2 by executing a transformation dependency (A,B), denoted $S_1 \overset{(A,B)}{\rightarrow} S_2$, if the following conditions hold:

1 $(A,B) \in \Gamma_1$,

2 $\nexists (E,F) \in \Gamma_1 : F \multimap B$,

3 $\Delta_2 = \{e \in \Delta_1 \mid e \notin A \vee (e \in A \wedge F_1(e) - (1+x) > 0)\} \cup B$,

4 $\forall e \in \Delta_2 : F_2(e) \leq \mathbb{W}(e)$

$$\wedge F_2(e) = \begin{cases} F_1(e) - (1+x), & e \in A \backslash B \\ F_1(e), & e \in \Delta_1 \backslash (A \cup B) \\ F_1(e) - (1+x) + y, & e \in A \cap B \\ F_1(e) + y, & e \in (\Delta_1 \backslash A) \cap B \\ y, & e \in B \backslash \Delta_1 \end{cases}$$

where $y = |\{(X,Y) \in \mathbb{T} \mid X \cap B \neq \phi\}|$ and $x = |\{(A,X) \in \mathbb{T} \mid \exists e \in X, \exists e' \in B, \exists C \in \mathbb{C} : e \neq e' \wedge \{e, e'\} \subseteq C\}|$, and

5 $\Gamma_2 = (\Gamma_1 \backslash (\{(A,B)\} \cup B^{\mathbb{C}})) \cup B^{\mathbb{T}} \cup B^{\mathbb{S}}$, where $B^{\mathbb{T}} = \{(B,X) \mid (B,X) \in \mathbb{T}\}$, $B^{\mathbb{S}} = \{(X,Y) \in \mathbb{T} \mid X \in \mathbb{S}, X \in \Delta_1 \cup B, B \subseteq X, Y \subseteq \xi\}$ and $B^{\mathbb{C}} = \{(W,X) \in \mathbb{T} \mid W \subseteq \xi, \exists e \in X, \exists e' \in B, \exists C \in \mathbb{C} : e \neq e' \wedge \{e, e'\} \subseteq C\}$.

According to the definition, all possible states of a dependency structure can be computed one by one. For a more detailed explanation of dependency structure, please refer to the literatures [18], [17], [16], [11].

4.3. Definition of Properties

Some properties are defined here, which will be used in the subsequent sections to analyze the behaviors of a mobile system.

Definition 4.3.1. $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure and S_0 be the initial state of DS.

1 A state S is said to be *reachable from* S', denoted as $S' \twoheadrightarrow S$, if there exist the states $S'_1, ..., S'_{n-1}$ such that $S' \overset{d'_1}{\rightarrow} S'_1...S'_{n-1} \overset{d'_n}{\rightarrow} S$, $(d'_i \in \mathbb{T}, i \in \{1,...,n\})$. A state S is said to be *reachable in DS* if there exist the states $S_1, ..., S_{n-1}$ such that $S_0 \overset{d_1}{\rightarrow} S_1...S_{n-1} \overset{d_n}{\rightarrow} S$, $(d_i \in \mathbb{T}, i \in \{1,...,n\})$. $Sta(DS)$ denotes the set of all reachable states in DS.

2 Let $S = \langle \Delta, F, \Gamma \rangle \in$ *Sta*(*DS*). *S* is said to be *terminated* in *DS* if $\Gamma = \phi$ and $\forall$ e $\in$ Δ: *F*(e) = 0 $\wedge$ e $\in$ $\cup_{X \in \mathbb{F}} X$. *S* is said to be *dead* in *DS* if *S* is not terminated and there does not exist a state *S′* such that $S \twoheadrightarrow S'$. *DS* is said to be *weakly terminated* if and only if for all $S \in$ *Sta*(*DS*), *S* is terminated or there exists a terminated state $S^t \in$ *Sta*(*DS*) such that $S' \twoheadrightarrow S^t$. *DS* is said to be *dead* (deadlocked) if and only if $\exists S \in$ *Sta*(*DS*): *S* is dead. *DS* is said to be *deadlock-free* if and only if $\nexists$S $\in$ *Sta*(*DS*): S is dead.

3 A *trace* of *DS* is a sequence $\sigma = d_1 \ldots d_n$ ($d_i \in \mathbb{T}$, $i \in \{1,\ldots,n\}$) such that there exist the states $S_1, \ldots, S_n$ such that $S_0 \xrightarrow{d_1} S_1 \ldots S_{n-1} \xrightarrow{d_n} S_n$. *n* is the length of σ. $\ulcorner\sigma$ and $\overline{\sigma}$ are respectively defined as the set of all dependencies in the trace σ and the set of event sets in σ, that is, $\ulcorner\sigma = \{d_1,\ldots, d_n\}$ and $\overline{\sigma} = \{A \subseteq \xi \mid \exists (C, D) \in \ulcorner\sigma : A = C \vee A = D\}$. *Tr*(*DS*) denotes the set of all traces of all traces of *DS*.

4 *DS* is said to be *bounded* if $\forall$ e $\in$ ξ: $\mathbb{W}$(e) is finite.

4.4. Modeling Mobile Systems

Based on the dependency structure model introduced above, the (mobility) behavior of a complex mobile system can be easily modeled, because in such a system, every two subsequent entering events of a single object in fact form a transformation dependency. Thus, the behavior of the running example system can be represented as $DS_{run} = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$, where

$\xi = \{ en^A_{c2}, en^A_{c6}, en^A_{c10}, en^A_{c14}, en^B_{c8}, en^B_{c7}, en^B_{c6}, en^B_{c5}, en^C_{c15}, en^C_{c11}, en^C_{c7}, en^C_{c3}, en^D_{c9}, en^D_{c10}, en^D_{c11}, en^D_{c12}\}$,

$\mathbb{I} = \{\{en^A_{c2}\}, \{en^B_{c8}\}, \{en^C_{c15}\}, \{en^D_{c9}\}\}$,

$\mathbb{T} = \{(\{en^A_{c2}\},\{en^A_{c6}\}), (\{en^A_{c6}\},\{en^A_{c10}\}), (\{en^A_{c10}\},\{en^A_{c14}\}), (\{en^B_{c8}\},\{en^B_{c7}\}), (\{en^B_{c7}\},\{en^B_{c6}\}), (\{en^B_{c6}\},\{en^B_{c5}\}), (\{en^C_{c15}\},\{en^C_{c11}\}),(\{en^C_{c11}\},\{en^C_{c7}\}), (\{en^C_{c7}\},\{en^C_{c3}\}), (\{en^D_{c9}\},\{en^D_{c10}\}), (\{en^D_{c10}\},\{en^D_{c11}\}), (\{en^D_{c11}\},\{en^D_{c12}\})\}$,

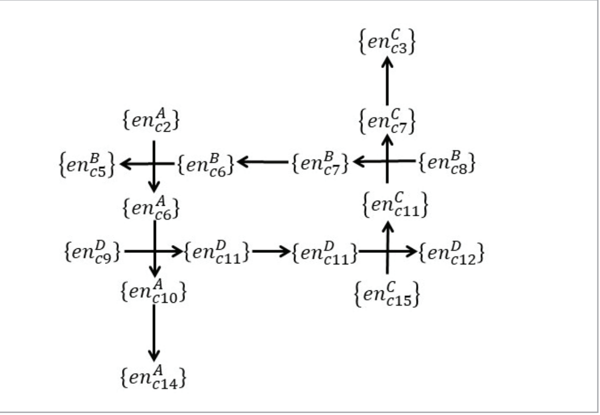
$\mathbb{S} = \phi$, $\mathbb{C} = \phi$, $\mathbb{P} = \phi$, $\forall$ e $\in$ ξ, $\mathbb{W}$(e) = ∞, and

$\mathbb{F} = \{\{en^A_{c14}\}, \{en^B_{c5}\}, \{en^C_{c3}\}, \{en^D_{c12}\}\}$.

For simplicity, a mobile system *DS* that contains mobile objects $M_1, \ldots, M_m \in \mathcal{M}$ and ambients $A_1,\ldots,A_n \in \mathcal{A}$ can also be denoted by *DS* = << $M_1,\ldots, M_m, A_1,\ldots,A_n$ >>. Let [*DS*] denote the set of mobile objects and ambients in *DS* and itself-that is, [*DS*] = { $M_1,\ldots,M_m$, $A_1,\ldots,A_n$,*DS*}. The notation $DS_x \subset DS$ is used to denote that DS_x is a mobile object or ambient of *DS*.

Figure 2

The dependency structure model of the running system



5. Scheduling

In this section, the notion of a schedule is defined based on the execution process of a mobile system, and the approach for generating a schedule is discussed accordingly.

Definition 5.1. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure.

A sequence $s = X_1 \ldots X_n$ ($X_i \subseteq \xi$, $i \in \{1,\ldots,n\}$) is called a *schedule* in DS if and only if there exist the states $S = <\Delta, F, \Gamma>$, $S_1 = <\Delta_1, F_1, \Gamma_1>$, ..., $S_n = <\Delta_n, F_n, \Gamma_n>$ in *DS* such that $S \twoheadrightarrow S_1 \twoheadrightarrow \ldots \twoheadrightarrow S_n$ and $X_i \cap \Delta_i = X_i$ for all $i \in \{1,\ldots,n\}$.

$\overline{s}$ denotes the set of all event sets in the schedule $s = X_1 \ldots X_n$, that is, $\overline{s} = \{X_1,\ldots,X_n\}$.

A schedule is an ordered event set sequence, where the events in the preceding event set occur prior to those behind. The scheduler of a system is in fact a controller or coordinator, which restricts the behavior of such a system to ensure that all given scheduling requirements are met [1].

For example, in Figure 1 a schedule $s_1 = \{en^B_{c6}\}\{en^B_{c5}\}\{en^A_{c6}\}$ can guarantee that the vehicles *A* and *B* do not collide in the cell c6. Similarly, a schedule $s_2 = \{en^A_{c10}\}\{en^A_{c14}\}\{en^D_{c10}\}$ can prevent the two vehicles *A* and *D* from any possible collision in the cell c10.

Theorem 5.1. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure and let *SX* be the set of all the schedule in *DS*. For all $s = X_1 \ldots X_n \in SX$, there exists a trace $\sigma = (A_1,B_1)(A_2,B_2) \ldots (A_m,B_m) \in Tr(DS)$ such that

$\forall i, j \in \{1,\dots,n\}, \exists i', j' \in \{1,\dots,m\}: i < j \Rightarrow i' < j' \wedge (X_i = B_{i'} \wedge X_j = B_{j'})$.

Proof. According to Definition 5.1 and Definition 4.3.1, every schedule in *DS* is a part of some trace in *DS*. Therefore, the result holds obviously.

The theorem states that any schedule is necessarily a part of a trace in the system. For instance, the schedule $s_1 = \{en^B_{c6}\}\{en^B_{c5}\}\{en^A_{c6}\}$ corresponds to the trace s = $(\{en^B_{c8}\},\{en^B_{c7}\})(\{en^B_{c7}\},\{en^B_{c6}\})(\{en^B_{c6}\},\{en^B_{c5}\})(\{en^A_{c2}\},\{en^A_{c6}\})$, which guarantees that the two vehicles *A* and *B* are collision-free in the running example system.

Theorem 5.2. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure. Given a sequence $s = X_1 \dots X_n$ ($X_i \subseteq \xi, i \in \{1,\dots,n\}$), if there exists a trace $\sigma = (A_1,B_1)(A_2,B_2)\dots(A_m,B_m) \in Tr(DS)$ such that $\forall i, j \in \{1,\dots,n\}, \exists i', j' \in \{1,\dots,m\}: i < j \Rightarrow i' < j' \wedge (X_i = B_{i'} \wedge X_j = B_{j'})$, then *s* is a schedule in *DS*.

Proof. According to Definition 4.3.1, any trace in *DS* is actually a path that the modeled system can execute along. Thus we can schedule the execution process of the system. By Definition 5.1, the sequence of event sets in the trace forms a schedule in *DS*.

The theorem shows that by computing the traces of the modeled system, we can generate proper schedules according to some given constraints and check whether a given event set sequence is a feasible (sometimes called schedulable [37]) schedule. For example, if we require that the running example system should run along the trace s = $(\{en^B_{c8}\}, \{en^B_{c7}\})(\{en^B_{c7}\},\{en^B_{c6}\})(\{en^C_{c15}\},\{en^C_{c11}\})(\{en^C_{c11}\},\{en^C_{c7}\})$, then by Definition 5.1, the sequence $\{en^B_{c8}\}\{en^B_{c7}\}\{en^B_{c6}\}\{en^C_{c11}\}\{en^C_{c7}\}$ forms a feasible schedule that ensures that the two vehicles *B* and *C* do not enter the same cell c7 at the same time. On the contrary, as for the sequence $\{en^B_{c7}\}\{en^B_{c5}\}\{en^B_{c6}\}$, since there does not exist a trace that contains the sequence in the running example system, by Definition 5.1, such a sequence cannot become a *feasible* schedule.

For convenience, given a dependency structure *DS* and a set of schedules *SX*, the restriction of *DS* to the schedules in *SX* denoted by $DS|_{SX}$. That is to say, $DS|_{SX}$ means the behavior of the system modeled by *DS* is restricted to the schedules in *SX* or the dependency structure *DS* runs under the control of the schedules in *SX*.

Proposition 5.2. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure and let *SX* be the set of all the schedules in *DS*. Then

1 $Sta(DS|_{SX}) \subseteq Sta(DS)$, and
2 $Tr(DS|_{\Sigma\Xi}) \subseteq Tr(DS)$.

Proof. This result obviously holds.

The proposition shows that the states and traces of scheduled system are part of those of the original system, respectively.

6. Decomposition and Composition of Scheduling

In this section, the decomposition and composition of scheduling are further discussed.

Definition 6.1. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure and let $t = X_1 \dots X_n$, $t' = Y_1 \dots Y_m$ be two schedules of *DS*.

1 A schedule t' is called a *sub-schedule* of *t*, denoted by $t' \lessdot t$, if and only if there exists a monotonically increasing successive sequence of integers $r_1, \dots, r_m$ such that $Y_i = X_{r_i}$ for all $i \in \{1, \dots, m\}$.
2 t' is composable with *t* if and only if there exist the states $S_1 = <\Delta_1, F_1, \Gamma_1>$, $S_2 = <\Delta_2, F_2, \Gamma_2> \in Sta(DS)$ such that $S_1 \twoheadrightarrow S_2$ and $Y_m \subseteq \Delta_1 \wedge X_1 \subseteq \Delta_2$. The composition of t' with *t*, denoted by $t't$, is the sequence $Y_1 \dots Y_m X_1 \dots X_n$.

A *sub-schedule* of a schedule is in fact an ordered part extracted from the original schedule. A composition of one schedule and another means that the occurrences of the events in the last event set of the former schedule should be prior to those in the first event set of the latter during the system's execution process.

Theorem 6.1. Let *DS* be a dependency structure.

1 If *s* is a schedule in *DS*, then for all $s' \lessdot s$, s' is a schedule in *DS*.
2 If *s*, *s'* are two schedules in *DS* and *s'* is composable with *s*, then *s's* is a schedule in *DS*.

Proof.

1 By Definition 5.1, the sequence $s = X_1 \dots X_n$ is a schedule means that there exist the states $S = <\Delta, F, \Gamma>$, $S_1 = <\Delta_1, F_1, \Gamma_1>, \dots, S_n = <\Delta_n, F_n, \Gamma_n>$ in *DS* such that $S \twoheadrightarrow S_1 \twoheadrightarrow \dots \twoheadrightarrow S_n$ and $X_i \cap \Delta_i = X_i$ for all $i \in \{1,\dots,n\}$. For all $s' \lessdot s$, assume $s' = Y_1 \dots Y_m$. By Definition 6.1, since *s'* is a monotonically increasing sequence of event sets in the sequence *s*, there exist the states S' $= <\Delta', F', \Gamma'>$, $S_1' = <\Delta_1', F_1', \Gamma_1'>, \dots, S_m' = <\Delta_m', F_m',$

$\Gamma_m'>$ in DS such that $S' \twoheadrightarrow S_1' \twoheadrightarrow ... \twoheadrightarrow S_m'$ and $Y_i \cap \Delta_i' = Y_i$ for all $i \in \{1,...,m\}$. Therefore, by Definition 5.1, s' is a schedule in DS.

2 The proof is similar to (1).

Theorem 6.1(1) states that a schedule can be decomposed into multiple sub-schedules while Theorem 6.1(2) shows that the composition of scheduling can be preserved under certain conditions.

7. Isolation Control and Collision Deadlock

In this section, the notion of isolation in a mobile system is firstly introduce on the basis of the dependency structure model, then the undesirable *collision deadlock* states are explored in details.

Definition 7.1. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure. A sequence $o = A_1...A_n$ ($A_i \in 2^{\xi}$, $i \in \{1,...,n\}$) is called an *occurrence sequence* in DS if and only if there exist the states $S, S_1, ..., S_n \in Sta(DS)$ and the event sets $B_1, ..., B_n \in 2^{\xi}$ such that $S \overset{(B_1,A_1)}{\rightarrow} S_1 ... S_{n-1} \overset{(B_n,A_n)}{\rightarrow} S_n$. $OcSq(DS)$ denotes the set of all the occurrence sequences in DS.

The restriction of $\mathcal{A}$ to the occurrence sequence $o = B_1...B_n$ is defined as $\mathcal{A}\uparrow_O = \{ A \in \mathcal{A} \mid \exists B \in \{ B_1, ..., B_n \}, \exists M \in \mathcal{M}: en^M_A \in B \}$.

When objects move about inside a typical mobile system, the successive occurrences of movement events (i.e., *entering* and *exiting* an ambient) naturally form an *occurrence sequence*. Such a sequence may involve the movement process of several mobile objects, and relate to multiple ambients that have been passed through. Here we use $\mathcal{A}\uparrow_O$ to refer to the set that contains all the ambients which are involved in the occurrence sequence o.

Proposition 7.1. Let DS be a dependency structure. Then $\forall\ o = A_1...A_n \in OcSq(DS)$, there exists a trace $\sigma = (B_1,B_2)(B_2,B_3)...(B_{m-1},B_m) \in Tr(DS)$ such that $\exists i \in \{1,...,m\}, \forall j \in \{2,...,n\}$: $(A_1 = B_i) \wedge (A_j = B_{i+j-1})$.

Proof. This result obviously holds.

Proposition 7.1 states that all occurrence sequences in a dependency structure can be obtained if we have all the traces of it computed in advance.

Definition 7.2. Let DS be a dependency structure, $A \in \mathcal{A}$, $M_1, M_2 \in \mathcal{M}$ and $A, M_1, M_2 \subset DS$.

M_1 is said to be isolated from M_2 for A in DS, denoted by $M_1 \circ_A M_2$, iff either $\forall\ o \in OcSq(DS), A \notin \mathcal{A}\uparrow_O$, or $\forall\ o = B_1...B_n \in OcSq(DS)$, $(\exists\ X \in \mathcal{A}, \nexists \{en^{M_2}_A\} \in \{B_1,...,B_n\}: B_1 = \{en^{M_1}_A\} \wedge B_1 = \{en^{M_1}_X\}) \vee (\exists\ Y \in \mathcal{A}, \nexists \{en^{M_1}_A\} \in \{B_1,...,B_n\}: B_1 = \{en^{M_2}_A\} \wedge B_1 = \{en^{M_2}_Y\})$.

This definition in fact shows that if M_1 is isolated from M_2 for the ambient A, one of the following three cases must have occurred: (1) neither M_1 nor M_2 enter A, (2) only one of the two mobile objects, that is, either M_1 or M_2 enters A, and (3) both M_1 and M_2 enter A, but one will not enter until the other exits A. In other words, M_1 and M_2 are strictly forbidden to appear at the same ambient simultaneously, and one ambient can only accommodate one mobile object at a time [18].

Obviously, in the real world, mobile objects are not born to be isolated. Hence extra scheduling control must be imposed so as to guarantee such a mobile system can meet the isolation requirements and run safely.

Theorem 7.1. Let DS be a dependency structure, $A \in \mathcal{A}$, $M_1, M_2 \in \mathcal{M}$ and $A, M_1, M_2 \subset DS$. Let SX be a set of schedules.

If $\forall S = <\Delta, F, \Gamma> \in Sta(DS|_{SX})$: $\{en^{M_1}_A, en^{M_2}_A\} \nsubseteq \Delta$, then M_1 is isolated from M_2 for A in $DS|_{SX}$.

Proof. (Proof by Contradiction) Assume M_1 is not isolated from M_2 for A in $DS|_{SX}$. By Definition 7.1 and Definition 7.2, M_1 and M_2 can enter the ambient at the same time. By Definition 4.2.1 and Definition 4.2.2, there exists a state $S = <\Delta, F, \Gamma> \in Sta(DS|_{SX})$ such that $\{en^{M_1}_A, en^{M_2}_A\} \subseteq \Delta$. This contradicts the condition $\forall S = <\Delta, F, \Gamma> \in Sta(DS|_{SX})$: $\{en^{M_1}_A, en^{M_2}_A\} \nsubseteq \Delta$.

The theorem states that, by checking the states of the target dependency structure that runs under the restriction of certain schedules, it can be decided whether two mobile objects are isolated or not for a given ambient.

Theorem 7.2. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure and $S_0 = <\Delta_0, F_0, \Gamma_0>$ be the initial state of DS.

If there exist a sequence $s = A_1...A_n$ ($A_i \in 2^{\xi}$, $i \in \{1,...,n\}$), the event sets $B_1,...,B_n \in 2^{\xi}$ and the states $S_1 = <\Delta_1, F_1, \Gamma_1>, ..., S_n = <\Delta_n, F_n, \Gamma_n> \in Sta(DS)$ such that $S_0 \overset{(B_1,A_1)}{\rightarrow} S_1 ... S_{n-1} \overset{(B_n,A_n)}{\rightarrow} S_n$, and S_n is terminated, then

1 the sequence s is a schedule in DS,

2 $DS|_{\{s\}}$ is weakly terminated, and

3 $\forall M, N \in \mathcal{M}, \forall X \in \mathcal{A}$: $\forall i \in \{1,...,n\}$, $\{en^M_X, en^N_X\} \not\subseteq \Delta i$ => M, N are isolated from each other for X in $DS|_{\{s\}}$.

Proof

1 By Definition 5.1, this result is straightforward.

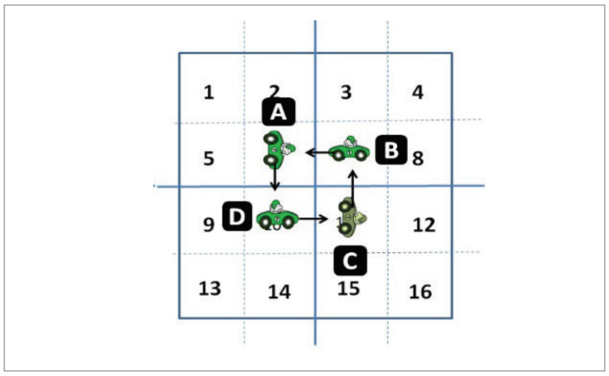
2 By Definition 4.3.1, the sequence s is actually a trace. Moreover, $DS|_{\{s\}}$ is the dependency structure that only runs along the trace s, that is, Sta($DS|_{\{s\}}$) $=\{S_0, S_1,..., S_n\}$. Since S_n is terminated, by Definition 4.3.1, $DS|_{\{s\}}$ is weakly terminated.

3 By Theorem 7.1, this result holds.

Theorem 7.2 in fact presents a schedule generating approach, which can generate scheduling policies to meet the requirements of isolation in a mobile system. For example, in the running example system, there exists the schedule $s = \{en^A_{c2}\}\{en^A_{c6}\}\{en^A_{c10}\}\{en^A_{c14}\}\{en^C_{c15}\}\{en^C_{c11}\}\{en^C_{c7}\}\{en^C_{c3}\}\{en^B_{c8}\}\{en^B_{c7}\}\{en^B_{c6}\}\{en^B_{c5}\}\{en^D_{c9}\}\{en^D_{c10}\}\{en^D_{c11}\}\{en^D_{c12}\}$ which can ensure the four vehicles A, B, C and D safely cross the road interaction without collision.

Figure 3

A collision deadlock scenario (from the literature [7])



Nevertheless, when several mobile objects are required to be isolated from each other for certain ambients in a mobile system, the risk of deadlock may possibly occur. For instance, in the running example, when vehicles A, B, C and D have respectively occupied the cells c6, c7, c11 and c10 (see Figure 3) [7], none of these four vehicles are able to move after that, otherwise collision will happen inevitably. Such a deadlock is not the kind of deadlock (deadlocked state) in common sense (see Definition 4.3.1), but is caused by collision avoidance. Therefore, such a deadlock is called a *collision deadlock.*

Definition 7.3. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure. A state $S = <\Delta, F, \Gamma> \in Sta(DS)$ is said to be *collision-deadlocked* if and only if

1 S is not terminated,

2 $\forall M_1, M_2 \in \mathcal{M}, A \in \mathcal{A}: \{en^{M_1}_A, en^{M_2}_A\} \not\subseteq \Delta$, and

3 $\forall S' = <\Delta', F', \Gamma'> \in Sta(DS)$: $S \rightarrow S'$ => $\exists M, N \in \mathcal{M}$, $\exists B \in \mathcal{A}$: $\{en^M_B, en^N_B\} \subseteq \Delta' \wedge (en^M_B \in \Delta \vee en^N_B \in \Delta)$.

A *collision-deadlocked* state indicates that the system can further evolve into other states (i.e., not terminated) and the mobile objects in such a system do not collide with each other in the current state. However, once one of these objects proceeds to move, it will inevitably collide with one of the other in an ambient, which are not the results we expected. Obviously, a complex mobile system may have a big chance of reaching such a collision-deadlocked state if we do not schedule the moving of these objects in advance.

Theorem 7.3. Let $DS = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ be a dependency structure and SX be the set of schedules in DS. Then $\forall M, N \in \mathcal{M}, \forall A \in \mathcal{A}$: M, N are isolated from each other for A in $DS|_{SX}$ => $\forall S \in Sta(DS|_{SX})$: S is not collision-deadlocked.

Proof. (Proof by Contradiction) Assume $\exists S \in Sta(DS|_{SX})$: S is collision-deadlocked. When the system reaches the state S, by Definition 7.3, there exists a mobile object such that if the mobile object continues to move, the system will enter a collision state. By Definition 7.2, $\exists M, N \in \mathcal{M}$, $\exists A \in \mathcal{A}$: M, N are not isolated from each other for A in $DS|_{SX}$. This contradicts the condition.

The theorem states that if a mobile system can ensure the isolation of its mobile objects, it should not possibly enter a collision-deadlocked state. The idea will be used in the next section for developing an algorithm for automatically generating schedules.

8. Algorithm

In this section, based on the theoretical analysis above, we have developed an algorithm (see Algorithm 1) for automatically generating a schedule, which can help to ensure the safety isolation of mobile objects in a mobile system. The input of the algorithm is the mobile system itself, including the dependency structure model, its initial state S_0, m mobile objects and n ambients.

Algorithm 1: Scheduling sequence generation

```
Input:
    $DS = <\xi, I, T, S, C, P, W, F>$ a mobile system,
    $S_0 = <\Delta_0, F_0, \Gamma_0>$   the initial state of $DS$,
    $MO = \{M_1, ..., M_m\}$   $m$ mobile objects,
    $IA = \{A_1, ..., A_n\}$   $n$ ambients
Output
    A schedule seq or non-existent.
1  let st be a stack;
2  let seq be a list of event sets;
3  let flag be a boolean variable;
4  seq: = empty; st: = empty; flag = false; $i \leftarrow 0$;
5  if $<\Delta_0, F_0, \Gamma_0>$ is collision-deadlocked then
6      return non-existent;
7  end
8  for each $(A,B) \in \Gamma_0$ do
9      st.push($<\Delta_0, F_0, \Gamma_0, A>$);
10 end
11 while st is not empty do
12     $<\Delta, F, \Gamma, X>$ = st.pop();
13     seq[$i$]: = $X$ ;
14     $i = i + 1$;
15     if $<\Delta, F, \Gamma>$ is terminated then
16         Exit;
17     else
18         if $<\Delta, F, \Gamma>$ is collision-deadlocked then
19             $i = i - 1$; Continue;
20         else
21             flag = false;
22             for each $(X, B) \in \Gamma$ do
23                 if $<\Delta, F, \Gamma> \xrightarrow{(X,B)} <\Delta', F', \Gamma'>$ and
                   $\nexists M, N \in MO, \nexists A \in IA$: $en_A^M, en_A^N \in \Delta'$
                   then
24                     for each $(A', B') \in \Gamma'$ do
25                         st.push($<\Delta', F', \Gamma', A'>$);
26                     end
27                     flag = true;
28                 end
29             end
30             if flag = = false then
31                 $i = i - 1$;
32             end
33         end
34     end
35 end
36 if $<\Delta, F, \Gamma>$ is not terminated then
37     return non-existent;
38 else
39     return seq;
40 end
```

The output either is the generated schedule **seq**, or concludes that a schedule is *non-existent* because the system reaches a collision-deadlocked state or abnormally terminates.

The computing process are described as follows:

1. Check whether S_0 is collision-deadlocked. If true, it means that the system can not evolve in its initial state, thus a schedule is non-existent and the algorithm exits (lines 5-7). Otherwise, push S_0 and the pre-dependency set of each activated transformation dependency onto the stack **st** one by one (lines 8-10).
2. Enter a while loop as long as the stack **st** is not empty (line 11).
3. Pop an element off the stack **st** and store the pre-dependency set (X) as an element of a schedule (lines 12-14).
4. Check whether the current state is terminated. If that is the case, then a schedule is completely generated and the algorithm exits the while loop (lines 15-16).
5. Check whether the current state is collision- deadlocked. If true, then the pre-dependency set (X) cannot be contained in the generated schedule. The algorithm continues to start the next iteration (lines 18-19).
6. Otherwise, compute the new states according to the current state and the currently activated transformation dependencies. The new states and their corresponding pre-dependency sets are pushed onto the stack **st** if the states meet the requirements of isolation (lines 22-29). If there does not exist a new state that is pushed onto the stack, the corresponding pre-dependency set is not an element of the schedule (lines 30-32).
7. When the while loop ends, check whether the current state is not terminated. If true, it means that the algorithm does not compute a feasible schedule (line 37); otherwise, a schedule is generated (line 39).

The complexity of the algorithm above is $2^{|\mathbb{T}|}$ where $\mathbb{T}$ is the transformation relation of the dependency structure DS. Except for this, the complexities of the other parts are all linear time. Fortunately, given an intersection or roundabout scenarios of intelligent transportation systems, the computation usually involves a limited number of events. Thus there exists ample time for modern computing devices to generate a feasible schedule (see Section 9.2).

To accurately evaluate the effectiveness and scalability of the algorithm, we have implemented it and incorporated it into DSTool (see Figure 7)–a prototype tool we developed to support the constructing and reasoning of dependency structure models. The latest version of DSTool is developed using JavaScript, which makes it capable of running directly on most of the major browsers without extra installation.

9. Experiment

In this section, a case study of a fully-automated container terminals (ACT) is presented, and a series of simulation experiments have been conducted by employing the scheduling policies generation algorithm to evaluate and demonstrate the effectiveness and scalability of our event-based scheduling approach.

9.1. Fully-Automated Container Terminal

Here we present a typical ACT like QQCT (Qingdao Qianwan container terminal, see Figure 4), whose layout is graphically presented in Figure 5. To meet the challenges of booming marine container transportation, the terminal employs automation equipments, such as quay cranes (QCs), automatic guided vehicles (AGVs) and automated stacking cranes (ASCs), to support the loading and discharging of containers. As major means of horizontal transport between the QCs and ASCs, AGVs move around to deliver containers. It is obviously that these vehicles should be effectively isolated from each other so as to ensure their safety and efficiency.

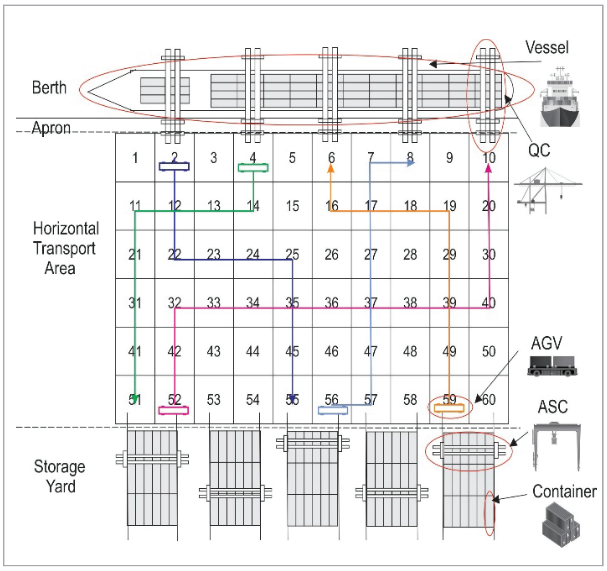
In the same way as the running example, the whole terminal area is divided into multiple grid cells, each of which is regarded as an ambient and marked with a unique identifier. Suppose there exist five AGVs *A*, *B*, *C*, *D* and *E* which have been assigned different delivery jobs (see Figure 5). To carry out the discharging operations, the vehicle *A* receives an inbound container from the QC in c2 and prepares to bring it to the delivery location in c55 where the container is planned to be stacked. So does the vehicle *B*, preparing to deliver its consignment from c4 to c51. Meanwhile, the loading on board is concurrently in progress, which requires the vehicles *C*, *D* and *E* to deliver their loadings from ASCs in the cells c52, c56, c59 to the QCs in the cells c10, c8, c6, respectively. According to certain traffic rules and protocols set by the

Figure 4

Qingdao Qianwan container terminal (QQCT) in China

Figure 5

The layout of a typical ACT



port authority, once a job is assigned to a particular AGV, the path from the loading point to the unloading point is predetermined solely by using the intermediate nodes. For example, as shown in Figure 5, *A* is required to take the path c2 → c12 → c22 → c23 → c24 → c25 → c35 → c45 → c55, and *B* does the path c4 → c14 → c13 → c12 → c11 → c21 → c31 → c41 → c51. Similarly, the vehicles *C*, *D* and *E* are demanded to move along the paths c52 → c42 → c32 →c33 → c34 → c35 → c36 → c37 → c38 → c39 → c40 → c30 → c20 → c10, c56 → c57 → c47 → c37 → c27 → c17 → c7 → c8 and c59 → c49 → c39 → c29 → c19 → c18 → c17 → c16 → c6, respectively. Obviously, there exist some intersections (e.g. c12, c17, c35, c37, c39) in the predetermined paths of different vehicles. In order to prevent possible collision, congestion and deadlocks in the yard, AGVs need to be isolated from each other for the whole terminal area, especially for those intersection cells. Using dependency structures, the behavior of such a mobile system can be modeled as $DS_{ACT} = <\xi, \mathbb{I}, \mathbb{T}, \mathbb{S}, \mathbb{C}, \mathbb{P}, \mathbb{W}, \mathbb{F}>$ where

$\xi = \{ en^{A}_{c2}, en^{A}_{c12}, en^{A}_{c22}, en^{A}_{c23}, en^{A}_{c24}, en^{A}_{c25}, en^{A}_{c35}, en^{A}_{c45}, en^{A}_{c55}, en^{B}_{c4}, en^{B}_{c14}, en^{B}_{c13}, en^{B}_{c12}, en^{B}_{c11}, en^{B}_{c21}, en^{B}_{c31}, en^{B}_{c41}, en^{B}_{c51}, en^{C}_{c52}, en^{C}_{c42}, en^{C}_{c32}, en^{C}_{c33}, en^{C}_{c34}, en^{C}_{c35}, en^{C}_{c36}, en^{C}_{c37}, en^{C}_{c38}, en^{C}_{c39}, en^{C}_{c40}, en^{C}_{c30}, en^{C}_{c20}, en^{C}_{c10}, en^{D}_{c56}, en^{D}_{c57}, en^{D}_{c47}, en^{D}_{c37}, en^{D}_{c27}, en^{D}_{c17}, en^{D}_{c7}, en^{D}_{c8}, en^{E}_{c59}, en^{E}_{c49}, en^{E}_{c39}, en^{E}_{c29}, en^{E}_{c19}, en^{E}_{c18}, en^{E}_{c17}, en^{E}_{c16}, en^{E}_{c6}\}$,

$\mathbb{I} = \{\{en^{A}_{c2}\}, \{en^{B}_{c4}\}, \{en^{C}_{c52}\}, \{en^{D}_{c52}\}, \{en^{E}_{c59}\}\}$,

$\mathbb{T} = \{(\{en^{A}_{c2}\},\{en^{A}_{c12}\}), (\{en^{A}_{c12}\},\{en^{A}_{c22}\}), (\{en^{A}_{c22}\},\{en^{A}_{c23}\}), (\{en^{A}_{c23}\},\{en^{A}_{c24}\}), (\{en^{A}_{c24}\},\{en^{A}_{c25}\}), (\{en^{A}_{c25}\},\{en^{A}_{c35}\}), (\{en^{A}_{c35}\}, \{en^{A}_{c45}\}), (\{en^{A}_{c45}\},\{en^{A}_{c55}\}), (\{en^{B}_{c4}\},\{en^{B}_{c14}\}), (\{en^{B}_{c14}\},\{en^{B}_{c13}\}), (\{en^{B}_{c13}\},\{en^{B}_{c12}\}), (\{en^{B}_{c12}\},\{en^{B}_{c11}\}), (\{en^{B}_{c11}\},\{en^{c21}_{B}\}), (\{en^{B}_{c21}\}, \{en^{B}_{c31}\}), (\{en^{B}_{c31}\},\{en^{B}_{c41}\}), (\{en^{B}_{c41}\},\{en^{B}_{c51}\}), (\{en^{C}_{c52}\},\{en^{C}_{c42}\}), (\{en^{C}_{c42}\},\{en^{C}_{c32}\}), (\{en^{C}_{c32}\},\{en^{C}_{c33}\}), (\{en^{C}_{c33}\},\{en^{C}_{c34}\}), (\{en^{C}_{c34}\}, \{en^{C}_{c35}\}), (\{en^{C}_{c35}\},\{ en^{C}_{c36}\}), (\{en^{C}_{c36}\},\{en^{C}_{c37}\}), (\{en^{C}_{c37}\},\{en^{C}_{c38}\}), (\{en^{C}_{c38}\},\{en^{C}_{c39}\}), (\{en^{C}_{c39}\},\{en^{C}_{c40}\}), (\{en^{C}_{c40}\},\{en^{C}_{c30}\}), (\{en^{C}_{c30}\}, \{en^{C}_{c20}\}), (\{en^{C}_{c20}\},\{en^{C}_{c10}\}), (\{en^{D}_{c56}\},\{en^{D}_{c57}\}), (\{en^{D}_{c57}\},\{en^{D}_{c47}\}), (\{en^{D}_{c47}\},\{en^{D}_{c37}\}), (\{en^{D}_{c37}\},\{en^{D}_{c27}\}), (\{en^{D}_{c27}\},\{en^{D}_{c17}\}), (\{en^{D}_{c17}\}, \{en^{D}_{c7}\}), (\{en^{D}_{c7}\},\{en^{D}_{c8}\}), (\{en^{E}_{c59}\},\{ en^{E}_{c49}\}), (\{en^{E}_{c49}\},\{en^{E}_{c39}\}), (\{en^{E}_{c39}\},\{en^{E}_{c29}\}), (\{en^{E}_{c29}\},\{en^{E}_{c19}\}), (\{en^{E}_{c19}\},\{en^{E}_{c18}\}), (\{en^{E}_{c18}\}, \{en^{E}_{c17}\}), (\{en^{E}_{c17}\},\{en^{E}_{c16}\}), (\{en^{E}_{c16}\},\{en^{E}_{c6}\})\}$,

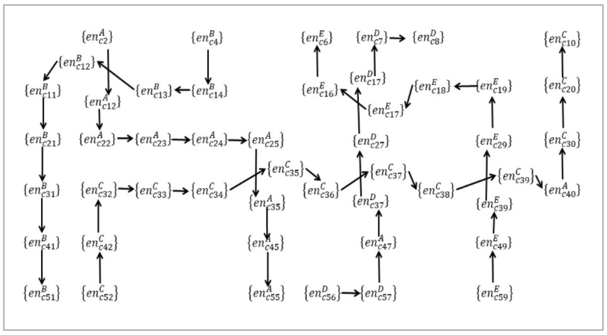
$\mathbb{S} = \phi, \mathbb{C} = \phi, \mathbb{P} = \phi, \forall e \in \xi, \mathbb{W}(e) = \infty$, and

$\mathbb{F} = \{\{en^{A}_{c55}\}, \{en^{B}_{c51}\}, \{en^{C}_{c10}\}, \{en^{D}_{c8}\}, \{en^{E}_{c6}\}\}$.

Using the algorithm above (see Algorithm 1), a scheduling sequence can be conveniently obtained which can guarantee the isolation of AGVs in this terminal, like the one below:

Figure 6

The dependency structure model of part of a fully-automated container terminal



$s = \{en^{A}_{c2}\}\{en^{A}_{c12}\}\{en^{A}_{c22}\}\{en^{B}_{c4}\}\{en^{B}_{c14}\}\{en^{B}_{c13}\}\{en^{B}_{c12}\}\{en^{B}_{c11}\}\{en^{B}_{c21}\} \{en^{B}_{c31}\}\{en^{B}_{c41}\}\{en^{B}_{c51}\}\{en^{A}_{c23}\}\{en^{A}_{c24}\}\{en^{A}_{c25}\}\{en^{A}_{c35}\}\{en^{A}_{c45}\}\{en^{A}_{c55}\} \{en^{C}_{c52}\}\{en^{C}_{c42}\}\{en^{C}_{c32}\}\{en^{C}_{c33}\}\{en^{C}_{c34}\}\{en^{C}_{c35}\}\{en^{C}_{c36}\}\{en^{C}_{c37}\}\{en^{C}_{c38}\} \{en^{D}_{c56}\}\{en^{D}_{c57}\}\{en^{D}_{c47}\}\{en^{D}_{c37}\}\{en^{D}_{c27}\}\{en^{D}_{c17}\}\{en^{D}_{c7}\}\{en^{D}_{c8}\}\{en^{C}_{c39}\} \{en^{C}_{c40}\}\{en^{E}_{c59}\}\{en^{E}_{c49}\}\{en^{E}_{c39}\}\{en^{E}_{c29}\}\{en^{E}_{c19}\}\{en^{E}_{c18}\}\{en^{E}_{c17}\}\{en^{E}_{c16}\} \{en^{E}_{c69}\}\{en^{C}_{c30}\}\{en^{C}_{c20}\}\{en^{C}_{c10}\}$,

Under the control of the generated schedule, the vehicle *B* will arrive at its destination first, then will the vehicle *A*, after that are the vehicles *D*, *E* and *C* in sequence.

In practical application, besides isolation, the scheduling of AGVs must take into account many other constraints and conditions, such as to maximize the QCs' productivity by reducing their delays, to minimize the CO2 emissions by reducing the empty-travel distances of the AGVs, etc. Based on these concerns, we can further improve the algorithm to generate the most optimal schedule in a more efficient way.

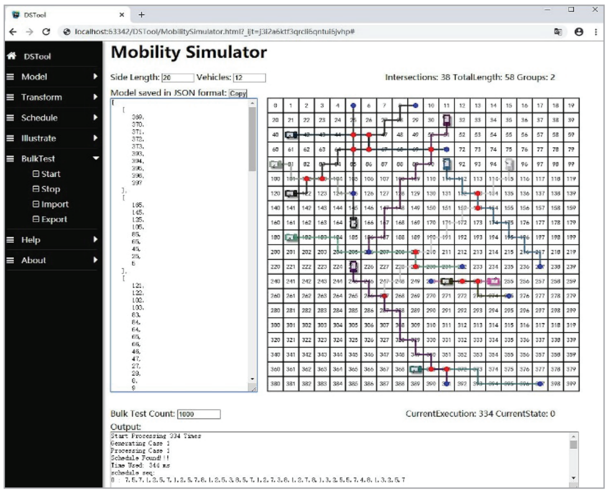
9.2. Scalability Evaluation

For performance evaluations, a total of more than 600000 simulation experiments are conducted in five batches by using the latest version (91.0.4472) of Google Chrome, on a standard windows laptop with an Intel(R) Core(TM) i7-4600U CPU and 12 GB of memory. In each batch, a square area composed of a fixed number of cells (that are 15×15,20×20,25×25,30×30 and 35×35 separately) is created to model the ACT and a variable numbers of AGVs are emulated to move in such an area (see Figure 7 for example). Every time when the number of AGVs and the upper limit of the length of their travelling paths are given, the moving paths of all AGVs are randomly determined, thus all of the necessarily isolated cells (i.e., intersection cells) are determined. Then the scheduling generating algorithm (Algorithm 1) are carried out to generate a feasible schedule which can guarantee the safety isolation of the AGVs in the whole area (or conclude that such a schedule is non-existent). Such a simulation is replicated again and again with AGVs' number ranging from 2 to 14, and the time spent in generating a schedule and the corresponding test case are recorded in CSV format each time for further detailed analysis.

The main results of the experiments are presented as follows. Table 1 lists the average time spent in generating a schedule under different number of vehicles in different size of areas, while Table 2 does the maximum time. Note that all statistics are in milliseconds. Based

Figure 7

The DS Tool



Side Length: 20 Vehicles: 12

Intersections: 38 TotalLength: 58 Groups: 2

Model saved in JSON format:

Bulk Test Count: 1000

CurrentExecution: 334 CurrentState: 0

Output:

```
Start Processing 334 Times
Generating Case 1
Processing Case 1
Schedule Found!!!
Time Used: 344 ms
schedule seq:
0 :  7,5,7,1,2,5,7,1,2,5,7,8,1,2,5,3,8,5,7,1,2,7,3,8,1,2,7,8,1,3,2,5,5,7,4,8,1,3,2,5,7
```

on the data collected, Figures 8-9 intuitively illustrate the feasibility and scalability of the approach. Figure 8 shows the average time spent in generating a schedule in different size of areas, and Figure 9 compares the average time spent with the maximum time spent in a standard 20×20 square area as a typical example. Evidently, as the number of vehicles increases, the average time spent in generating a schedule grows accordingly (see Figure 8), while the maximum time spent is further higher than the average time (see Figure 9).

This conclusion can be well explained as follows, by Figures 10-11.

Firstly, given the upper limit of the length of the paths, once the size of the area (i.e., the amount of cells) is determined, the vehicular traffic capacity in such an area is considered to be constant. The more vehicles you arrange in the area, the more intersections that

Table 1

Average time spent in generating a schedule(in milliseconds)

Vehicle Number / Size of area	2	3	4	5	6	7	8	9	10	11	12	13	14
15 × 15	6.46	13.49	17.96	20.98	22.4	24.88	35.59	91.88	271.62	1010.7	-	-	-
20 × 20	3.91	10.18	14.17	18.23	20.12	21.99	26.16	41.64	127.48	436.09	1234.3	2340.71	5645.93
25 × 25	2.94	6.43	11.85	15.55	20.87	22.83	24.50	29.52	46.1	121.39	197.49	499.83	1415.27
30 × 30	1.94	9.37	10.78	15.69	20.45	23.78	24.43	24.91	61.88	65.07	83.57	104.74	236.7
35 × 35	1.44	4.01	7.48	10.83	14.65	17.68	20.32	24.38	25.97	31.34	33.64	46.01	63.14

Note that ”-” means being omitted because of the insufficiency of the samples.

Table 2

Maximum time spent in generating a schedule (in milliseconds)

Vehicle Number / Size of area	2	3	4	5	6	7	8	9	10	11	12	13	14
15 × 15	113	332	1150	1228	1249	2513	**29837**	42909	84237	115174	-	-	-
20 × 20	44	157	193	269	318	375	1214	**10923**	52366	106521	149194	124421	239505
25 × 25	35	35	53	65	120	672	531	**2803**	15669	54328	64646	99292	285744
30 × 30	38	37	52	54	66	217	734	969	**31107**	39536	80774	62067	95376
35 × 35	29	31	36	40	39	43	88	85	**512**	3536	7508	14425	65614

Note that ”-” means being omitted because of the insufficiency of the samples and data in bold text indicates where the drastic increases occur.

Figure 8

Average time spent in generating a schedule in different sizes of areas

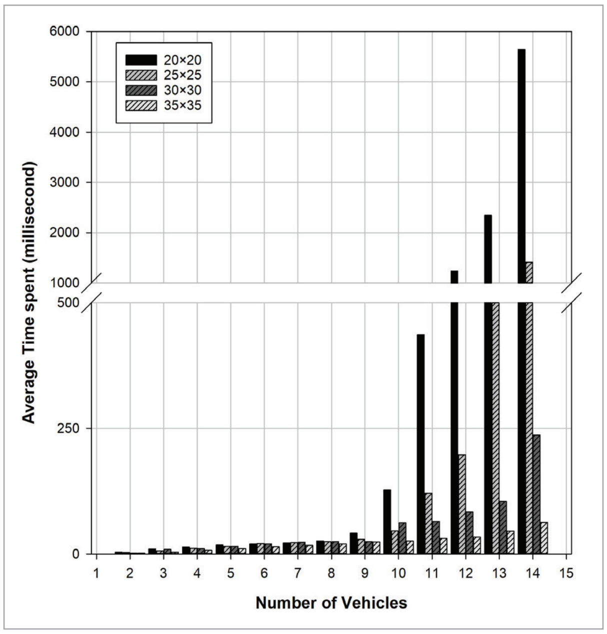
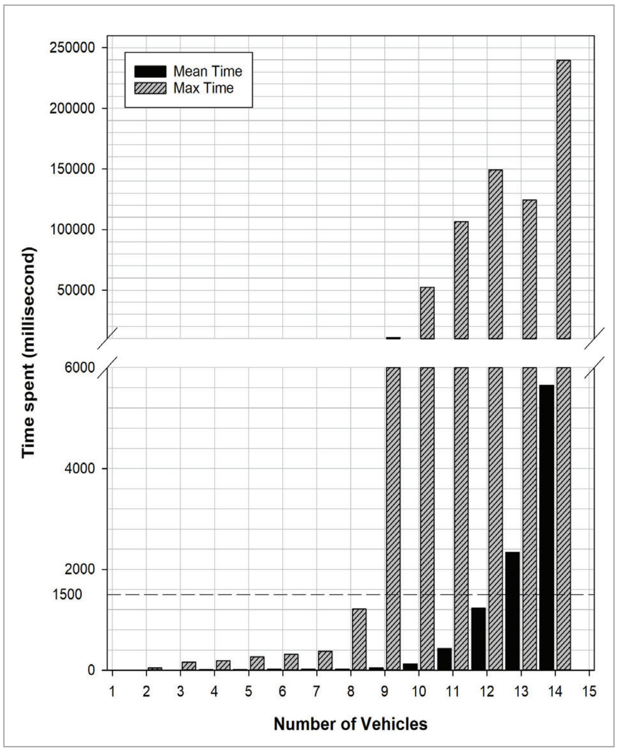


Figure 9

Average and maximum time spent in generating a schedule under different number of vehicles in a 20 × 20 square area



may appear on their paths, which makes the vehicles more likely to collide with each other. Hence it takes the algorithm more time to explore much more states to find a feasible schedule, which is confirmed by the statistics shown in Figure 10. Besides that, it is also noticeable that under the same conditions a larger area can accommodate more vehicles than smaller ones so that the schedule can be generated more quickly, as indicated in Figure 8.

Secondly, as shown in Figure 11, when the number of vehicles increases, the probability of non-existent of schedule (i.e., isolation failure caused by path conflict) grows significantly, which forces the algorithm to iterate over nearly all possible states before making a final decision, resulting in the significant increase in computing time, especially the maximum time. From the massive simulation cases we collected, it is not difficult to find out that the maximum time spent usually comes from the cases where the number of intersections is great and all paths intersect with each other sometimes may even lead to the non-existence of a schedule. However, fortunately, complex cases like these are only a small part of all generated cases so that the average time spent turns out to be much lower than the maximum time (confirmed by Figure 11).

In addition, it is also worth mentioning that a drastic increase can be observed in the maximum time spent as the number of vehicles reaches or exceeds a certain threshold. For example, when the number of vehicles reaches 9 in a 20×20 area, as shown in Figure 9. Similar changes can also be found in the results for 15×15,25×25,30×30 and 35×35 areas (see Table 2), which is supposed to be the result of some kind of vehicle saturation in the bounded area.

In order to enhance algorithm performance, several optimizing methods are applied in the actual algorithm implementation. To simplify the input, the length of the AGVs' moving paths can be reduced by only considering the intersection cells. Moreover, those paths can be further split into several independent (non-overlapping) groups so that a divide-and-conquer strategy can be employed to accelerate the schedule generation. In addition, some typical collision-deadlock states can be recognized in advance by directly using certain common patterns. Due to the high dimensionality of complex search spaces and the inherent performance limitation of JavaScript and browsers, in extreme cases (e.g., when

Figure 10

Relationship between the number of intersections in the vehicles' paths when the number of vehicles is 10 in a 20×20 square area

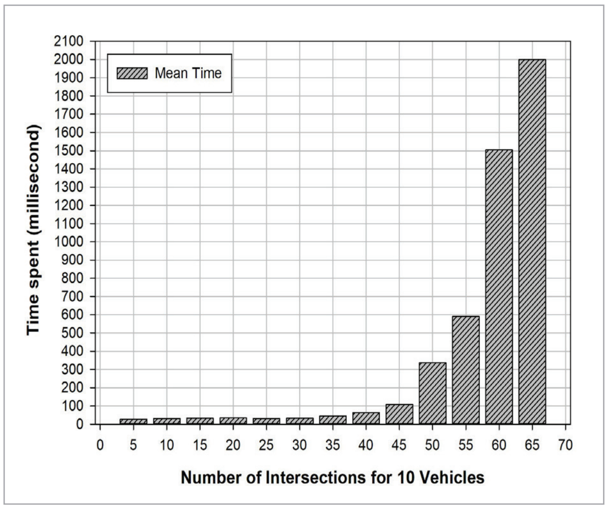
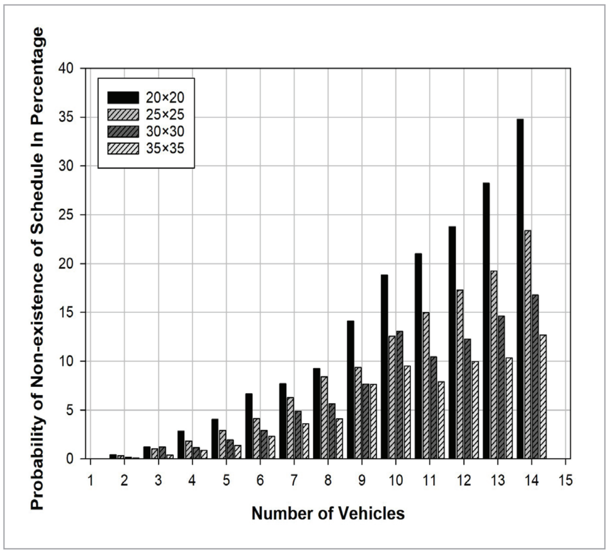


Figure 11

Probability of non-existent scheduling under different number of vehicles in different sizes of areas



the AGVs' number exceeds 11 in a 15×15 area), the simulation is still time-consuming, preventing it from being executed enough times. Consequently, this part of results are not convincing which are not presented in the illustration.

Despite all this, as shown in Figure 9, in a 20×20 area, when the number of vehicles is less than 9, the algorithm can always generate a schedule (or conclude the schedule is non-existent) in less than 1.5 seconds, which is efficient enough for most autonomous driving scenarios according to the report [13]. Because in the case of autonomous driving, a vehicle only needs to consider a very limited area around it. In that case, it only needs to consider at most 8 vehicles which are the nearest to it in all directions.

10. Conclusion

In this paper, a novel approach to analyzing and implementing the safety isolation of mobile objects in a complex mobile system is proposed, which employs a more fine-grained event-based formal model called Dependency Structure. Thus an automatic schedule generating algorithm is provided and implemented to generate the isolation scheduling policies in such a mobile system. Simulation experiments are conducted to solve the intersection isolation problems in a concrete intelligent traffic system and the result demonstrates the effectiveness and scalability of our approach.

In general, our results are mainly twofold:

1 Finer-grained event scheduling is more applicable for complex scheduling problems in complex mobile systems than traditional task scheduling.

2 The safety isolation of mobile objects in complex mobile systems can be achieved by using a schedule generated automatically by an algorithm.

Since intersection isolation (collision avoidance) is only one particular form of the concept of isolation, the work here is just the beginning. Future research will further extend and optimize the isolation control theory and scheduling generation policies of the complex concurrent mobile system, and we wish it to be finally applied for real-world intelligent transportation systems, such as the Chinese train control system [34], to ensure their safety operations.

Acknowledgement

This work is supported by National Natural Science Foundation of China (61772004), Scientific Research Foundation for Advanced Talents of Chengdu University of Information Technology (KYTZ202009), and Sichuan Major Science and Technology Special Program under Grant (2017GZDZX0002).

References

1. Altisen, K., Go¨ ßler, G., Sifakis, J. Scheduler Modeling Based on the Controller Synthesis Paradigm. Real-time Systems, 2002, 23(1-2), 55-84. https://doi.org/10.1023/A:1015346419267
2. Anderson, J. H., Bud, V., Devi, U. C. An EDF-based Restricted-migration Scheduling Algorithm for Multiprocessor Soft Real-time Systems. Real-Time Systems, 2008, 38(2), 85-131. https://doi.org/10.1007 /s11241-007-9035-0
3. Aoki, S., Rajkumar, R. A Configurable Synchronous Intersec- tion Protocol for Self-driving Vehicles. 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, (RTCSA 2017), Hsinchu, Taiwan, August 16-18, 2017, 1-11. https://doi.org/10.1109/RTCSA.2017. 8046306
4. Aoki, S., Rajkumar, R. A Merging Protocol for Self-driving Vehicles. Proceedings of the 8th International Conference on Cyber-Physical Systems, (ICCPS 2017), Pittsburgh, Pennsylvania, USA, April 18-20, 2017, 219-228. https://doi.org/ 10.1145/3055004.3055028
5. Aoki, S., Rajkumar, R. R. Dynamic Intersections and Self-driving Vehicles. Proceedings of the 9th ACM/IEEE In- ternational Conference on Cyber-Physical Systems, (ICCP- S 2018), Porto, Portugal, April 11-13, 2018, 320-330. https://doi.org/10.1109/ICCPS.2018.00038
6. Audsley, N. C., Burns, A., Wellings, A. J. Deadline Monotonic Scheduling Theory and Application. Control Engineering Practice, 1993, 1(1), 71-78. https://doi.org/10.1016/0967-0661 (93) 92105-D
7. Azimi, S. R., Bhatia, G., Rajkumar, R., Mudalige, P. STIP: Spatio-temporal Intersection Protocols for Autonomous Vehicles. ACM/IEEE International Conference on Cyber-Physical Systems, (ICCPS 2014), Berlin, Germany, April 14-17, 2014, 1-12. https://doi.org/10.1109/ICCPS.2014.6843706
8. Azimi, S. R., Bhatia, G., Rajkumar, R., Mudalige, P. Ballroom Intersection Protocol: Synchronous Autonomous Driving at Intersections. 21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, (RTCSA 2015), Hong Kong, China, August 19-21, 2015, 167-175. https://doi.org/10.1109/RTCSA.2015.20
9. Biondi, A., Buttazzo, G. C. Modeling and Analysis of Engine Control Tasks Under Dynamic Priority Scheduling. IEEE Transactions on Industrial Informatics, 2018, 14(10), 4407-4416. https://doi.org/10.1109/TII.2018.2791939
10. Cardelli, L., Gordon, A. D. Mobile Ambients. In: Nivat, M. (Ed.), Foundations of Software Science and Computation Structure, Lecture Notes in Computer Science, 1378, Springer, Lecture Notes in Com puter Science, 1998, 140-155. https://doi.org/10.1007/BFb0053547
11. Chen, H., Jiang, J., Hong, Z. Lin, L. Decomposition of UML Activity Diagrams. Software: Practice and Experience, 2018, 48(1), 105-122. https://doi.org/10.1002/spe.2519
12. Dresner, K. M., Stone, P. A Multiagent Approach to Autonomous Intersection Management. Journal of artificial intelligence research, 2008, 31, 591-656. https://doi.org/10.1613/jair.2502
13. Jernigan, J. D., Kodaman, M. F. An Investigation of the Utility and Accuracy of the Table of Speed and Stopping Distances Specified in the Code of Virginia. Technical Report, Virginia Transportation Research Council, 2001
14. Jiang, J., Zhang, S., Gong, P., Hong, Z. Configuring Business Process Models. ACM SIGSOFT Software Engineering Notes, 2013, 38(4), 1-10. https://doi.org/10.1145/2492248.2492267
15. Jiang, J., Zhu, H., Li, Q., Zhang, S., Gong, P., Hong, Z. Configuration of Services based on Virtualization. 2014 Theoretical Aspects of Software Engineering Conference, (TASE 2014), Changsha, China, September 1-3, 2014, 177-184. https://doi.org/10.1109/TASE.2014.28
16. Jiang, J.-M., Zhu, H., Li, Q., Zhao, Y., Hong, Z., Zhang, S., Gong, P. Isolation Modeling and Analysis Based on Mobility. ACM Transactions on Software Engineering and Methodology (TOSEM), 2019, 28(2), 10:1-10:31. https://doi.org/10.1145/3306606
17. Jiang, J.-M., Zhu, H., Li, Q., Zhao, Y., Zhao, L., Zhang, S., Gong, P., Hong, Z. Analyzing Event-based Scheduling in Concurrent Reactive Systems. ACM Transactions on Embedded Computing Systems (TECS), 2015, 14(4), 86. https://doi.org/10.1145/2783438
18. Jiang, J.-M., Zhu, H., Li, Q., Zhao, Y., Zhao, L., Zhang, S., Gong, P., Hong, Z., Chen, D. Event-Based Mobility Modeling and Analysis. ACM Transactions on Cyber-Physical Systems, 2017, 1(2), 9:1-9:32. https://doi.org/ 10.1145/2823353
19. Kim, J., Kim, H., Lakshmanan, K., Rajkumar, R. Parallel Scheduling for Cyber-physical Systems: Analysis and Case Study on a Self-driving Car. ACM/IEEE 4th International Conference on Cyber-Physical Systems (with CPS Week 2013), (ICCPS 2013), Philadelphia, PA, USA, April 8-11, 2013, 31-40. http://doi.acm.org/10.1145/2502524.2502530

20. Kim, K.-D., Kumar, P. R. Cyber-Physical Systems: A Perspective at the Centennial. Proceedings of the IEEE, 2012, 100(Special Centennial Issue), 1287-1308. https://doi.org/10.1109/JPROC.2012.2189792

21. Lee, J., Park, B. Development and Evaluation of a Cooperative Vehicle Intersection Control Algorithm Under the Connected Vehicles Environment. IEEE Transactions on Intelligent Transportation Systems, 2012, 13(1), 81-90. https://doi.org/10.1109/TITS.2011.2178836

22. Lee, J., Shin, K. G. Development and Use of a New Task Model for Cyber-physical Systems: A Real-Time Scheduling Perspective. Journal of Systems and Software, 2017, 126, 45-56. http s://doi.org/10.1016/j.jss.2017.01.004

23. Li, Q. Scheduling in Cyber-Physical Systems, 2012. Dissertation

24. Li, Q., Negi, R. Maximal Scheduling in Wireless Ad Hoc Networks with Hypergraph Interference Models. IEEE Transactions on Vehicular Technology, 2012, 61(1), 297-310. https://doi.org/10.1109/TVT.2011.2176520

25. Liu, C. L., Layland, J. W. Scheduling Algorithms for Multipro- gramming in a Hard-real-time Environment. Journal of the ACM, 1973, 20(1), 46-61. https://doi.org/10.1145/321738.321743

26. Liu, K., Lee, V. C. S., Ng, J. K., Son, S. H., Sha, E. H. Scheduling Temporal Data with Dynamic Snapshot Consistency Requirement in Vehicular Cyber-Physical Systems. ACM Transactions on Em- bedded Computing Systems (TECS), 2014, 13(5s), 163:1-163:21. https://doi.org/10.1145/2629546

27. Martini, D. D., Benetti, G., Vedova, M. L. D., Facchinetti, T. Adaptive Real-time Scheduling of Cyber-Physical Energy Systems. ACM Transactions on Cyber-Physical Systems, 2017, 1(4), 20:1-20:25. https://doi.org/10.1145/3047412

28. Pattanayak, P., Kumar, P. Quantized Feedback Scheduling for MIMO-OFDM Broadcast Networks with Subcarrier Clustering. Ad Hoc Networks, 2017, 65, 26-37. https://doi.org/10.1016/j.adhoc.2017.07.007

29. Perronnet, F., Buisson, J., Lombard, A., Abbas-Turki, A., Ah- mane, M., El Moudni, A. Deadlock Prevention of Self-Driving Vehicles in a Network of Intersections. IEEE Transactions on Intelligent Transportation Systems, 2019, 20(11), 4219-4233. https://doi.org/10.1109/TITS.2018.2886247

30. Schneider, R., Goswami, D., Masrur, A., Becker, M., Chakraborty, S. Multi-layered Scheduling of Mixed-criticality Cyber-physical Systems. Journal of Systems Architecture, 2013, 59(10-D), 1215- 1230. https://doi.org/10.1016/j.sysarc.2013.09.003

31. Stankovic, J. A., Spuri, M., Ramamritham, K., Buttazzo, G. Deadline Scheduling for Real-time Systems: EDF and Related Algorithms. The Springer International Series in Engineering and Computer Science, 460, Springer Science & Business Media, 1998. https://doi.org/10.1007/978-1-4615-5535-3

32. Stone, P., Zhang, S., Au, T.-C. Autonomous Intersection Man- agement for Semi-Autonomous Vehicles. Routledge Handbook of Transportation, 2015, 116-132

33. Tang, Q., Gupta, S. K. S., Varsamopoulos, G. A Unified Methodology for Scheduling in Distributed Cyber-Physical Systems. ACM Transactions on Embedded Computing Systems (TECS) Special Section on CAPA'09, Special Section on WHS'09, and Special Section VCPSS' 09, 2012, 11(S2), 57:1-57:25. https://doi.org/10.1145/2331147.2331162

34. Wang, H., Schmid, F., Chen, L., Roberts, C., Xu, T. A Topology-Based Model for Railway Train Control Systems. IEEE Transactions on Intelligent Transportation Systems, 2013, 14(2), 819-827. https://doi.org/10.1109/TITS.2012.2237509

35. Wang, T., Homsi, S., Niu, L., Ren, S., Bai, O., Quan, G., Qiu, M. Harmonicity-Aware Task Partitioning for Fixed Priority Scheduling of Probabilistic Real-Time Tasks on Multi-Core Platforms. ACM Transactions on Embedded Computing Systems, 2017, 16(4), 101:1-101:21. https://doi.org/10.1145/3064813

36. Wu, W., Zhang, J., Luo, A., Cao, J. Distributed Mutual Exclusion Algorithms for Intersection Traffic Control. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(1), 65-74. https://doi.org/10.1109/TPDS.2013.2297097

37. Xu, D., He, X., Deng, Y. Compositional Schedulability Analysis of Real-Time Systems Using Time Petri Nets. IEEE Transactions on Software Engineering, 2002, 28(10), 984-996. https://doi.org/10.1109/TSE.2002.1041054

38. Zhang, F., Szwaykowska, K., Wolf, W. H., III, V. J. M. Task Scheduling for Control Oriented Requirements for Cyber-physical systems, Proceedings of the 29th IEEE Real-Time Systems Symposium, (RTSS 2008), Barcelona, Spain, 30 November - 3 December, 2008, 47-56. https://doi.org/10.1109/RTSS.2008.52