Automatic Clash Correction Sequence Optimization Using a Clash Dependency Network

Yuqing Hu[1], Daniel Castro-Lacouture[2], Charles M. Eastman[3], and Shamkant B. Navathe[4]

[1] Ph.D. Student, School of Building Construction, Georgia Institute of Technology, Atlanta, GA 30332 (corresponding author). email: yhu390@gatech.edu

[2] Professor, School of Building Construction, Georgia Institute of Technology, Atlanta, GA 30332. email: daniel.castro@design.gatech.edu

[3] Professor, School of Architecture, Georgia Institute of Technology, 245 4th St NW, Atlanta, GA 30332; e-mail: eastman@design.gatech.edu

[4] Professor, College of Computing, Georgia Institute of Technology, 801 Atlantic Dr., NW, Atlanta, GA 30332; e-mail: sham@cc.gatech.edu

Abstract

Building information modeling has demonstrated its advantage to support design coordination, specifically for automatic clash detection. Detecting clashes helps us identify problems, but the process for solving these problems is still manual and time-consuming. This paper proposes using network theory to improve clash resolution by optimizing the clash correction sequence. Building systems are often interdependent of each other, and the dependency relations between building components propagate the impacts of clashes. Ignoring the dependency may cause new clashes when solving a clash or cause iterative adjustments for a single building component. However, a well-organized clash correction sequence can help reduce these issues. Therefore, it is necessary to holistically discuss the clash correction sequence by considering the dependence between clashes. This paper analyzes clash dependencies based on building component dependency relations. We design an optimization algorithm for determining the optimal sequence based on the clash dependency network to minimize feedback dependency, which may cause design rework on a project in project practice. The proposed method is validated on a real building project. After comparing with the natural sequence detected by commercial software, we find that the optimized sequence significantly reduces feedback and automatically groups dependent clashes, which facilitates design coordination.

Keywords: Clash Correction Sequence, Clash Dependency Network, Minimum

Feedback Arc Set, Heuristic Algorithm

1. Introduction

Design coordination is a problem-solving process, which involved experts from multiple disciplines iteratively identify and solve problems to make sure that a design meets its expected functional, economic, and aesthetic requirements [1–3]. With the increase of building complexity, the design coordination process becomes more challenging, specifically among mechanical, electrical, and plumbing (MEP) disciplines. Previous studies argued that the MEP coordination was one of the most challenging tasks for project delivery because it needs to coordinate the location of a large number of interrelated components in a limited space to avoid interferences [2–4]. The cost of MEP coordination is significant, and according to some estimates, it accounts for 6% of MEP cost, while the MEP cost can exceed 50% of the total construction cost on heavily equipped buildings, such as hospitals and laboratories [2,3]. Therefore, effective MEP coordination is important for project success.

In a traditional setting, MEP coordination is manually conducted by specialists from multiple disciplines. They sequentially overlap their transparent 2D drawings on a lighting table to identify component clashes by vision and discuss clash resolutions [5]. Because of the limitation of human vision, the clash detection process is time-consuming. With the application of building information modeling (BIM), automatic clash detection has been widely used in construction projects [6]. BIM can integrate multi-disciplinary models and compute clashes in a federated model based on geometric information of building components [7]. After detecting clashes, BIM coordinators propose these clashes at design coordination meetings for solution discussion. The coordination process can be conducted sequentially or parallelly among multiple disciplines depending on how to build and integrate models to detect clashes. A previous study compared sequential and parallel strategies based on a case study and argued that the parallel method by simultaneously generating multi-disciplinary models and detecting clashes after these models were finished, was less efficient for coordination because clashes were interrelated and simultaneously dealing with many dependency issues was difficult to control potential ripple effects, which increased coordination cycles [2]. Sequential developing models and solving clashes seems more efficient in [2], but the case context focused on building models based on 2D drawings.

Nowadays, many projects adopt model-oriented design methods or 2D and 3D mixed methods. Because of the time pressure from the manufacturing or construction processes, it is difficult to wait for one discipline to fix its model and then develop the model of another discipline. In addition, the adoption of integrated delivery methods and fast-track processes also promotes parallel design [8,9]. To fully unleash the BIM potential, clashes are periodically detected in a federated model that integrates multi-disciplinary models [10–12]. Multi-disciplinary clashes are detected simultaneously. In this scenario, it is important to know how to identify the dependency relations between these clashes and how to organize clash correction sequences to control ripple effects and avoid iterative adjustments.

Previous discussions about BIM-enabled clash correction focused more on the individual clash level to identify clash responsible trades or clash solutions without considering the interaction between clashes and their nearby building components [4,13], which may cause iterative adjustments and increase coordination cycle [2]. Therefore, instead of discussing clash one-by-one, this paper considers clashes from a holistic viewpoint by analyzing the dependency between clashes from a component level and uses the dependency to optimize the clash correction sequence to minimize iterative adjustments. This paper discusses hard clashes among MEP disciplines because the definition of hard clash is accurate and unambiguous. The paper is organized as follows: first, clash dependency scenarios are discussed through analyzing the spatial relations among building components. Then an algorithm is designed for optimizing clash corrections. Finally, the proposed method is validated in a real project and the optimized sequence is compared with the original sequence detected by BIM commercial software to show the feasibility and benefit of the proposed method.

2. Related works

One important task of design coordination is clash management, which includes clash detection and clash correction processes [10]. Traditionally, the two processes are integrated to some degree. Multi-disciplinary specialists detect clashes by sequentially overlapping their transparent 2D drawings on a lighting table and discuss how to solve these clashes after detecting them [5]. With the application of BIM, clash detection and correction processes tend to be separated. BIM coordinators integrate models from multiple disciplines and detect clashes by BIM software. Then, these clashes are proposed in the design coordination meeting and specialists discuss corresponding solutions. Many studies have been conducted to further improve clash detection accuracy by reorganizing BIM models [14], improving clash detection algorithms [15], or using machine learning methods to filter out important clashes [10].

Comparing with the clash detection process, the attention and the automatic level of the clash correction process are much lower. Several research teams have contributed to this field. Wang and Leite [13] pointed out that determining responsible trade is a key issue when correcting clashes. They used machine-learning methods, including decision tree, a rule-based model, and Bayesian method, to train historical data and built a model that can automatically determine the responsible trade for one clash by given required attributes. However, the accuracy of the model (around 70%) still needs to be improved. Korman et al. [4] discussed how to deal with MEP interference from a knowledge management perspective. Based on design criteria and intent, construction, and operations and maintenance knowledge, they used a reasoning structure consisting of model-based reasoning (MBR) and heuristic reasoning to support decision-making in dealing with MEP interference. However, they still focused on a single clash and ignored the dependency between clashes.

Building components are interrelated and the clashes between these components are not isolated [2,16]. From the information processing perspective, an effective coordination system needs the matching of the information processing needs and the information processing abilities, while information needs are generated from information uncertainty [17]. The dependency among building components and among clashes adds information uncertainty, which increase information processing needs [2,17]. Some studies tried to decrease information uncertainty to control information processing needs. Radke et al. [18] mentioned that existing clashes should be solved one-by-one, and the adjustment of each clash should be controlled in a “sticky area” to avoid generate new clashes. Sticky areas were certain locations that objects preferred, which were manually defined in this paper. However, they did not elaborate on how to derive these areas and the size of these areas. In addition, they assumed that for any clashes, the valid space for a clash always existed without impacting nearby objects. This assumption cannot be supported in real projects. In an MEP intensive area, it may be difficult to find a valid space. Lee and Kim [2] discussed coordination strategies based on a case study, and they argued that sequentially generating MEP models by system priorities and detecting clashes can control information uncertainty, which improved coordination efficiency. However, in this sequential strategy, low priority disciplines (for example, electrical system) were modeled until the models of high priority disciplines (for example, HVAC system) were completed. In many situations, the sequential strategy is difficult to conduct because of time pressure. Parallel design is common in many projects and these projects detect clashes in a federated model that integrate multi-discipline [7,10–12].

The above research tried to eliminate the impact between dependent clashes in space by controlling change areas or in time by sequentially coordinating clashes. In their discussions, the dependency is a concept that lack of specific content and measurements. Instead of ignoring dependency or viewing dependency as a negative concept, some studies try to clarify the dependency. Wang and Leite [19] mentioned that clash management should not only focus on clash attributes, but also need to consider clash context, for example, the location of a clash, its spatial relations with nearby objects, and the available space. However, they did not discuss how to represent the information specifically and how to automatically query the context information. Hu et al. [16] classified the dependency between building components into three types: connect, clash, and impact, and discussed how to query this information from models. They used the dependency relations to improve the clash detection process without discussing how this information can support the clash resolution process.

Correcting clashes is the process of change management in nature. Mokhtar et al. [20] depicted a scenario of one space function change. In the scenario, the space change will cause an HVAC duct size change, a beam size change, a wall finishing change, a luminaire type change, and so on. These changes were interrelated. Figuring out their dependency relations and organizing change sequence based on these relations helped to decrease information uncertainty and avoid applying the same change multiple times. This work provides hints for our research. Building components are interdependent and clashes are interrelated through these components. it is imperative to discuss clashes from a holistic view and decide clash correction sequence based on the dependency relations between clashes.

In summary, previous research rarely discussed the dependency between clashes, and they usually solve clashes one-by-one [4] [13]. For a few studies that have realized that the dependency between building systems would impact clash management, one group tried to eliminate the dependency by limiting spatial scopes of clash change [18] or applying sequential design coordination strategies [2]. These methods are difficult to implement in construction projects, especially when projects are complex or have tight schedules. Another group of studies discussed clash context [19] and dependency relations between building components [16], but they did not elaborate on how to use the information to facilitate the decision making of solving clashes. This paper fills the gap and argues that the dependency between clashes should be used to decide the clash correction sequence to minimize the potential iterative adjustments for single clashes. We discuss how to define clash dependency and designed algorithms to search for the optimal correction sequence based on the dependency. This paper proposes a new perspective to solve clashes and presents how to use the information in BIM to refine clash management from a holistic perspective.

3. Methodology

The paper aims to analyze the dependency relations between clashes and based on the dependency structure to optimize clash correction sequence. Network theory is used in this paper to analyze the dependency relations between clashes because a network focuses on depicting relationships between objects rather than the properties of a single object. A network consists of nodes and relations [21]. This paper constructs a clash dependency network considering every clash as a node and the dependency relationships between clashes as edges. Dependency structure matrix (DSM) is widely used to represent dependent activities and analyze their sequence to decrease rework [22–24]. There are three types of relations between activities: dependent, interdependent, and independent, as shown in Figure 1. Independent relations are not discussed in this paper since they do not impact the clash correction sequence. In Figure 1, each red point in the matrix means that correcting the clash located in the row location of the red point will impact the clash located in the column location of the red point. Therefore, in the matrix, super-diagonal elements indicate feedforward information and sub-diagonal elements are feedbacks. Feedforward means pre-activities will impact post-activities. Since when conducting post-activities, their input information (pre-activities) has been fixed, the information processing needs of project teams will not increase because of feedforward dependency. Therefore, it is acceptable in practice. However, feedbacks usually relate to reworks because it means post-activities will impact the pre-activities. Since pre-activities have been finished, changing the pre-condition of the activity may cause the rework of it. Therefore, a reasonable sequence for dependent activities should have as less as sub-diagonal elements/feedbacks in DSM. For example, in Figure 1, the optimization direction of a sequence of four clashes is to change from left order (a) to right order (b). Each clash is considered as an activity in this example.

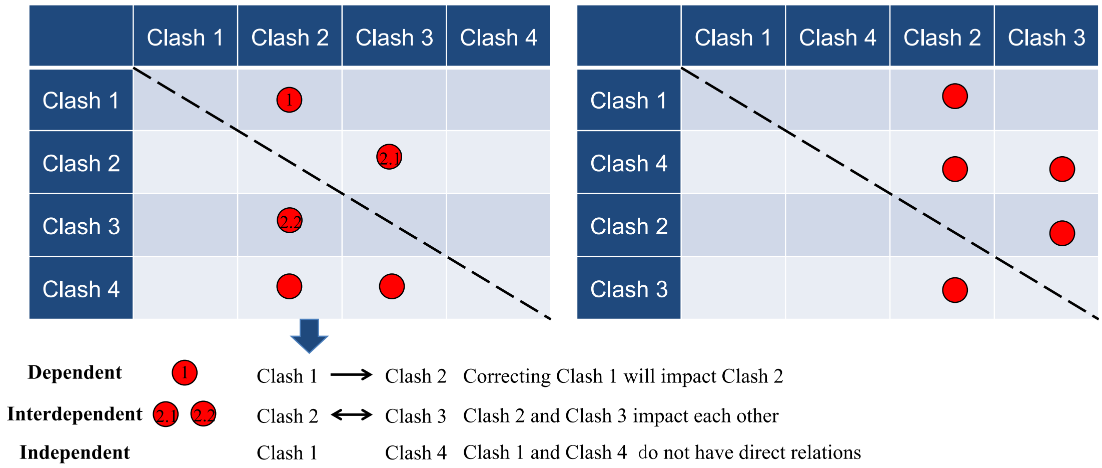


Figure 1. Dependency structure matrix for clash sequence

3.1 Clash Dependency Analysis

The first step for optimizing clash correction sequence consists of identifying the dependency relations between clashes. Previous studies [16,19] did not fully discuss how to define clash dependency. If viewing a correcting clash as a design change, the methods to define change dependency are either manual or automatic. Manual methods use interviews or questionnaires to involve experts in the process to define change dependency [25,26]. However, a project can contain hundreds and thousands of clashes or even more, and the clash coordination time is limited. Therefore, it is difficult to use these methods to identify clash dependency in reality. Instead of manually detecting change dependency, previous studies discussed information in BIM models can be used to analyze component relations and the component relations can represent change dependency [27,28]. For example, “IfcRelConnectsElements” in Industry Foundation Classes (IFC) structure can be used to describe connect relations and changing one component may impact its connected component [28]. A clash is a kind of topology relation between building components in nature [16]. Lee and Kim [2] also argued that clash coordination was complex because moving one building component may affect other components. Therefore, the clash dependency originates from component dependency to some degree. This paper decides to use automatic methods to extract component dependency information from BIM models and based on their relations to define clash dependency.

Many studies discussed the relations between building components [29–34]. Our previous study filtered these relations under the clash management context and classified them into three categories: connection (CO), clash (CL), and impact (IM). We designed algorithms to automatically query these relations from BIM models and built a component dependency network (CDN) for improving clash detection [16]. To improve generality, these algorithms was designed based on IFC format. The elements in a CDN and the methods for querying the information are listed in Table 1. This paper used the CDN as a basis to analyze the dependency relations between clashes to construct the clash dependency network (CLDN) and discussed how the CLDN supports the clash resolution process and optimizes clash correction sequence.

Table 1. Element summary for a component dependency network [16]

Elements		Explanation	Properties	Query Method
Node		Each node represents a building component	GlobalID, IFCType, System Type, Boundary Box Coordinates (minX, minY, minZ, maxX, maxY, maxZ), Component Size Property	Ifc entities was used to query corresponding properties.
Relations	Clash Relation	Represent hard clashes between building components	Minimum move distance of two clash components to avoid the clash in six directions corresponding to the project world coordinate system (AMoveAxisXP, AMoveAxisYP, AMoveAxisZP, AMoveAxisXN, AMoveAxisYN, AMoveAxisZN; BMoveAxisXP, BMoveAxisYP, BMoveAxisZP, BMoveAxisXN, BMoveAxisYN, BMoveAxisZN;)	Clash component id was extracted from clash detection software (e.g. Navisworks). Distance information was calculated by using primitive-based geometric methods. Bounding volume hierarchy (BVH) structure was used to improve computational performance
	Connect Relation	Represent logical connection relations between building components	No extra properties	Connect relations was queried by using Ifc relationship entity, for example:IfcRelConnectsElements, IfcRelConnectsPortToElement, IfcPort, and IfcRelConnectsPorts
	Impact Relation	Impact relations mean that moving one component along a direction in a certain distance, it will impact another component.	Move direction (one of the six directions corresponding to project world coordinate system, AxisXP, AxisYP， AxisZP， AxisXN， AxisYN， AxisZN), The minimum and maximum distances between the impacted component (BLimit, ULimit) and the clash component in the direction	Impact relations were calculated by using primitive-based geometric methods. BVH structure was used to improve computational performance

To construct clash dependency, this paper applied two assumptions. First, to solve a clash, one should move the component with lower priority (low priority principle) and the system type is a key attribute to decide component priority [2,5]. For example, if there is a clash involving an HVAC duct and an electrical conduit, engineers prefer to move the conduit rather than the HVAC duct. Korman and Tatum [5] analyzed the system priority in MEP coordination, from high to low as follows: Dry HVAC, wet HVAC, gravity-driven plumbing system, process piping system, fire protection system, pressure-driven plumbing system, electrical system, control systems, and telephone/data communications. In our proposed method, we use this system priority as the default rank, but also provides users with the flexibility to change the rank based on their project characteristics.

In addition, the clashing volume is another standard to decide the clash sequence. In practice, project participants prefer to solve clashes with a larger clashing volume first rather than some tiny clashes [10]. This paper uses the minimum distance that one object needs to be moved to avoid a clash in the six directions corresponding to the project world coordinate system to represent the clashing volume. Figure 2 is the graphic representation of the clashing volume. For Clash A in this figure, the clashing volume is “d”. As for the distance calculation, our previous study [16] has elaborated on the algorithm by using a bounding volume hierarchy structure based on the axis-aligned bounding box.

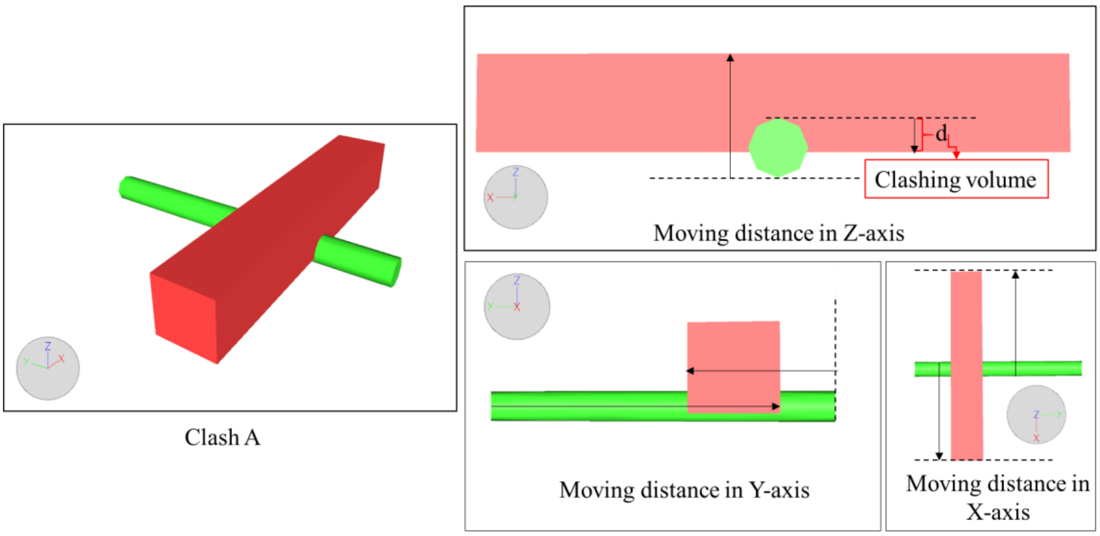


Figure 2. Clashing Volume Representation

In fact, deciding clash sequence based on clashing volume has the following advantages:

1) A clash with a large clashing volume usually needs a large space to fix it, which

may impact a lot of building components and trigger more uncertainties. Therefore, Solving the clash first facilitates to control uncertainties.

2) Sometimes, solving clashes with a larger volume first can automatically solve clashes with a smaller volume. For example, in Figure 3, Clash 1 has a larger clashing volume than Clash 2. If the project team decides to move up the exhaust air duct to solve Clash 1, then Clash 2 is automatically solved.

Therefore, the clashing volume is used to decide the clash sequence when the two clashes have the same system priority combination.

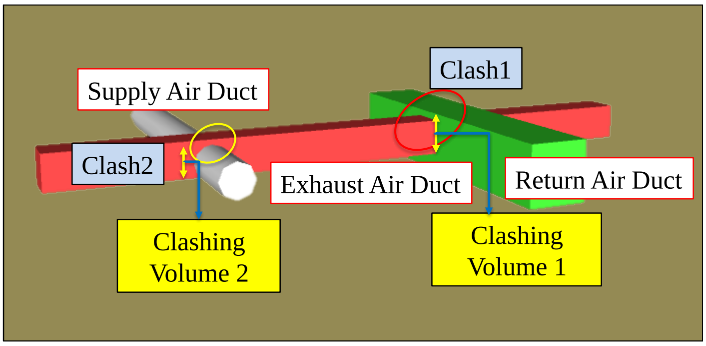


Figure 3. Clashing volume comparison example

Component Dependency Patterns for Two Clashes

We analyze the component dependency patterns by summarizing clash dependency scenarios and the analysis unit is the relationship between two clashes (for example, Clash 1 and Clash 2). If the dependency relations between any two clashes are figured out, the whole dependency network among clashes can be constructed. In order to exhaustively list these scenarios, the paper divides nodes into two types: clash nodes (C) and non-clash nodes (NC). Clash nodes refer to clash components and non-clash nodes refer to other components that have no clash relations in the selected pattern. Two clashes involve three or four clash nodes[1]. Non-clash nodes are media for transferring the impact of clash changes; the number varies from zero to any number. However, project engineers will not allow change to excessively propagate and will attempt to localize change in a small scale. This is the common choice for clash correction based on the constraints from project cost and schedule. This paper analyzes dependency through one non-clash node. Relation class is denoted as R. Node class is denoted as N. The relations incident to a node are denoted as R_N (The relations that relate a clash node

[1] There are situations that three or more building components will overlap at the same location. However, these situations are rare in practice. Therefore, this paper only discusses clashes that are overlapped by two components.

are denoted as R_C, and for a non-clash node as R_{NC}). The relation between Node i and Node j is denoted as $R_{N_i\_N_j}$. The cardinality of relations is denoted as Num_R and of nodes as Num_N. A valid component dependency pattern needs to meet the following constraints:

1) $N \in \{C, NC\}$
2) $R \in \{CO, CL, IM\}$
3) $R_C = CL \cup \{CO, IM\}$
4) $R_{NC} = \{CO, IM\}$
5) $Num_{R_{N_i - N_j}} \in \{0,1\}$
6) $Num_{R_{NC_i - C}} \geq 2$
7) $Num_{R_{NC_i - C_m}} + Num_{R_{NC_i - C_n}} < 2 \quad if\ Num_{R_{C_n - C_m}} = 1$

Constraint 1 means that nodes have two types: clash nodes (C) and non-clash node (NC). Constraint 2 represents the three component dependency relations. Constraints 3 and 4 require that a clash node at least has one clash relation in the pattern and a non-clash node can only have connection or impact relations in the pattern. After cleaning out clashes between connected components [16], the identified three component relations are exclusive. Therefore, Constraint 5 requires that the number of relations between a certain node pair should be 1 or 0. Non-clash nodes serve as intermediate nodes for transferring changes between two clashes. Therefore, a non-clash node at least needs to link with two clash nodes (Constraint 6). Otherwise, it cannot transfer changes. In addition, if Node n and Node m are linked by a clash relation, the situation that one non-clash node connects with the two clash nodes is not considered because clash nodes have been directly linked (Constraint 7) and do not need other node to transfer changes between them. Table 2 summarizes all the valid component patterns in the situation containing three clash nodes, while Table 3 presents all the valid component patterns in the situation containing four clash nodes. Since Clash 1 and Clash 2 are interchangeable, the paper just discusses the situation in which Clash 2 depends on Clash 1 or they are interdependent. The situation in which Clash 1 depends on Clash 2 can be defined by switching the location of them in these standards.

Table 2. Clash dependency scenarios for three clash nodes

Node Type		Relation Type			Graph Representation
No. Of Clash	No. Of Non-clash	No. of Clashes	No. of Connections	No. of Impacts	Clash Node Non-Clash Node

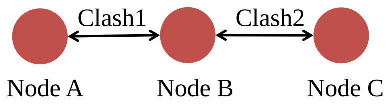
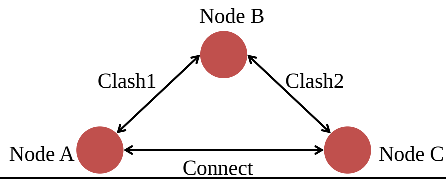
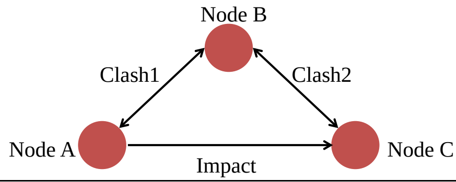
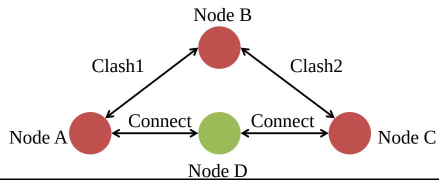
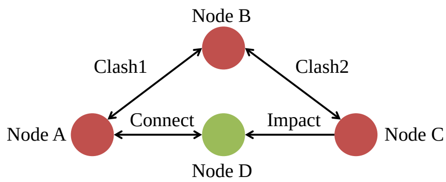
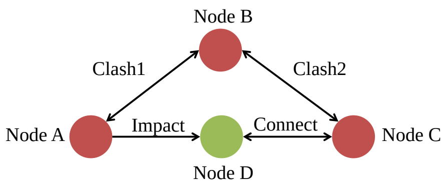
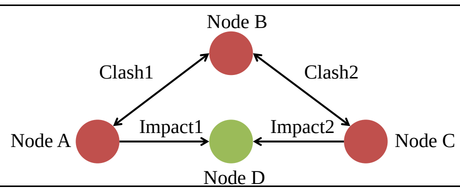
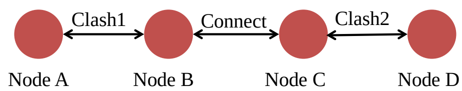
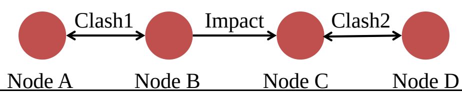
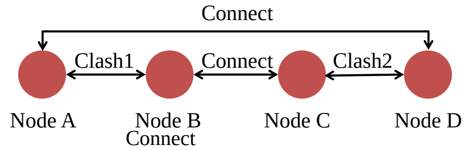
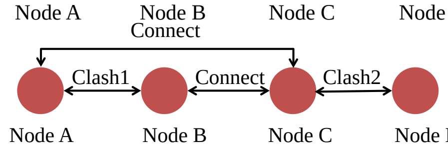
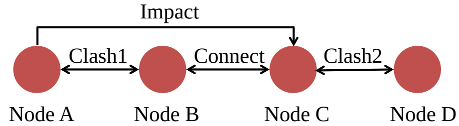
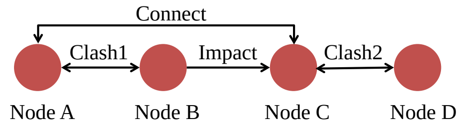
Nodes.	Nodes.				
3	0	2	0	0	3a 
3	0	2	1	0	3b 
3	0	2	0	1	3c 
3	1	2	2	0	3d 
3	1	2	1	1	3e1  3e2 
3	1	2	0	2	3f 

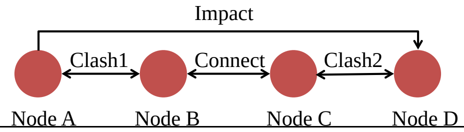
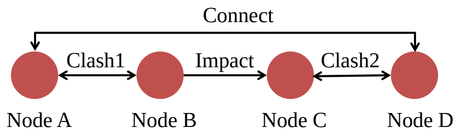
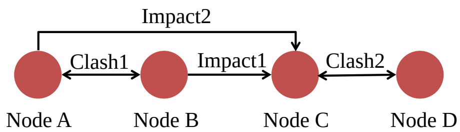
Table 3. Clash dependency scenarios for four clash nodes

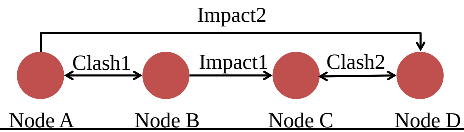
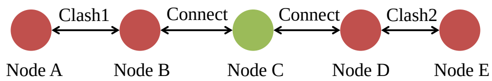
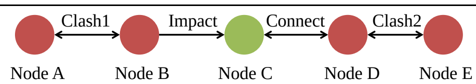
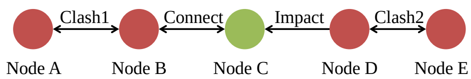
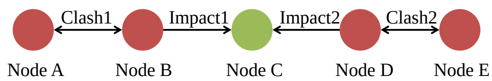
Node Type		Relation Type			Graph Representation
No. Of Clash Nodes.	No. Of Non-clash Nodes.	No. of Clashes	No. of Connections	No. of Impacts	 Clash Node Non-Clash Node
4	0	2	1	0	4a 
4	0	2	0	1	4b 

<table>
<tr><td>4</td><td>0</td><td>2</td><td>2</td><td>0</td><td>4c1


4c2

</td></tr>
<tr><td rowspan="2">4</td><td rowspan="2">0</td><td rowspan="2">2</td><td rowspan="2">1</td><td rowspan="2">1</td><td>4d1


4d2


4d3

</td></tr>
<tr><td>4d4

</td></tr>
<tr><td>4</td><td>0</td><td>2</td><td>0</td><td>2</td><td>4e1


4e2

</td></tr>
<tr><td>4</td><td>0</td><td>2</td><td colspan="2">3 or 4</td><td>can be simplified into 4 edges</td></tr>
<tr><td>4</td><td>1</td><td>2</td><td>2</td><td>0</td><td>4f

</td></tr>
<tr><td rowspan="2">4</td><td rowspan="2">1</td><td rowspan="2">2</td><td rowspan="2">1</td><td rowspan="2">1</td><td>4g1

</td></tr>
<tr><td>4g2

</td></tr>
<tr><td>4</td><td>1</td><td>2</td><td>0</td><td>2</td><td>4h

</td></tr>
<tr><td>4</td><td>1</td><td>2</td><td colspan="2">3-8</td><td>can be simplified into 4 edges</td></tr>
</table>

Clash Dependency Relations based on Component Dependency Patterns

After enumerating the component dependency patterns, the clash dependency relations are analyzed based on different system priority combinations and the clashing volume difference. In the last section, 21 component patterns were discussed. In fact, some complex patterns can be viewed as a combination of some simple patterns, as shown in Table 4. For example, 3d is the combination of 3a or 3b. If Component A has a higher priority than Component B, then for Clash 1, changing Component B is a better choice based on the priority principle, which means Clash 1 and Clash 2 will generate relations through the shared component (Component B). This is equal to 3a, in which the two clashes generate relations based on Component B. Otherwise, if Component B has a higher priority than Component A, the clash dependency is transferred through connection relations, which is equal to 3b.

Table 4. Equivalent simple component dependency patterns

Complex Component Dependency Pattern	Equivalent Simple Component Dependency Pattern
3d	3a, 3b
4c2	4a
4d2	4a, 4b
4e1, 4e2	4b

4d2 is the combination of 4a and 4b. 4c2 can be viewed as two 4a. 4e1 and 4e2 are two 4b. In addition, some patterns are the same with regards to system priority combinations because connected components have the same system priority, as shown in Table 5. Therefore, 21 patterns are simplified into 6 patterns: 3a, 3b, 3c, 3f, 4b, and 4h

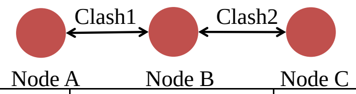
Table 5. Equivalent component dependency pattern pairs

Component Dependency Pattern	Equivalent Component Dependency pattern
4a, 4f	3a
4c1	3b
4d1, 3e1,3e2, 4d2, 4d3, 4d4	3c
4g2, 4g1	4b

Pattern 3a has two clashes sharing a common component (Node B), and the dependency relations are shown in Table 6. The priority of Component A larger than the priority of Component B means that Component A has higher system priority. Taking Situation 5 as an example, in the three objects, Component A has the highest priority and

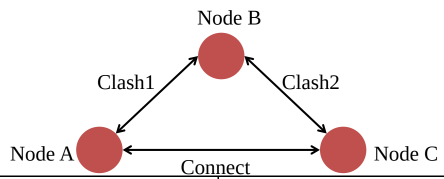
Component C has the lowest priority. According to the low priority principle, when correcting Clash 1, Component B should be changed, which will impact the status of Clash 2 because the location of Node C will be affected by the location of Node B. Therefore, Clash 2 depends on Clash 1. When the system priority is not enough to decide dependency relations, the clashing volume is further used to decide the dependency. For example, in Situation 2, these components have the same priority. When the clashing volume of Clash 1 is larger than Clash 2, based on clashing volume principle, solving Clash 1 and then Clash 2 is a better choice. Therefore, Clash 2 depends on Clash 1.

Table 6. Clash relation analysis-Pattern 3a

Graph representation					
Priority	Component A	Component B	Component C	Clashing Volume	Clash1→Clash2
Situation 1	Priority A= Priority B = Priority C			Clash1=Clash2	Interdependent
Situation 2				Clash1>Clash2	Dependent
Situation 3	Priority A>Priority B && Priority C>Priority B			Clash1=Clash2	Interdependent
Situation 4				Clash1>Clash2	Dependent
Situation 5	Priority A≥Prority B≥Prority C (excluding Situation 1& Situation 2)			No need to consider	Dependent
Situation 6	Others			No Need to consider	Independent

Pattern 3b has two clashes sharing a common component (Node B), and the other two components connected, as shown in Table 7, which is a supplement to Pattern 3a. In Pattern 3a, if Component B has the highest priority, it belongs to an independent situation. In fact, the two clashes will impact each other when Component A and Component C are connected. Since Component A and Component C are connected, they have the same priority from the system perspective. If the clashing volume of Clash 1 is larger, Clash 2 depends on Clash 1. If the clashing volume are the same, they are interdependent.

Table 7. Clash relation analysis- Pattern 3b

Graph representation			
Priority	Component A, B, C	Clashing Volume	Clash1→Clash2

Situation 1	Priority A=Priority C<Priority B	Clash1=Clash2	Interdependent
Situation 2		Clash1>Clash2	Dependent

Pattern 3c has two clashes sharing a common component (Node B), and the other two components impact each other, as shown in Table 8. It is also a supplement to Pattern 3a, similar to Pattern 3b, which discusses the situation when Component B has the highest system priority. This pattern relates to impact relations. When discussing impact relations, the moving direction and whether there is enough room for moving in this direction need be discussed. First, the situation is only considered when the direction of the impact relations is the most promising direction, which is decided by the clash relation. This is because if the impact direction is not the most promising one, it will be hard to find reasons to move in that direction. In this paper, the most promising direction for a component in a clash is defined as the direction that the component needs to move the minimum distance to avoid the clash. For example, in Figure 4, the most promising direction for Component A is the negative Z axis.

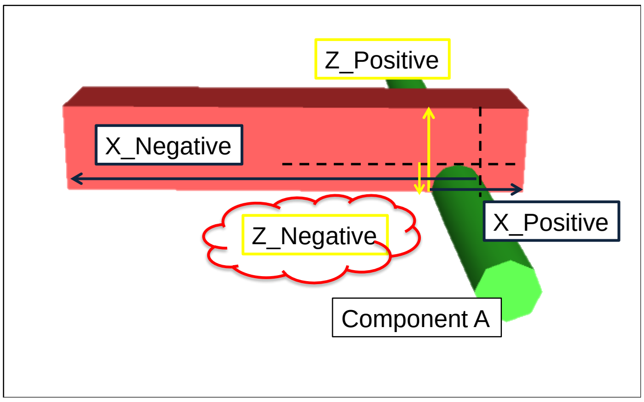
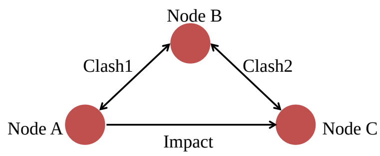


Figure 4. Promising direction for a component

Whether enough room exists is also important for impact relations. Checking enough room for a component consists of comparing the required distance (d1) for avoiding clashes and the distance (d2) with its impacted components in the same direction. If d2 is smaller than d1, this paper defined that enough room does not exist. In pattern 3c, if the most promising direction of Component A is also the opposite promising direction for Component C (Figure 5a), checking enough room needs to compare the distance between A and C in the direction and the sum of the distance for Component A moving along the direction to avoid Clash 1 and Component C moving along the opposite direction to avoid Clash 2. Otherwise, checking enough room is to compare the distance between A and C and the distance required for Component A in its most promising direction (Figure 5b). If enough room exists, the two clashes are independent. Otherwise, they may impact each other. If the system priority of A and C are the same and their clashing volumes are the same, they are interdependent. If the system priority of A is higher than C or the clashing volume of Clash 1 is larger than Clash 2, then Clash 2 depends on Clash 1.

Table 8. Clash relation analysis-Pattern 3c

Graph representation				
Pre-condition	Priority B > Priority A && Priority B > Priority C	Enough room	Clashing Volume	Clash1→Clash2
Situation 1	Priority A = Priority C	No	Clash1=Clash2	Interdependent
Situation 2		No	Clash1>Clash2	Dependent
Situation 3	Priority A > Priority C	No	No need to consider	Dependent

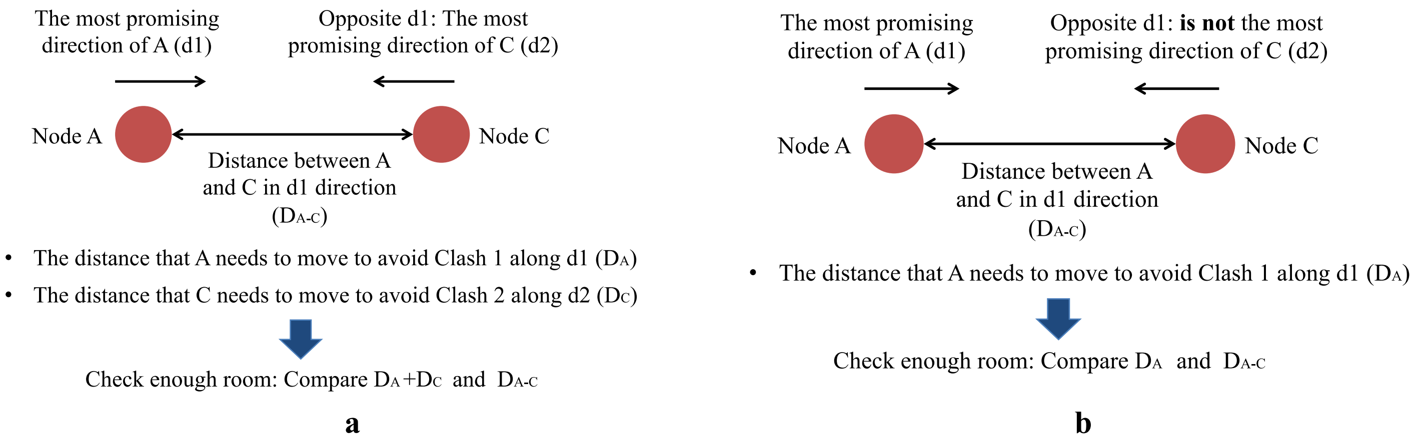


Figure 5. Checking for enough room situations

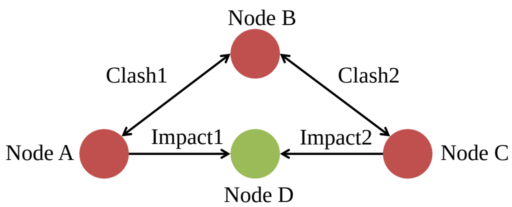
Pattern 3f has two clashes sharing a common component (Node B), while the other two components impact the same object. It is also a supplement to Pattern 3a; in which Component A and Component C are independent (when Component B has the highest priority). In this pattern, the direction of impact relations satisfies two conditions, otherwise the non-clash component cannot transfer changes between the two clashes:

1) The direction of Impact Relation 1 is the opposite direction of Impact Relation 2.

2) The direction of Impact Relation 1 is the most promising direction for Component A.

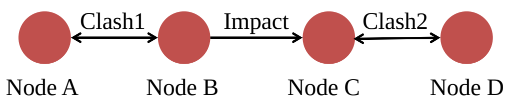
For checking enough room, if the direction of Impact Relation 2 is the most promising direction for Component C, the required distance for the two clashes are the minimum distance required by Component A adding the minimum distance required by Component B. Otherwise, the minimum distance required by Component A compared with the distance between Component A and Component D in the direction of Impact 1 is used to check enough room. The different situations for pattern 3f are shown in Table 9.

Table 9. Clash relation analysis-Pattern 3f

Graph representation				
Pre-condition	Priority B > Priority A && Priority B > Priority C	Enough Room	Clashing Volume	Clash1→Clash2
Situation 1	Priority A = Priority D = Priority C	No	Clash1=Clash2	Interdependent
Situation 2		No	Clash1>Clash2	Dependent
Situation 3	Priority A ≥ Priority D ≥ Priority C (excluding Situation 1)	No	No need to consider	Dependent
Situation 4	Priority A > Priority D && Priority C> Priority D&& Object C has not enough room	No	Clash1=Clash2	Interdependent
Situation 5		No	Clash1>Clash2	Dependent

Pattern 4b contains two clashes that do not share any components, but they impact each other. The methods used to check enough room and define the impact direction is equal to Pattern 3c. The dependency relations are shown in Table 10.

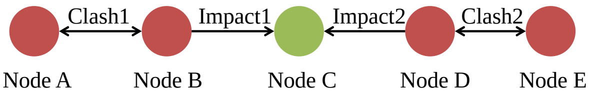
Table 10. Clash relation analysis-Pattern 4b

Graph representation			
Pre-condition	Not Enough Room	Clashing Volume	Clash1→Clash2
Situation 1	Priority A= Priority B = Priority C= Priority D	Clash1=Clash2	Interdependent
Situation 2		Clash1>Clash2	Dependent
Situation 3	Priority A>Priority B = Priority C<Priority C	Clash1=Clash2	Interdependent

Situation 4		Clash1>Clash2	Dependent
Situation 5	Priority A ≥ Priority B ≥ Priority C (excluding Situation 1, 2)	No need to consider	Dependent

Pattern 4h contains two clashes, which impact the same objects. The methods used to check enough room and defined the impact direction requirements is equal to Pattern 3f. The detailed dependency relations are shown in Table 11.

Table 11. Clash relation analysis- Pattern 4h

Graph representation				
Priority	Object A, B, C, D, E	Enough Room	Clashing Volume	Clash1→Clash2
Situation 1	Priority A ≥ Priority B > Priority C ≥ Priority D	No	No need to consider	Dependent
Situation 2	Priority A ≥ Priority B ≥ Priority C& Priority E ≥ Priority D≥ Priority C	No	Clash1=Clash2	Interdependent
Situation 3		No	Clash1>Clash2	Dependent

Clash Dependency Relation Query

This paper uses a Neo4j graph database management system to save component dependency networks because database systems based on a graph data model are better suited for querying graph data, compared with relational databases [35–37]. Neo4j version 3.3.5 is used in this project and Cypher is used as the query language to query the above component dependency patterns. For example, to query pattern 3a, the following query sentence is used: "Match (n1)-[r1:ClashRelationship]-(n2)-[r2:ClashRelationship]-(n3) Unwind[r1.n2MoveAxisZP, r1.n2MoveAxisXP, r1.n2MoveAxisYP, r1.n2MoveAxisZN, r1.n2MoveAxisXN, r1.n2MoveAxisYN] AS clashVolume1 Unwind[r2.n2MoveAxisZP, r2.n2MoveAxisXP, r2.n2MoveAxisYP, r2.n2MoveAxisZN, r2.n2MoveAxisXN,r2.n2MoveAxisYN] AS clashVolume1 return n1.SystemPriority, n2.SystemPriority, n3.SystemPriority, r1.ID As ID1, r2.ID As ID2, min(clashVolume1) as CV1, min(clashVolume2) as CV2". This query returns the system priorities of involved three components and the clashing volumes information. Then the clash dependency relation between the two clashes is decided based component priorities and clashing volumes as discussed above (Table 6).

3.2 Clash Correction Sequence Optimization

The clash dependency network is built using the above pattern analysis (Table 6-Table 11). This network is transferred to a graph where each clash is viewed as a vertex. If two clashes are interdependent, they are connected by bidirectional edges, as shown in Figure 6a. If they are dependent, they are connected by directional edges, as shown in Figure 6b. We use the directed clash graph as an input to optimize clash correction sequence.

Figure 6. Clash dependency network unit

Essentially, optimizing the sequence of activities to minimize feedback translates to a minimum feedback arc set problem (Min-FAS) in graph theory. Min-FAS consists of deleting a minimum number of edges to make a directed graph acyclic. If a directed graph is acyclic (Directed Acyclic Graph-DAG), many algorithms, such as Kahn's algorithm, can be used to calculate a topological sort for the DAG in linear time [38]. A topological sort is a linear order of vertices in a DAG to achieve a sorted order such that for every edge (U, V) from Vertex U to Vertex V, U comes before V in the order, which means that there is no feedback in this order. Therefore, minimizing feedback is equal to finding Min-FAS. However, Min-FAS is a non-deterministic polynomial harness problem (NP-hard), so the computation cost is high. Many studies have designed algorithms to solve this problem, as summarized in Table 1 of the Appendix. In the table, V is the number of nodes in the graph, and E is the number of edges.

These algorithms contain three types: approximation methods, heuristic methods, and exact methods. The best-known approximation ratio cost is proportional to O(logVloglogV) [39]. The approximation methods are usually the fastest in the three types, among which the greedy method can finish in linear time and the KwikSort method has a cost proportional to O(VlogV) (V is the number of nodes in the graph). However, this method cannot guarantee an optimal solution. In fact, most of the time, they cannot achieve an optimal solution. Local search methods start from a candidate solution, iteratively adding perturbations to the solution and moving from this solution to one of the neighboring solutions and using evaluation function to choose among neighboring solutions to realize the continuous improvements. The advantage of this method is that it usually generates an acceptable solution in limited time and the solution is better than approximation methods because it usually uses approximation methods to generate the initial solution. However, the optimal solution still cannot be guaranteed by using this method. Exact methods require an exhaustive search to some degree and they usually adopt some methods to prune the search by eliminating non-promising search space in order to expedite the search speed. Therefore, this method can find an optimal solution, but the computation cost is very high, especially when the size of the problem is large.

In order to find an approach to acquire an optimal solution and control search time, this paper combines a greedy method, linear programming and iterative local search methods to identify min-FAS. Since min-FAS is an NP-hard problem, conducting pre-processing to decrease the size of the problem is important for improving the performance of the algorithms. First, disjointed sets of the given graph are detected by the union-find algorithm [40]. In the context of this paper, disjointed sets mean that there are no dependency relations between these sets, so they can be scheduled in parallel. For each connected set, strongly connected components (SCC) are calculated by Korsaraju's algorithm with O(V+E) complexity [41] (V is the number of nodes in the graph, and E is the number of edges). Using each SCC as input is the common pre-processing method for the FAS problem [42]. SCC is a directed graph in which every node is reachable from any other nodes in the graph. If the number of SCCs is equal to the number of vertices, which means that no circle exists in this set, then Kahn's algorithm is used to calculate a topological sort [43]. Otherwise, the algorithm checks whether the vertices in an SCC are fully connected in both directions. If they are fully connected, all vertices are equivalent, and randomly choosing a sequence is optimum. Otherwise, the algorithm runs the min-FAS algorithm in the SCC. After deciding the optimal sequence for each SCC, these SCCs can be represented as one node, which makes the whole graph acyclic. The overall procedure is shown in Figure 7.

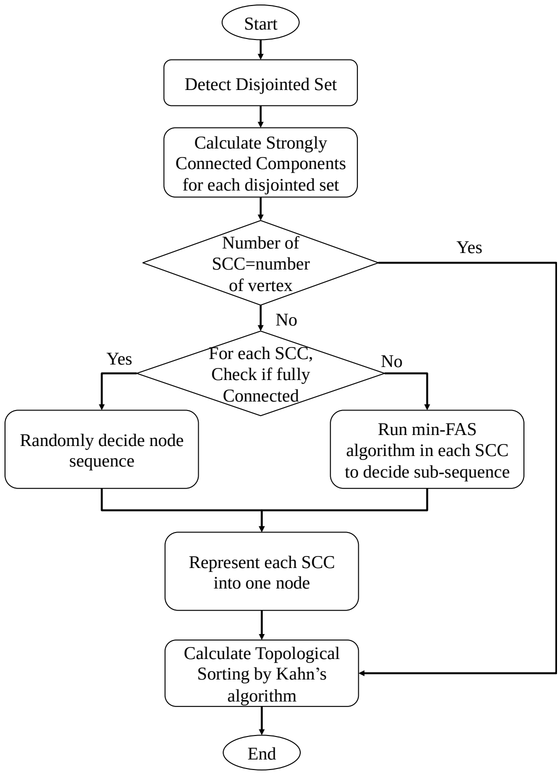


Figure 7. Sequence optimization procedure

The min-FAS algorithm in Figure 7 includes four steps:

Step 1: use a greedy method to generate an initial solution. A feasible solution contains two parts: 1) a vertex order; 2) a feedback arc set, noted as set E. The greedy method is organized as: order vertices by the value of its outdegree minus its indegree in decreasing order, and if the values are the same, order vertices with larger outdegree first. Add edges that their starting vertices come after their ending node in Set E.

Step 2: when the running time is less than a predefined cut-off time, randomly select an edge from set E for removal (removed_edge (U, V))and select all edges that relate with Vertex U and Vertex V in a certain distance (the distance of the edges incident to Vertex U or Vertex V (except edge (U, V)) is one) from E, add these edges (add-back edges noted as E_add) back to the graph. For example, in Figure 7, if edge (D, B) is selected, and the distance is one. Edge (D, C) is a feedback arc and its distance to edge (D, B) is one. Therefore, edge (D, B) and edge (D, C) are added back to the graph.

Step 3: use vertices from Vertex U to Vertex V to constitute a subgraph Gsub, recalculate SCCs, and if the number of E_add is less than the number of SCCs with more than one node, run a linear program to generate the optimal order of nodes in each SCC of Gsub and calculate sub-removed edge set Esub. We explain the formulation of the linear program below. If the size of Esub is smaller than the number of E_add, the algorithm updates the candidate solutions. Otherwise, keep the original solution. To avoid the algorithm stuck in the subproblem, we set a local cut-off time. If in local cut-off time, no better solution is found, the algorithm also keeps the original solution. LpsolveDotNet driver for C# was used to solve the linear programming problem [44]. In Figure 8, if edge (D, B) and edge (D, C) are added back, the subgraph contains Vertex B, Vertex C, and Vertex D, and edge (B, C), edge (D, C) and edge (D, B).

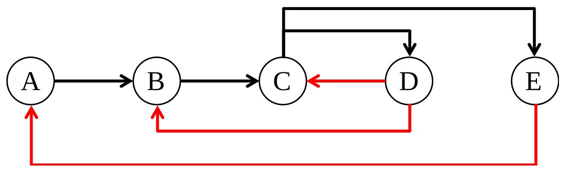


Figure 8. Strongly connected component sequence

Step 4: repeat Step 2 and Step 3 until the solution has no improvement more than pre-defined times, increase the perturbance distance, and repeat Step 2 and Step 3. The algorithm stops when the running time exceeds the pre-defined cut-off time, or the iteration exceeds the pre-defined steps.

The linear programing problem is constructed as follows:

Objective: $\min \sum_{e(u,v)\in G} b(u,v)$

Constraints:

$$d_{ij} \in \{0,1\}, i,j \in [1,V] \tag{1}$$

$$\sum_{j=1}^{V} dij = 1; \sum_{i=1}^{V} dij = 1 \tag{2}$$

$$\forall e(u,v) \in G, \sum_{j=1}^{V} j * dvj - \sum_{j=1}^{V} j * duj + V * b(u,v) \geq 0 \tag{3}$$

$$b(u,v) \in \{0,1\} \tag{4}$$

The design assigns to each vertex a number ranging from 1 to V (number of nodes). dij represents whether Vertex i is ordered in jth position. $\sum_{j=1}^{V} j * duj$ represents the order of Vertex u. $\sum_{j=1}^{V} dij = 1$ and $\sum_{i=1}^{V} dij = 1$ are used to constrain that each vertex has a different order. For each edge e (u, v), if the order of Vertex v is smaller than Vertex u, $b(u, v)$ should be 1 based on the Constraint (3). The objective is to minimize the sum of $b(u, v)$.

The pseudo code for calculating minimum feedback arc set is shown in Appendix 1 (Algorithm 1-Algorithm3).

To validate the robustness of the proposed approach, We use the graphs provided in [45] as the test graphs because these graphs have known min-FAS. The properties of these graphs are shown in Table 12, and the plots of these graphs are shown in Appendix 1 (Figure 1-Figure 10). We used a laptop computer with an Intel-core i7-8750H CPU with 2.21 GHz, and 16.0GB RAM as the testing platform. The results in Table 13 showed that in a given time (the maximum time is 2 seconds), the proposed approach identified the optimal solution in all test graphs. Even though large construction projects can have tens of thousands of clashes, normally these clashes will not belong to the same connected set, and they can be solved set-by-set. For example, in our validated case, we have 191 clashes, the largest connected set only contains 22 nodes (the black circle in Figure 11). Therefore, even though we tested our approach in graphs with up to 109 nodes, it has the capability to calculate the optimal sequence for a larger clash dependency graph.

Table 12. Basic information of test graphs [45]

ID	Nodes	Edges	SCCs (num of nodes>1)	Optimum
1	10	90	1	45
2	12	21	1	2
3	15	35	3	6
4	19	31	1	6
5	25	32	1	3
6	29	37	1	5
7	30	42	1	3
8	41	61	1	5
9	50	79	1	8
10	109	163	1	12

Table 13. Test Results

ID	Cut-off (s)	Local-cutoff (s)	Calculate_Result	Optimal
1	0.1	0.05	45	Yes
2	0.1	0.05	2	Yes
3	0.1	0.05	6	Yes
4	0.1	0.05	6	Yes

5	0.5	0.3	3	Yes
6	0.5	0.3	5	Yes
7	1	0.5	3	Yes
8	2	0.5	5	Yes
9	2	0.5	8	Yes
10	2	1	12	Yes

4. Case Validation

4.1 Project Introduction

We validated the approach presented so far on an actual construction project. The test building is a five-story student residence hall covering 4700-square-meter located in a public university and accommodated 285 beds. The project spanned from Dec 2014 to Aug 2018. Navisworks was used in this project for design coordination and clash detection. Navisworks is a 3D design review tool, which is owned by Autodesk [46]. The proposed method was applied in the coordination of MEP disciplines of the project. To avoid detecting too many tiny clashes that bother the clash coordination process, the project set clash tolerance value as 50mm. According to [7] analysis, around 90% critical clashes have a size larger than 50mm. Under this setting, the project team detected 221 MEP clashes by using Navisworks in a federated MEP model in the detailed design phase. Before analyzing clash dependency relations, we preprocessed the automatically detected clash result by cleaning out irrelevant clashes based on the four scenarios identified in [16], resulting in 191 relevant clashes.

The system classification for these clashes is shown in Figure 9. C# in visual studio 2017 with .Net framework 4.6.1 was used to extract component dependency information from IFC files by using xBIM library [47], and the dependency information (three types of dependency relations: Clash, Connect, Impact relations, listed in Table 1) was saved in a Neo4j database for querying clash dependency relations. Figure 10 displays a screenshot of the user interface to set system priority. System type is extracted from the properties of building components, which were saved in Neo4j and the default rank follows previous studies [5]. Users have the flexibility to change the system priority by selecting and changing the order, and they can also set two systems to have the same priority (see how two systems in the dropdown boxes are set to the same priority in Figure 10). After deciding system priority, the rank information is saved into the Neo4j database. The “Run sequence algorithm” button implements querying component patterns from the database, constructs a clash dependency network, and searches an optimal clash correction sequence. In the validated case, the system priority was kept using the default order.

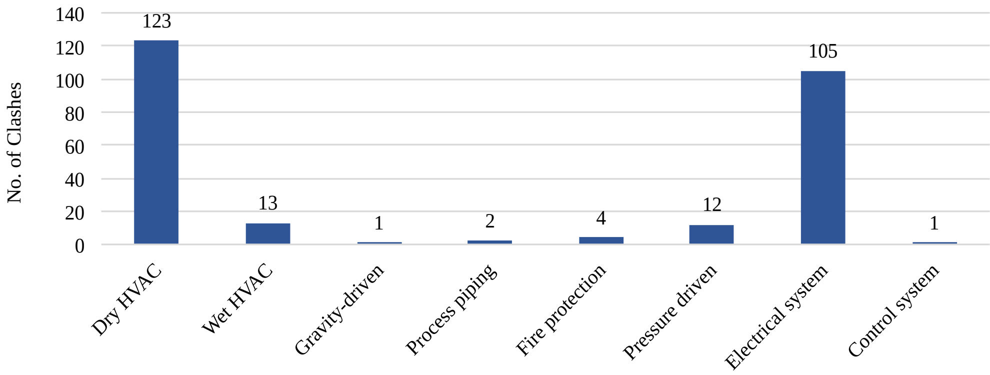


Figure 9. Clash system summary

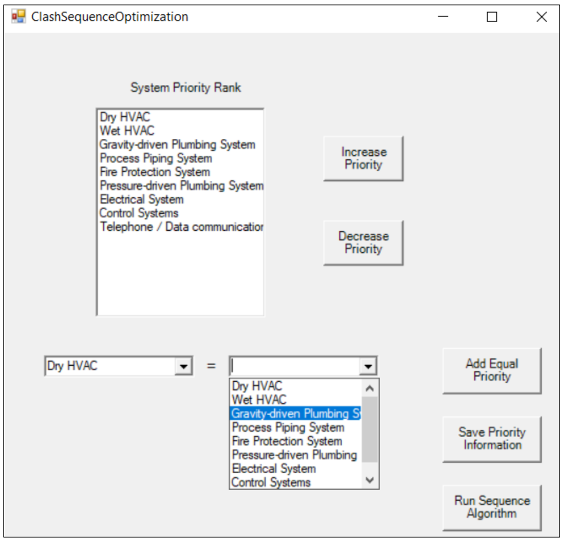


Figure 10. User interface for setting system priority

Figure 11 is the clash dependency network built based on the building component dependency network. In the network, each vertex represents one clash and the vertex ID is the corresponding order in Navisworks. The color and size of vertices are decided by outdegree of a node. Green and small represents low outdegree. The network contains 191 nodes and 281 edges. 58 vertices are isolated. Isolated clashes have no dependency relations with other clashes which can be solved in parallel. 25 disjointed sets exist among the remaining 133 connected nodes, which consist of 24 strongly connected components. These disjointed sets can be scheduled in parallel. These dependency relations are reasonable from the project perspective. For example, in the network, Clash 10 depends on Clash 79 and their geometric representations are shown in Figure 12a. Clash 10 exists between a return air duct and a lighting panel. Clash 79 consists of the intersection between the return air duct and an exhaust air duct. In practice, the mechanical system has a higher priority than the electrical system. Therefore, the location of the lighting panel should be decided after fixing the location of the return air duct. In the graph, it is shown as an edge from Clash 79 to Clash 10. Another interdependent example is shown in SCC1 and their geometric representations are shown in Figure 12b. These clashes exist between a cable tray with an electrical cabinet. In this model, the cabinet consists of seven sets. Navisworks detected seven clashes. However, these clashes are equivalent. Therefore, in the graph, they are interdependent and fully connected.

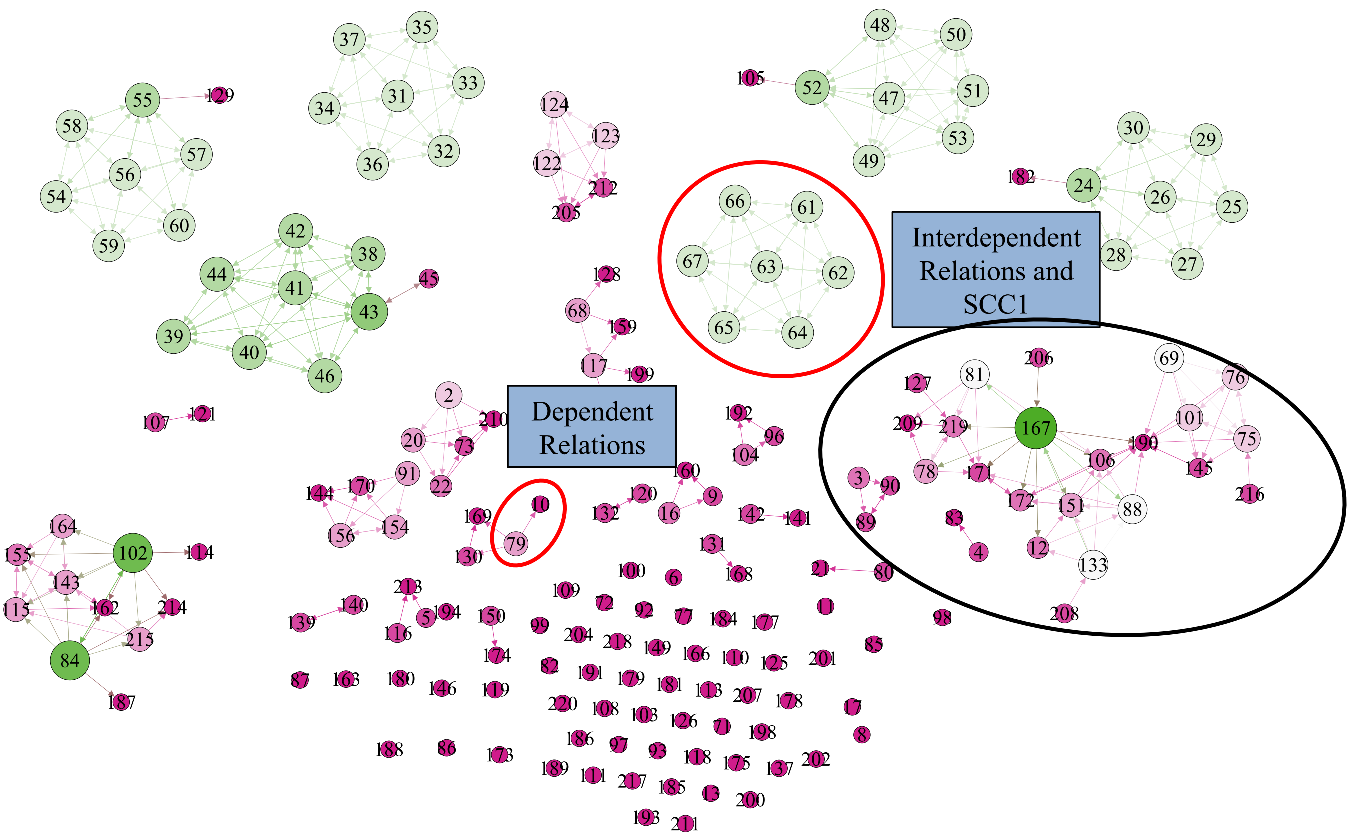
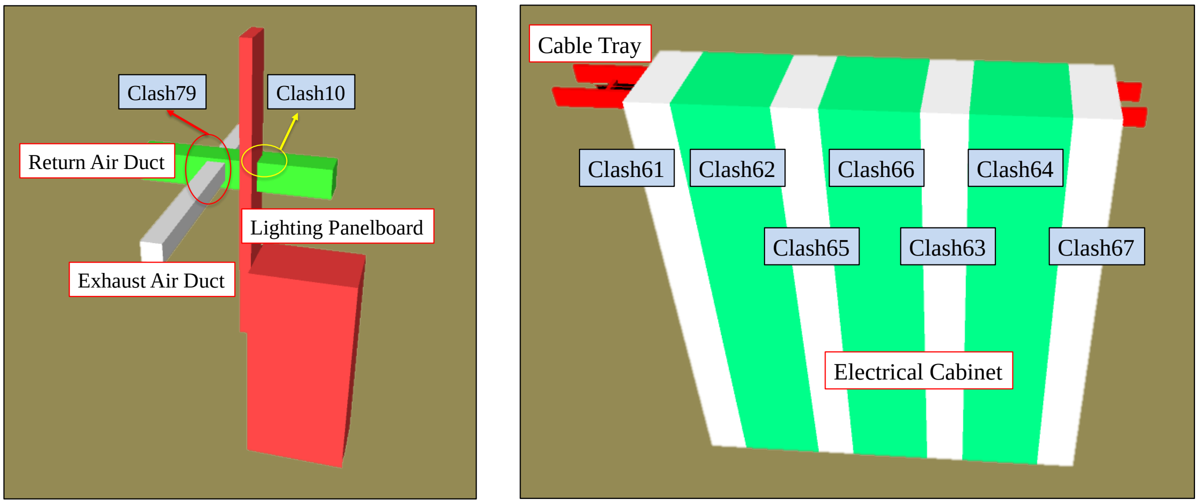


Figure 11. Clash dependency network for the test case



a b

Figure12. Dependency relationship example

4.2 Clash Correction Sequence Optimization Result

The clash dependency network was used as the input for investigating an optimal sequence by using min-FAS algorithm. The algorithm always found an optimal solution in the test case (the optimum solution was captured by using linear programming without time constraints) because the size of each SCC is not very big and the average time to find optimum was around 3000ms in 50 times of test. To compare the result, we constructed a Dependency Structure Matrix following the sequence detected by Navisworks, as shown in Figure 13a and a DSM following the optimized sequence, as shown in Figure 13b. To represent the feedback arc information, the parallel information was not shown in the DSM graphs. In the DSM graphs, red points mean dependency relations and yellow means that the two clashes are independent. Obviously, the two graphs show that the original sequence has more sub-diagonal relations than the optimized sequence. In fact, the feedback arc number decreased from 180 to 150. One example is already listed in Figure 11 and Figure 12a. If the original sequence from Navisworks is followed, Clash 10 will be scheduled before Clash 79. However, in the optimal sequence, Clash 10 depends on Clash 79 and Clash 79 is scheduled before Clash 10. The optimized sequence conforms better with project practice.

Another finding in the graphs is that in the original sequence, the dependency relations are decentralized, which hinders project engineers to notice the dependency between clashes and consider the dependency to optimize the clash resolutions. In the optimized sequence, dependent clashes are closely scheduled because of the disjointed set and strongly connected components calculations, which provides opportunities for project engineers to solve these related clashes from a holistic view rather than focusing on a single clash.

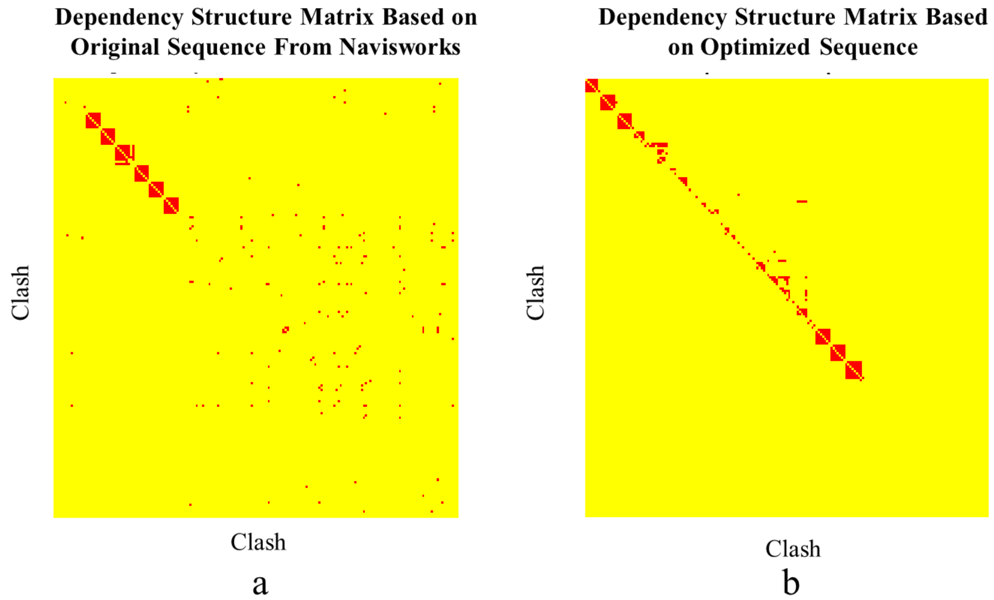


Figure 13a. DSM for Naviswork sequence
Figure 13b. DSM for optimized sequnece

5. Discussion

Clash detection has been viewed as one of the most valued applications of BIM [6]. The adoption of BIM has changed the clash detection practice from visually detecting clashes by sequentially overlapping 2D drawings, to automatically detecting multi-disciplinary clashes in a federated model. Identifying problems is the first step, and how to use the information embedded in BIM models to support the clash correction process is also important. Clashes are interrelated, as moving one building component to solve one clash may affect other components and cause ripple effects [2]. Even though sequentially building models and managing clashes can control ripple effects, under time pressure, parallel design and simultaneously detecting clashes by integrating multi-disciplinary together is common [7,10–12]. We argue that the information embedded in BIM models help to construct a hybrid method that integrates parallel and sequential coordination strategies and decrease information uncertainty to improve clash management.

Multi-disciplinary clashes are detected from a federated model. To organize clash correction sequence and control ripple effects, we analyzed the dependency relations between clashes from a component level and conducted a clash dependency network.

Using the clash network as an input, the paper identified disjointed sets. Since there were no dependency relations among these sets, they could be parallelly solved. For clashes in each jointed set, we argued that dependency should be distinguished as feedback dependency and feedforward dependency. Feedback dependency means that that post-corrected clashes will impact pre-corrected clashes and may cause iterative adjustments and reworks because of information uncertainty, while feedforward dependency is acceptable. Then, the focus tended to minimize feedback dependency. Information inscribed in BIM models helps to refine the clash management strategies to combine parallel and sequential methods, and graph theory provides a method to link clashes and conduct the specific analysis.

BIM has been discussed over many years; instead of acting as a database to store information and a visualization tool to facilitate communication, more analyses and optimizations can be conducted based on BIM information to support the project decision-making process. Dossick and Neff [12] argued that BIM helps project teams to tightly couple technologically by integrating models, while these teams are still divided organizationally. BIM models can be used to analyze information dependency relations and support organization collaboration. We clarified the dependency relations between clashes, which helps to organize organizational coordination. For example, as for feedforward dependency, one-way confirmation is enough, while for interdependency and unavoidable feedback dependency, organizing meetings that integrated related disciplines is a better choice. This paper is one example of using BIM information to support refined management, but more analyses can be conduct based on BIM data to fully exploit the benefits of BIM.

6. Conclusions and Limitations

Clashes are interrelated and the dependency relations between clashes complicate the clash coordination process and may cause ripple effects when correcting one clash [2]. From the information processing perspective, a well-organized clash correction sequence helps to decrease information uncertainty and control change propagation, which improve coordination efficiency. This paper proposed to use graph theory to optimize the clash correction sequence and figured out that the sequence optimization problem was equivalent to the minimum feedback arc set problem from a graph perspective. To construct a clash graph, we discussed how to identify clash dependency relations by analyzing different spatial scenarios, system properties of clash components and clashing volumes. A min-FAS algorithm integrated approximation method, local search, and a linear program were designed to find the optimal sequence. Before running the algorithm, several pre-processing methods, such as calculating disjoined set, strongly connected components, and fully connected SCC, were adopted to decrease the graph size and improve the performance of the algorithm. The proposed method was validated in a project and the results showed that the number of feedback arcs of the optimized sequence had a significant decrease compared with the clash sequence detected by the clash detection software (Navisworks in the validation case). In addition, the optimized sequence automatically grouped dependent clashes together, which provides an opportunity for project participants to discuss clash solutions by considering these related clashes together. This paper concluded that using graph theory and BIM information helped to clarify clash dependency relations and optimize clash correction sequence by mixing parallel and sequential methods.

A large amount of information is embedded in BIM models, which has not been fully used to support design or construction activities. This paper proposed how to use the spatial and system properties of clash components combined with graph theory to facilitate the clash correction process and control ripple effects caused by clash dependency. The limitation of the paper includes two levels. From the research itself, the paper used an automatic method to define clash dependency, which made it possible to be applied in practice. However, manual methods (interviews or survey experts) are able to conduct a more comprehensive assessment about clash dependency by considering various context, for example, production schedule and installation difficulty. How to combine the advantages of the two methods can be further discussed. In addition, this paper focused on MEP clashes, even though MEP coordination is the most challenging part for complex projects, clashes between MEP disciplines and structural components are also important and other non-MEP clashes also need to be solved. The method to identify dependency among MEP clashes and among non-MEP clashes can be different. For example, solving clashes between structural components (e.g. slabs or walls), sometimes, needs to consider the locations of preformed holes. How to discuss these openings to identify clash dependency to further improve the generality of the proposed methods is worthy of further discussions. Then, as for implementing in construction projects, the paper mainly discussed clash correction sequence optimization from a technical perspective. However, to implement the sequence, it is needed an organizational support and a well-organized multi-disciplinary collaboration process. It can be imagined that the sequence can be implemented more easily in a project environment with project teams integrated and working together. Therefore, how to integrate the method in the design coordination practice is worthy of further study. Furthermore, in practice, some clashes may have a fixed sequence because of organizational or management requirements. How to change the current system to allow parts of fixed sequences can be discussed further. One potential solution is to set weights to the edges between clashes. Therefore, how to capture requirements to identify unchangeable clash sequence, set weights to clash dependent relations, and change the proposed method to involve edge weights need to be answered in the future. Third, the clash graph constructed in this paper is still a static graph that analyzes the clash dependency relations in one federated model. However, in many projects, clash detection is periodically conducted with a project going. In the process, parts of old clashes can be solved, and new clashes can be detected. How to deal with the dynamics of design and how to continually update the clash graph in an effective way can be a future research direction.

Reference

1. Dorst K, Dijkhuis J, Comparing Paradigms for Describing Design Activity, Design Studies. Vol.16, (1995), pp.261–274, https://doi.org/10.1016/0142-694X(94)00012-3
2. Lee G, Walter J, Parallel vs . Sequential Cascading MEP Coordination Strategies : A Pharmaceutical Building Case Study, Automation in Construction. Vol.43, (2014), pp.170–179, http://dx.doi.org/10.1016/j.autcon.2014.03.004
3. Mehrbod S, Staub-french S, Mahyar N, Tory M, Characterizing Interactions with BIM Tools and Artifacts in Building Design Coordination Meetings, Automation in Construction. Vol.98, (2019), pp.195–213, https://doi.org/10.1016/j.autcon.2018.10.025
4. Korman TM, Fischer MA, Tatum CB, Knowledge and Reasoning for MEP Coordination, Journal of Construction Engineering and Management. Vol.129, (2003), pp.627–634. http://ascelibrary.org/doi/10.1061/%28ASCE%290733-9364%282003%29129%3A6%28627%29
5. Korman TM, Tatum CB, Development of a Knowledge-Based System to Improve Mechanical, Electrical, and Plumbing Coordination, CIFE Technical Report #129. (2001), pp.1–162. www.stanford.edu/group/CIFE/Publications
6. Cao D, Wang G, Li H, Skitmore M, Huang T, Zhang W, Practices and Effectiveness of Building Information Modelling in Construction Projects in China, Automation in Construction. Vol.49, (2015), pp.113–122. http://dx.doi.org/10.1016/j.autcon.2014.10.014
7. Pärn EA, Edwards DJ, Sing MCP, Origins and Probabilities of MEP and Structural Design Clashes within a Federated BIM Model, Automation in Construction. Vol.85, (2018), pp.209–219. https://doi.org/10.1016/j.autcon.2017.09.010
8. Peña-Mora F, Li M, Dynamic Planning and Control Methodology For Design/Build Fast-Track Construction Projects, Journal of Construction Engineering and Management. Vol.127, (2001), pp.1–17. https://doi.org/10.1061/(ASCE)0733-9364(2001)127:1(1)
9. Ilozor BD, Kelly DJ, Building Information Modeling and Integrated Project Delivery in the Commercial Construction Industry : A Conceptual Study, Journal

of Enginnering, Project, and Production Management. Vol.2, (2012), pp.23–36. http://www.ppml.url.tw/EPPM_Journal/volumns/02_01_January_2012/ID_013_2_1_23_36.pdf

10. Hu Y, Castro-Lacouture D, Clash Relevance Prediction Based on Machine Learning, Journal of Computing in Civil Engineering. Vol.33, (2018). http://ascelibrary.org/doi/10.1061/%28ASCE%29CP.1943-5487.0000810

11. Wang J, Wang X, Shou W, Chong HY, Guo J, Building Information Modeling-based Integration of MEP Layout Designs and Constructability, Automation in Construction. Vol.61, (2016), pp.134–146. http://dx.doi.org/10.1016/j.autcon.2015.10.003

12. Dossick CS, Neff G, Organizational Divisions in BIM-Enabled Commercial Construction, Journal of Construction Engineering and Management.Vol.136, (2010),pp.459–467. http://ascelibrary.org/doi/10.1061/%28ASCE%29CO.1943-7862.0000109

13. Wang L, Leite F, Knowledge Discovery of Spatial Conflict Resolution Philosophies in BIM- enabled MEP Design Coordination using Data Mining Techniques: a Proof-of- Concept, Proceedings of The International Workshop on Computing in Civil Engineering. Los Angeles, California: American Society of Civil Engineers. (2013), pp. 399–404. https://doi.org/10.1061/9780784413029.053

14. Hartmann T, Detecting Design Conflicts using Building Information Models: a Comparative Lab Experiment, Proceedings of The 27th International Conference - Applications of IT in the AEC Industry & Accelerating BIM Research Workshop. Cairo, Egypt: Virginia Tech. (2010), pp.16–18. http://itc.scix.net/cgi-bin/works/Show?w78-2010-57

15. Van den Helm P, Böhms M, van Berlo L, IFC-based Clash Detection for the Open-Source BIMserver, Proceedings of The International Conference on Computing in Civil and Building Engineering. Nottingham, UK: Nottingham University Press. Vol.181, (2010), http://bimserver.org/wp-content/uploads/sites/6/2010/11/Helm_Clashdetection.pdf

16. Hu Y, Castro-lacouture D, Eastman CM, Holistic Clash Detection Improvement Using a Component Dependent Network in BIM Projects, Automation in

Construction. Vol.105, (2019). https://doi.org/10.1016/j.autcon.2019.102832

17. Galbraith JR, Organization Design : An Information Processing View, Interfaces. Vol.4, (1974), pp.28–36. https://www.jstor.org/stable/25059090

18. Radke AM, Wallmark T, Tseng MM, An Automated Approach for Identification and Resolution of Spatial Clashes in Building Design, Proceedings of The International Conference on Industrial Engineering and Engineering Management. Hong Kong, China: IEEE. (1974), pp.2084-2088. https://ieeexplore.ieee.org/abstract/document/5373167

19. Wang L, Leite F, Formalized Knowledge Representation for Spatial Conflict Coordination of Mechanical, Electrical and Plumbing (MEP) Systems in New Building Projects, Automation in Construction. Vol.64, (2016), pp.20-26. http://dx.doi.org/10.1016/j.autcon.2015.12.020

20. Mokhtar A, Bedard C, Fazio P, Collaborative Planning and Scheduling of Interrelated Design Changes, Journal of Architectural Engineering. Vol.6, (2000), pp.66–75. https://doi.org/10.1061/(ASCE)1076-0431(2000)6:2(66)

21. Chinowsky P, Diekmann J, Galotti V, Social Network Model of Construction, Journal of Construction Engineering and Management. Vol.134, (2008), pp.804–812. https://doi.org/10.1061/(ASCE)0733-9364(2008)134:10(804)

22. Maheswari JU, Varghese K, A Structured Approach to Form Dependency Structure Matrix for Construction Projects, Proceedings of The 22nd International Symposium on Automation and Robotics in Construction. Ferrara, Italy. (2005), pp. 1–6. https://doi.org/10.22260/ISARC2005/0062

23. Oloufa AA, Hosni YA, Fayez M, Axelsson P, Using DSM for Modeling Information Flow in Construction Design Projects, Civil Engineering and Environmental Systems. Vol.21, (2004), pp.105–125. https://doi.org/10.1080/10286600310001638474

24. Srour IM, Abdul-Malak MAU, Yassine AA, Ramadan M, A Methodology for Scheduling Overlapped Design Activities based on Dependency Information, Automation in Construction. Vol.29, (2013), pp.1–11. http://dx.doi.org/10.1016/j.autcon.2012.08.001

25. Zhao ZY, Lv QL, Zuo J, Zillante G, Prediction System for Change Management in Construction Project, Journal of Construction Engineering and Management.

Vol.136, (2010), pp.659–669. https://doi.org/10.1061/(ASCE)CO.1943-7862.0000168

26. Wynn DC, Caldwell NHM, John Clarkson P, Predicting Change Propagation in Complex Design Workflows, Journal of Mechanical Design. Vol.136, (2014). http://mechanicaldesign.asmedigitalcollection.asme.org/article.aspx?doi=10.1115/1.4027495

27. Pilehchian B, Staub-french S, A Conceptual Approach to Track Design Changes Within a Multi-disciplinary Building Information Modeling Environment, Canadian Journal of Civil Engineering. Vol.152, (2015), pp.139–152. https://doi.org/10.1139/cjce-2014-0078

28. Isaac S, Navon R, A Graph-based Model for the Identification of the Impact of Design Changes, Automation in Construction. Vol.31, (2013), pp.31–40. http://dx.doi.org/10.1016/j.autcon.2012.11.043

29. Nguyen T-H, Oloufa AA, Computer-Generated Building Data: Topological Information. Journal of Computing in Civil Engineering, Vol.15, (2001), pp.268–274. https://doi.org/10.1061/(ASCE)0887-3801(2001)15:4(268)

30. Solihin W, A Simplified BIM Data Representation using a Relational Database Schema for an Efficient Rule Checking System and its Associated Rule Checking Language, (2016), Ph.D. diss, School of Architecture, Georgia Insititute of Technology. http://hdl.handle.net/1853/54831

31. Borrmann A, Rank E, Topological Analysis of 3D Building Models using a Spatial Query Language, Advanced Engineering Informatics. Vol.23, (2009), pp.370–385. http://dx.doi.org/10.1016/j.aei.2009.06.001

32. Nepal MP, Staub-French S, Pottinger R, Webster A, Querying a Building Information Model for Construction-Specific Spatial Information, Advanced Engineering Informatics. Vol. 26, (2012), pp.904–923. https://doi.org/10.1016/j.aei.2012.08.003

33. Khalili A, Chua DKH, IFC-Based Graph Data Model for Topological Queries on Building Elements, Journal of Computing in Civil Engineering. Vol.29, (2015). https://doi.org/10.1061/(ASCE)CP.1943-5487.0000331

34. Solihin W, Eastman C, Lee Y-C, Yang D-H, A Simplified Relational Database Schema for Transformation of BIM Data into a Query-efficient and Spatially

Enabled Database, Automation in Construction. Vol.84, (2017), pp.367–383. https://doi.org/10.1016/j.autcon.2017.10.002

35. Elmasri R, Navathe SB, Fundamentals of Database System. six. Addison-Wesley. (2016), p. 1242. http://iips.icci.edu.iq/images/exam/databases-ramaz.pdf
36. Huang H, Dong Z, Research on Architecture and Query Performance based on Distributed Graph Database Neo4j, Proceedings of The 3rd International Conference on Consumer Electronics, Communications and Networks. Xianning, China: IEEE. (2013), pp.533–536. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6703387&isnumber=6703249
37. Vicknair C, Nan X, Chen Y, Wilkins D, A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective, Proceedings of The 48th Annual Southeast Regional Conference. New York, USA: Association for Computing Machinery. Vol. 42, (2010), pp.1–6. http://doi.acm.org/10.1145/1900008.1900067
38. Barnat J, Brim L, Ročkai P, Parallel Partial Order Reduction with Topological Sort Proviso. Proceedings of The 8th IEEE International Conference on Software Engineering and Formal Methods. Pisa, Italy: IEEE. (2010), pp.222–231. https://ieeexplore.ieee.org/document/5637433
39. Even G, Naor J, Schieber B, Sudan M, Approximating Minimum Feedback Sets and Multicuts, Algorithmica. Vol.20, (1998), pp.151–174. https://doi.org/10.1007/PL00009191
40. Wikipedia, Disjoint-Set Data Structure. (February 23, 2020) https://en.wikipedia.org/wiki/Disjoint-set_data_structure
41. Wikipedia, Kosaraju’s Algorithm.(March 7, 2019) https://en.wikipedia.org/w/index.php?title=Kosaraju%27s_algorithm&oldid=886695367
42. Saab Y, A Fast and Effective Algorithm for the Feedback Arc Set Problem, Journal of Heuristics. Vol.7, (2001), pp.235–250. https://doi.org/10.1023/A:1011315014322
43. Kahn AB, Topological Sorting of Large Networks, Communications of the ACM. Vol.5, (1962), pp.558-562. https://doi.org/10.1145/368996.369025

44. Gosselin M, Linear Programming Solver. (2018). https://www.nuget.org/packages/LpSolveDotNet/
45. Baharev A, Schichl H, Neumaier A, An Exact Method for the Minimum Feedback Arc Set Problem. University of Vienna. Vol.10, (2015), pp.35–60. https://www.mat.univie.ac.at/~neum/ms/minimum_feedback_arc_set.pdf
46. Wikipedia, Navisworks. (February 11, 2020). https://en.wikipedia.org/wiki/Navisworks
47. Lockley S, XBIM -The BIM Toolkit. (2015). https://github.com/xBimTeam/
48. Eades P, Lin X, Smyth WF, A Fast and Effective Heuristic for the Feedback Arc Set Problem, Information Processing Letters. Vol.47, (1993), pp.319–323. https://doi.org/10.1016/0020-0190(93)90079-O
49. Ailon N, Charikar M, Newman A, Aggregating inconsistent information: Ranking and Clustering, Journal of the ACM. Vol.55, (2008), pp.1–27. https://dl.acm.org/citation.cfm?doid=1411509.1411513
50. Meier C, Yassine AA, Browning TR, Design Process Sequencing With Competent Genetic Algorithms, Journal of Mechanical Design. Vol.129, (2007), pp.566–585. http://mechanicaldesign.asmedigitalcollection.asme.org/article.aspx?articleid=1449386
51. Brandenburg FJ, Hanauer K, Sorting Heuristics for the Feedback Arc Set Problem, Technical Report MIP-1104, University of Passau, Germany. (2011). https://pdfs.semanticscholar.org/220c/e6b7b095b4608e671ca5ea9889ac0f5d09d8.pdf
52. Hecht M, Exact Localisations of Feedback Sets, Theory of Computing Systems. Vol.62, (2018), pp.1048–1084. https://link.springer.com/article/10.1007/s00224-017-9777-6

Appendix

Table 1. Algorithm Summary for Min-FAS (V is the number of vertices and E is the number of edges)

Method Type	Author	Description	Time Complexity & Performance	Method Comments
Approximation method	Eades et al. [48]	Greedy method: order nodes by outdegree-inDegree (ELS) or abs(outdegree-inDegree) (ELS-abs), and add the feedback arc into FAS	Time complexity: O(V+E) Upper bound E/2-V/6	Advantage: very fast, finish in linear time, easy to implement Disadvantage: usually can not find optimal solutions
	Even et al.[39]	Using sphere-growing technique to acquire approximated optimal solution	Time complexity: polynomial time Approximation ratio is O(log V loglogV)	Advantage: the bound is tighter than Greedy method Disadvantage: optimal solutions are not guaranteed
	Ailon et al. [49]	KwikSort: sorting the order of notes in Graph based on quicksort methods and delete feedback arc	Time complexity: O(nlog(n)) Approximation ratio is 3	Advantage: fast and easy to implement Disadvantage: usually cannot find optimal solutions
Heuristic/Local search methods	Saab [42]	Divide & Conquer+ stochastic evolution/dynamic cluster: Divide & Conquer is to divide graph G into subgraphs G1 and G2 and order nodes in G1 before nodes in G2. FAS(G)=FAS(G1) U FAS(G2) U {i → j :i ∈ G2 and j ∈ G1}. stochastic evolution/dynamic cluster are used to find optimal bisection of graph	The time and performance depend on iterative times, and some initial iterative parameters	Advantage: solutions can keep improving compared approximation methods Disadvantage: optimal solutions are not guaranteed,
	Meier et al. [50]	Genetic algorithm: conduct position-based crossover and shift mutation to guarantee feasible solutions in these phases		

	Brandenburg and Hanauer [51]	KwikSort heuristic methods-KwikSort with randomly choose initial pivot		
Exact methods	Baharev et al. [45]	Linear programming	The constrain is O(n3) or the number of simple circles in the graph, which can be $\Omega(2n)$ Exact answer	Advantage: Optimal solution can be found Disadvantage: time consuming
	Hecht [52]	Dynamic programming	Time Complexity: When FAS possess Bellman principle. $O(2m\|E\|4 \log(\|V\|) m \leq \|E\| - \|V\| + 1$ Exact answer	

Algorithm 1 MinFAS Algorithm

```
function MinFAS(Graph G)
    List< Vertex > vertexOrder ← ∅
    SCCs ← G.CalculateSCC // SCCs is a list of subgraph of G and SCC
                              is calculated by Korsaraju's algorithm
    SCCs.order() //view each SCC as one vertices,
                   using Kahn's algorithm to order SCCs.
    for SCC in SCCs do
        subVertexOrder ← ∅
        boolean Connected ← checkFullyConnected(SCC) //compare the
                              number of edges and the number of vertices
        if Connected then
            subVertexOrder ← randomly assign one
        else
            subVertexOrder ← subMFAS(SCC, worldCutoff, localCutoff)
        vertexOrder.add(subVertexOrder)
    return vertexOrder
```

Algorithm 2 SubMFAS Algorithm

```
function SubMFAS(SCC, worldCutoff, localCutoff)
    bestSol ← GreedyMFAS(SCC) // order vertices by the value of
                outdegree-indegree in decreasing order (if the values are the
                same, order the vertex with larger outdegree first), add the
                feedback arc in this order as the feedback arc set
    bestCost ← the number of feedback arc in bestSol
    distance ← 0
    iterationsPerDistance ← bestCost*3/4 // update 75% of initial solution,
                                           the number can be changed
    iter ← 0
    while time < worldCutoff do
        newSol ← bestSol
        if iter ≥ iterationsPerDistance then
            iter ← 0
            distance ← distance+1
            localCutoff ← localCutoff*1.1 // when disturbance distance
                            increases, the size of sub problems increases,
                            which needs more time to find solutions
        addBackSet ← ∅
        $E_{uv}$ ← bestSol.feedbackArcs.randomSelect() // random select an
                edge from the feedback arc sets
        newSol.removeWithinDistance($E_{uv}$,distance) //removes feedback
                                    arcs within distance of $E_{uv}$ and $E_{uv}$
        solFound ← linearProgramming(newSol, localCutoff) //Runs a linear
                    programming on the subproblem with a local cutoff time
        if solFound then
            if newSol ≤ bestSol then
                bestSol ← newSol
                bestCost ← newSol.feedbackArc.Count()
                iterationsPerDistance ← bestCost*3/4
        iter ← iter + 1
    return bestSol
```

Algorithm 3 Linear Programming Algorithm

```
function LinearProgramming(newSol, localCutoff)
    addBackSet ← newSol.removedFeedbackArcs
    subG ← getSubGraph // the sub graph contains all vertices between
                the first vertex and last vertex in addBackSet, and all edges
                incidents to these vertices
    SCCs ← subG.CalculateSCC
    lowBound ← the number of SCC with more than one vertices in SCCs
            //at least one feedback arc (FA) exists in this kind of SCC
    upperBound ← addBackSet.count() // the sub problem should have a
     better solutions than original, otherwise using original solutions
    if lowBound ≥ upperBound then
        return false
    subSols ← ∅
    SCCs.order() //view each SCC as one vertices,
                    using Kahn's algorithm to order SCCs.
    for SCC in SCCs do
        while time > localCutoff do
            subSol ← copy the original sequence in SCC
            subSols.add(subSol)
        objectBound ← upperBound-(lowBound-1) //lowBound-1 means
         that except this SCC, at least (lowBound-1) circles exists in the
         subGraph. Therefore, for this SCC, the number of FA cannot exceed
         upperBound-(lowBound-1). Otherwise, the sum of FA for the sub
         graph will exceed upperBound
        if objectBound ≤ 0 then
            return false
        subSol ← conduct linear programing by LpSolveDotNet
                solver with objectBound as bound and
                localCutoff-time as time constraints
        subSols.add(subSol)
        upperBound ← upperBound-subSol.FeedbackArcs.Count()
        lowBound ← lowBound-1
    newSol.updateSolution(subSols) // update overall solution by replacing
                                        part of solutions with subSols
    return true
```

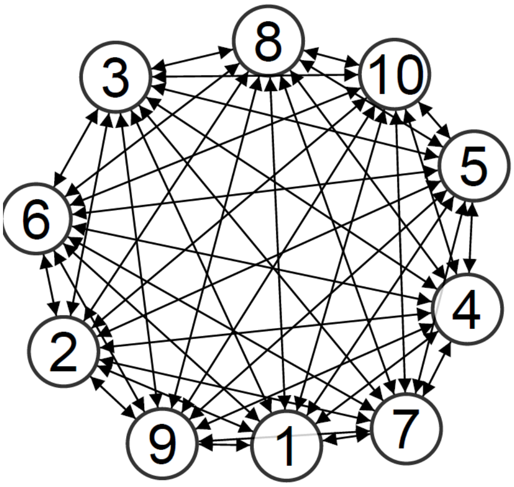


Figure 1. Test Graph 1[45]

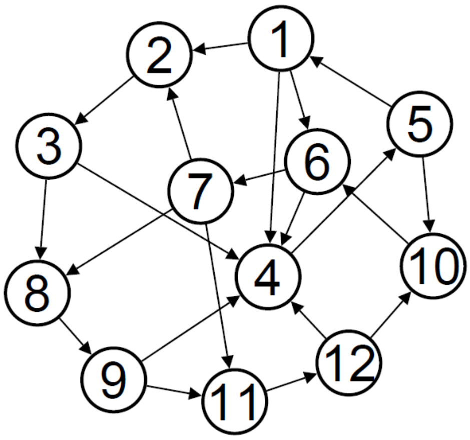


Figure 2. Test Graph 2[45]

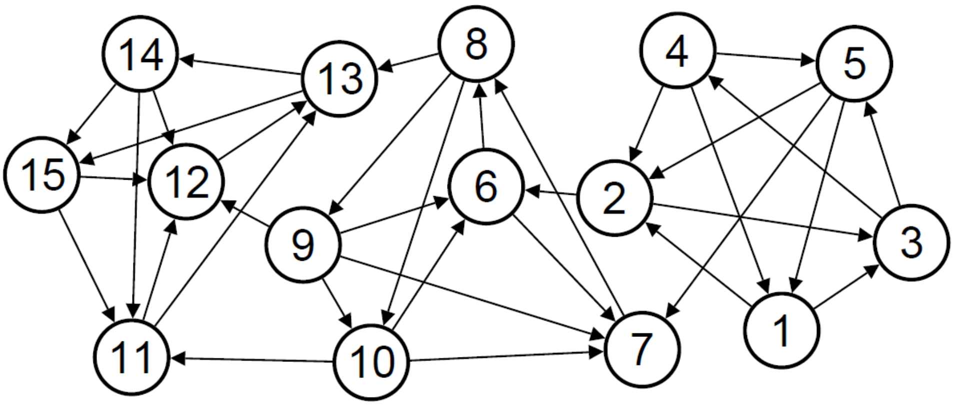


Figure 3. Test Graph 3[45]

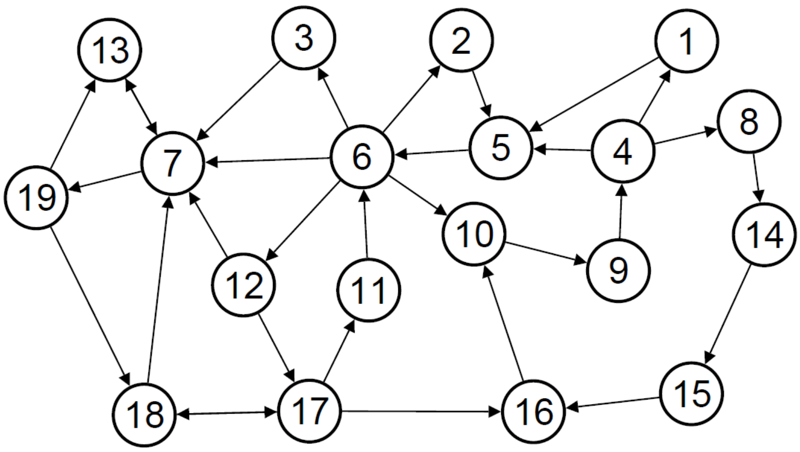


Figure 4. Test Graph 4[45]

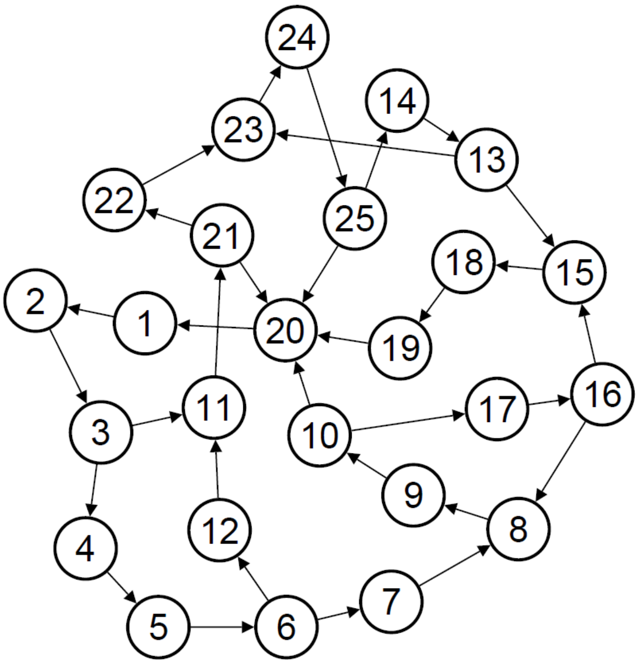


Figure 5. Test Graph 5[45]

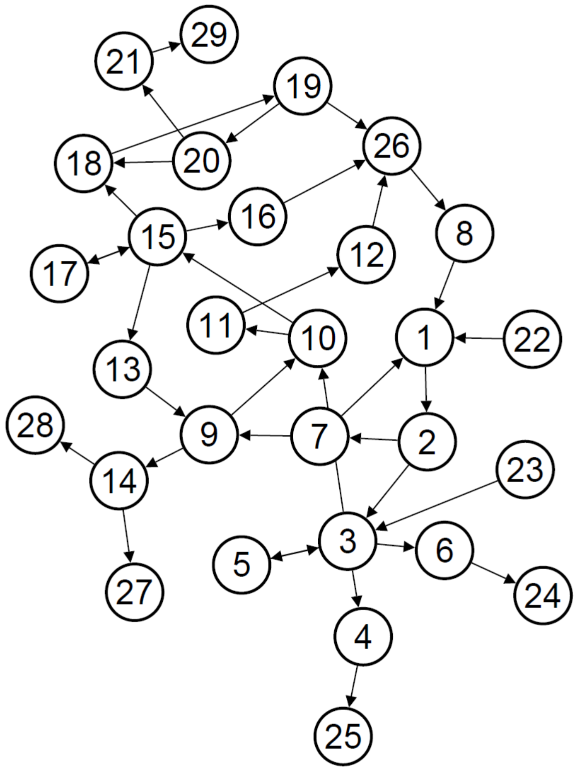


Figure 6. Test Graph 6[45]

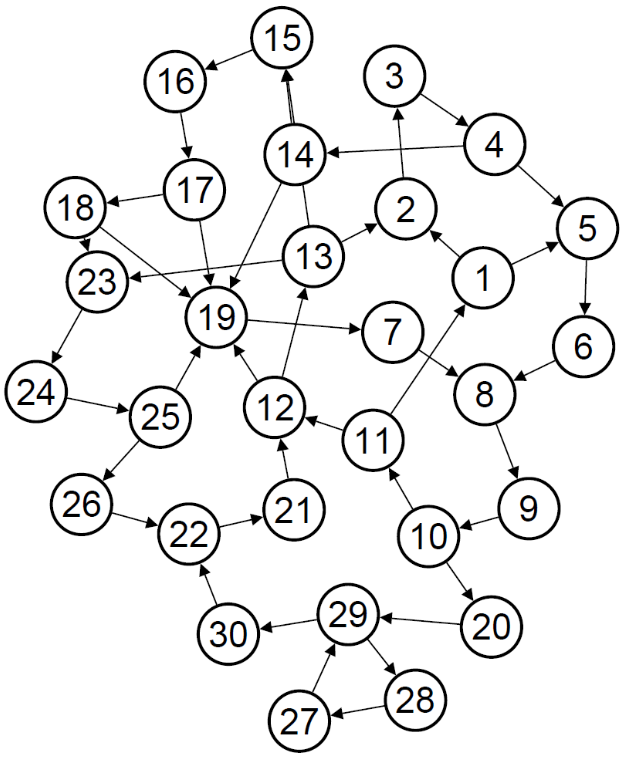


Figure 7. Test Graph 7[45]

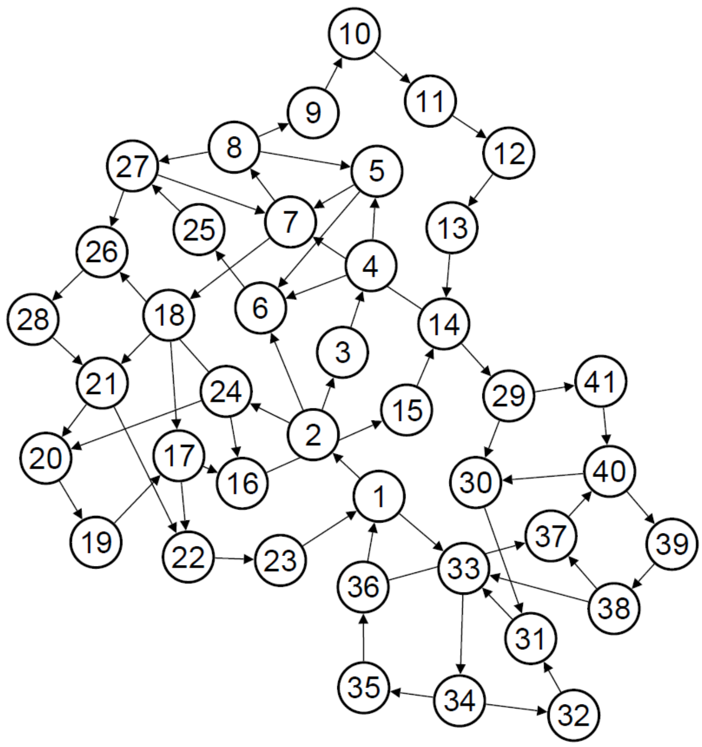


Figure 8. Test Graph 8[45]

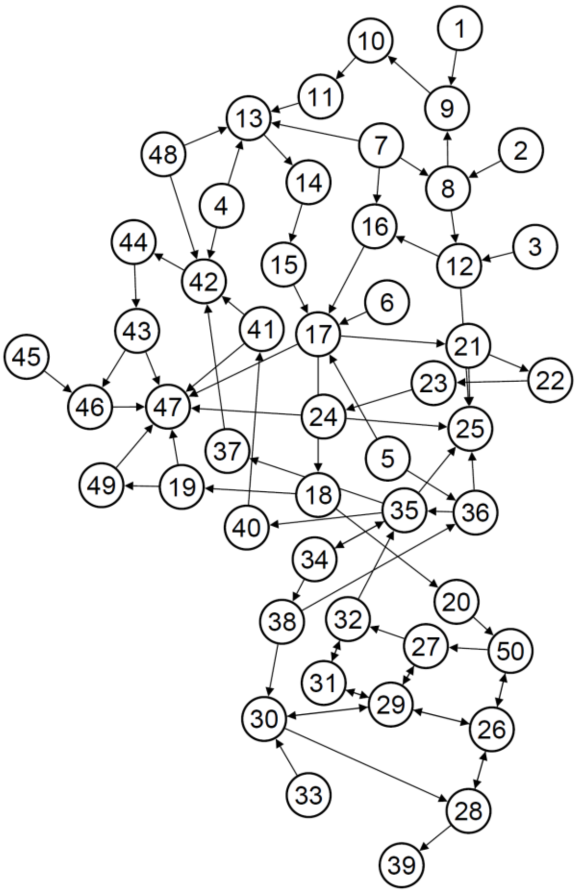


Figure 9. Test Graph 9 [45]

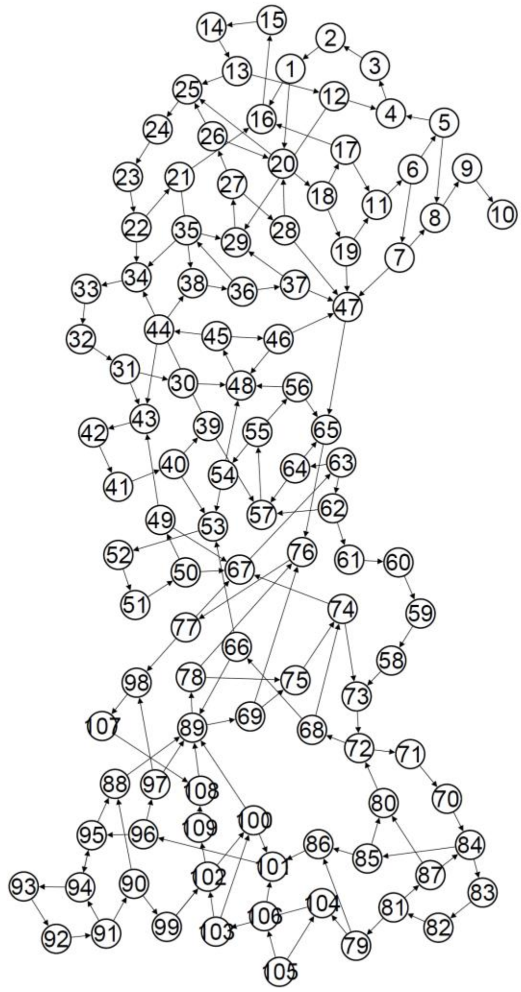


Figure 10. Test Graph 10[45]