Designing Algorithms for the Shortest Path Reconfiguration Problem Using Decision Diagram Operations

Shou Ooba, Jun Kawahara[a] and Shin-ichi Minato[b]
Graduate School of Informatics, Kyoto University, Japan

Keywords: Decision Diagrams, Reconfiguration Problem, Shortest Path, Graph Algorithm.

Abstract: This paper proposes decision diagram (DD)-based algorithms for the (edge-unweighted) shortest *s*-*t* path reconfiguration problem. In the problem, given a graph and two shortest *s*-*t* paths, the task is to decide whether one shortest path can be transformed into the other one by repeatedly applying the reconfiguration rule to the path, where the reconfiguration rule is to change one vertex of the path at a time while maintaining shortest *s*-*t* paths. We propose several DD-based algorithms for the problem and confirm their performance by computer experiments. We succeeded in finding a shortest reconfiguration sequence with length 961,012 in 629.0 seconds for some instance.

1 INTRODUCTION

A combinatorial reconfiguration problem is a problem in which two solutions of a combinatorial optimization problem are given, and one solution is transformed into the other one according to a given rule. The task is to determine whether it can be transformed from one to the other while satisfying that the intermediate states are also feasible. Taking the token jumping model of the independent set reconfiguration problem (Ito et al., 2011) as an example, given a graph G and two independent sets I_s and I_t of G, the rule is to arbitrarily remove one element from the independent set and arbitrarily add one element to it at the same time, and to determine whether it is possible to transform I_s to I_t, while maintaining the generated sets being independent.

Combinatorial reconfiguration is a young research area (Ito et al., 2011; Nishimura, 2018). According to the web survey[1] on combinatorial reconfiguration, 69 papers on combinatorial reconfiguration have been published, including the independent set reconfiguration problem (Ito et al., 2011) and the graph coloring reconfiguration problem (Bonsma and Cereceda, 2009). Reconfiguration problems can be viewed as problems of changing the configuration of a social system without stopping it, such as changing the switch configuration of a power distribution network to prevent power outages. However, many reconfiguration problems, including the independent set reconfiguration, are known to be PSPACE-complete in general (Ito et al., 2011; Nishimura, 2018), and it may be hard to solve such problems in practically acceptable time.

Under such circumstances, CoRe Challenge 2022 (Soh et al., 2022), a programming competition for reconfiguration, was held. Some 369 instances of the independent set reconfiguration problem were provided and 10 solvers participated, including SAT-based, answer set programming-based (Yamada et al., 2023), model checking-based (Toda et al., 2023), and AI planning-based solvers (Christen et al., 2023). Some of the solvers successfully solved instances consisting of graphs with several hundreds of vertices.

There exists a data structure called the Zero-suppressed binary Decision Diagram (ZDD) (Minato, 1993) that efficiently represents a family of sets; a solver for the reconfiguration problem using ZDDs also participated in the competition (Ito et al., 2023). This solver is particularly good at instances where the shortest reconfiguration length is very long, and has successfully solved instances that no other solver has been able to solve. According to the paper (Ito et al., 2023), the ZDD solver succeeded in obtaining the shortest reconfiguration sequence for an instance consisting of a graph with 247 vertices, 1,578 edges and shortest reconfiguration length of 5,767,157. As long as the family of objects that we want to trans-

[a] https://orcid.org/0000-0001-7208-044X
[b] https://orcid.org/0000-0002-1397-1020
[1] https://www.ecei.tohoku.ac.jp/alg/coresurvey/

Ooba, S., Kawahara, J. and Minato, S.
Designing Algorithms for the Shortest Path Reconfiguration Problem Using Decision Diagram Operations.
DOI: 10.5220/0012379900003636
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 641-648
ISBN: 978-989-758-680-4; ISSN: 2184-433X
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

form is represented as a ZDD, the ZDD solver can compute the shortest reconfiguration sequence under the token jumping model; that is, the rule of removing one element and adding one element at the same time. Their paper (Ito et al., 2023) designed a ZDD-based algorithm that works only under the token jumping model, using so-called ZDD operations. However, there are some reconfiguration rules (models) that are more complex than it. If we would like to apply the ZDD-based technique to reconfiguration problems under such rules, we need to design other ZDD operations.

In this paper, we treat the shortest *s*-*t* path reconfiguration problem, where each edge is unweighted; that is, all the edges have unit length one. In the problem, given a graph and two shortest *s*-*t* paths, the task is to decide whether one shortest path can be transformed into the other one by repeatedly applying the reconfiguration rule to the path, where the reconfiguration rule is to change one vertex of the path at a time while maintaining shortest *s*-*t* paths. We propose several ZDD-based algorithms for the problem by designing ZDD operations. We conducted computer experiments to compare them, and succeeded in finding a shortest reconfiguration sequence with length 961,012 in 629.0 seconds for some instance.

The paper is organized as follows. In Sec. 2, preliminaries for reconfiguration problems, ZDDs, and a ZDD-based framework for solving reconfiguration problems are provided. Sec. 3 describes ZDD-based algorithms. The results of computer experiments are shown in Sec. 4. We conclude the paper in Sec. 5.

2 PRELIMINARIES

We denote by $U + a$ the addition to a set U, i.e., $U \cup \{a\}$. We also denote by $U - a$ and $U - U'$ the removal of an element a and all the elements in U' from a set U, i.e., $U \setminus \{a\}$ and $U \setminus U'$, respectively. For a graph $G = (V, E)$, we denote by $uv \in E$ an edge whose endpoints are $u, v \in V$.

2.1 Reconfiguration Problems

Let $\mathcal{F}$ be the set of feasible solutions of a combinatorial problem. By fixing some reconfiguration rule, the adjacency relation between two solutions in $\mathcal{F}$ is determined. Given $\mathcal{F}$ and two solutions $S, T \in \mathcal{F}$, the reconfiguration problem of $(\mathcal{F}, S, T)$ asks us to decide whether there is a reconfiguration sequence $S = F_0, F_1, F_2, \ldots, F_\ell = T$ such that $F_i \in \mathcal{F}$ holds for $i = 0, 1, \ldots, \ell$ and F_i and F_{i+1} are adjacent for $i = 0, 1, \ldots, \ell - 1$ under the given adjacency relation.

Let us consider the token jumping model of the independent set reconfiguration as an example. In the token jumping model, two independent sets I and I' are adjacent if $|I \setminus I'| = |I' \setminus I| = 1$. Given a graph G and two independent sets S and T of G, the independent reconfiguration problem asks to decide whether there is a reconfiguration sequence $S = I_0, I_1, I_2, \ldots, I_\ell = T$ such that $I_0, I_1, \ldots, I_\ell$ all are independent and I_i and I_{i+1} are adjacent for all $i = 0, 1, \ldots, \ell - 1$.

2.2 ZDD

ZDD is a data structure for compactly representing the family of sets. We briefly explain the characteristics of ZDD (Minato, 1993). In this subsection, assume that the universe set of families represented by ZDDs is $U = \{x_1, \ldots, x_n\}$ and the elements are ordered as $x_1 < \cdots < x_n$. A ZDD $\mathcal{Z}$ is a directed acyclic graph that has two nodes with outdegree 0, called 0- and 1-*terminals*, and one node with indegree 0, called *root*, and denoted by $\mathsf{root}(\mathcal{Z})$. The outdegree of each non-terminal nodes is two; the arcs are called 0- and 1-arc. Each non-terminal node ν has a label, one of $x_1, \ldots, x_{n-1}$ or x_n, which we denote by $\mathsf{label}(\nu)$. For non-terminal nodes ν and ν', if ν points at ν' by its 0- or 1-arc, $\mathsf{label}(\nu) < \mathsf{label}(\nu')$ must hold.

We interpret that a ZDD represents the family of sets in the following manner: We consider a route from the root to 1-terminal, say $\nu_1, a_1, \nu_2, a_2, \ldots, \nu_k, a_k, \nu_{k+1}$, where ν_i is a non-terminal node for $i = 1, \ldots, k$, a_i is a 0- or 1-arc of ν_i and ν_{k+1} is 1-terminal. Then, the route represents the set $\{\mathsf{label}(\nu_i) \mid a_i \text{ is 1-arc}\}$; that is, collecting the labels of the nodes each of whose outgoing arc on the route is 1-arc. A ZDD represents the family of sets each of which is represented by a route from the root to 1-terminal. For a ZDD $\mathcal{Z}$, the family of sets represented by $\mathcal{Z}$ is denoted by $\mathcal{S}(\mathcal{Z})$.

A ZDD has the following recursive structure. Given a ZDD $\mathcal{Z}$, consider the root node ν of $\mathcal{Z}$ and the nodes pointed by 0- and 1-arcs of ν, which we denote by ν_0 and ν_1, respectively. Then, for $i = 0, 1$, we consider a directed graph consisting of the nodes and arcs reachable from ν_i. The graph can be regarded as a ZDD whose root is ν_i and we denote it by $\mathsf{child}_i(\mathcal{Z})$. The ZDDs $\mathsf{child}_0(\mathcal{Z})$ and $\mathsf{child}_1(\mathcal{Z})$ represent the family of sets in $\mathcal{S}(\mathcal{Z})$ not including the label of $\mathsf{root}(\mathcal{Z})$ and the family of sets in $\mathcal{S}(\mathcal{Z})$ including the label of $\mathsf{root}(\mathcal{Z})$, respectively.

We use the following operations, which are efficiently performed by recursive algorithms. See the book (Knuth, 2011) for detail. Let $\mathcal{F}, \mathcal{G}$ be ZDDs, and a be an element in U.

- $\mathcal{F} \cup \mathcal{G}$, $\mathcal{F} \cap \mathcal{G}$, and $\mathcal{F} \setminus \mathcal{G}$ are ZDDs representing the union, intersection, and subtraction of $\mathcal{S}(\mathcal{F})$ and $\mathcal{S}(\mathcal{G})$, respectively; that is, $\{F \mid F \in \mathcal{S}(\mathcal{F}) \text{ or } F \in \mathcal{S}(\mathcal{G})\}$, $\{F \mid F \in \mathcal{S}(\mathcal{F}) \text{ and } F \in \mathcal{S}(\mathcal{G})\}$, and $\{F \mid F \in \mathcal{S}(\mathcal{F}) \text{ and } F \notin \mathcal{S}(\mathcal{G})\}$ (Bryant, 1986; Minato, 1993; Knuth, 2011).
- Operation $\mathsf{subset1}(\mathcal{F}, a)$ is a ZDD representing the family of sets obtained by extracting the sets containing a from $\mathcal{S}(\mathcal{F})$ and removing a from the sets; that is, $\{S - a \mid S \in \mathcal{S}(\mathcal{F}), S \ni a\}$ (Minato, 1993).
- Operation $\mathsf{multip}(\mathcal{F}, a)$ is a ZDD representing the family of sets obtained by adding a to each set in $\mathcal{S}(\mathcal{F})$; that is, $\{S + a \mid S \in \mathcal{S}(\mathcal{F})\}$ (Okuno et al., 1998).
- Operation $\mathsf{restrict}(\mathcal{F}, \mathcal{G})$ is a ZDD representing the family of sets obtained by extracting the sets from $\mathcal{S}(\mathcal{F})$ each of which completely subsumes a set in $\mathcal{S}(\mathcal{G})$; that is, $\{F \in \mathcal{F} \mid G \in \mathcal{G}, G \subseteq F\}$ (Okuno et al., 1998).

2.3 ZDD-Based Algorithmic Framework for Reconfiguration Problems

Ito et al. (Ito et al., 2023) proposed a ZDD-based algorithmic framework for solving reconfiguration problems. We explain the framework taking the token jumping model of the independent set reconfiguration as an example. First, we construct a ZDD representing the family of solutions. In the case of the independent set reconfiguration, we construct the ZDD, say $\mathcal{Z}_{\mathrm{sol}}$, representing the family of all the independent sets of a given graph. This can be performed by an existing algorithm (Hayase et al., 1995). Then, for a ZDD $\mathcal{Z}$, let $\mathsf{swap}_I(\mathcal{Z})$ be the ZDD representing

$$\{U + v - v' \mid U \in \mathcal{S}(\mathcal{Z}), v \in I \setminus U, v' \in U\}.$$

This is the family of sets obtained by applying the reconfiguration rule (removing one element and adding another element) to each set in $\mathcal{S}(\mathcal{Z})$ and collecting all the possible resulting sets. Let $\mathcal{Z}^i$ be the family of sets obtained by applying the reconfiguration rule to S, i times, where $\mathcal{Z}^0$ is the ZDD representing $\{S\}$. $\mathcal{Z}^i$ can be computed by $\mathcal{Z}^i = \mathsf{swap}_I(\mathcal{Z}^{i-1}) \cap \mathcal{Z}_{\mathrm{sol}}$. Since $\mathcal{S}(\mathsf{swap}_I(\mathcal{Z}^{i-1}))$ includes not independent sets, we exclude them by "$\cap\, \mathcal{Z}_{\mathrm{sol}}$," which can be performed by a ZDD operation described in Sec. 2.2. By computing $\mathcal{Z}^i$ for $i = 1, 2, \ldots$, and checking whether $T \in \mathcal{S}(\mathcal{Z}^i)$, we can find a shortest reconfiguration sequence from S to T. If $\mathcal{S}(\mathcal{Z}^i) = \emptyset$ for some i, there is no reconfiguration sequence.

Instead explaining the construction of $\mathsf{swap}_I(\mathcal{Z})$, we explain how to compute $\mathsf{remove}(\mathcal{Z})$, a ZDD for

$$\{U - v \mid U \in \mathcal{S}(\mathcal{Z}), v \in U\}$$

because $\mathsf{remove}(\mathcal{Z})$ is easier than $\mathsf{swap}_I(\mathcal{Z})$ and $\mathsf{swap}_I(\mathcal{Z})$ can be computed in similar to the way for $\mathsf{remove}(\mathcal{Z})$. The algorithm uses the recursive structure of ZDDs. Let $x = \mathsf{root}(\mathcal{Z})$. We divide $\mathcal{S}(\mathsf{remove}(\mathcal{Z}))$ into two groups: (i) sets including x and (ii) sets not including x. Group (i) can be obtained by collecting the sets obtained by removing some element other than x from each set including x in $\mathcal{S}(\mathcal{Z})$. The ZDD representing the family of group (i) can be constructed by recursively calling $\mathsf{remove}(\mathcal{F}_1)$, where $\mathcal{F}_1 = \mathsf{child}_1(\mathcal{Z})$, because $\mathsf{child}_1(\mathcal{Z})$ is the family of sets containing x (see the description of the recursive structure in Sec. 2.2). Group (ii) is further divided into two subgroups: (ii-1) the sets obtained by removing some element from each set not including x in $\mathcal{S}(\mathcal{Z})$, and (ii-2) the sets obtained by removing x from each set including x in $\mathcal{S}(\mathcal{Z})$. The ZDD for (ii-1) is constructed by recursively calling $\mathsf{remove}(\mathsf{child}_0(\mathcal{Z}))$, and that for (ii-2) is exactly $\mathsf{child}_1(\mathcal{Z})$. The ZDD for (ii) is the union of the ZDDs (ii-1) and (ii-2). Let $\mathcal{F}_0 = \mathsf{remove}(\mathsf{child}_0(\mathcal{Z})) \cup \mathsf{child}_1(\mathcal{Z})$. Therefore, the ZDD for $\mathsf{remove}(\mathcal{Z})$ is constructed as follows: First, we construct $\mathcal{F}_0$ and $\mathcal{F}_1$ by recursive calls. Then, we create a node labeled x and make its 0-arc and 1-arc point at the roots of $\mathcal{F}_0$ and $\mathcal{F}_1$, respectively.

The add_I operation is defined by the ZDD representing

$$\{U + v \mid U \in \mathcal{S}(\mathcal{Z}), v \notin U, v \in I\},$$

which can be computed similarly. We need index I because we need to specify what we add to a set. The swap_I operation can also be performed similarly. Refer to the paper (Ito et al., 2023) for detail.

3 SHORTEST *s-t* PATH RECONFIGURATION PROBLEM

We begin with the definition of the shortest s-t path reconfiguration problem (Kamiński et al., 2011), where each edge is unweighted. Given an edge-unweighted undirected simple graph G and two s-t paths S and T of G, the problem asks us to decide whether there is a reconfiguration sequence $P_0 = S, P_1, P_2, \ldots, P_\ell = T$, where P_i is an s-t path for $i = 0, \ldots, \ell$ and two s-t paths P_i and P_{i+1} have all common edges except two edges and all common vertices except one vertex. This problem is shown to be PSPACE-complete in general, but can be solved in polynomial-time if the input graph G is a planar graph (Kamiński et al., 2011).

To apply the ZDD-based framework to the shortest s-t path reconfiguration, we first represent the solution space as a ZDD, and then we design recursive ZDD operation(s) to obtain $\mathcal{Z}^i$ from Z^{i-1}, which we define in Sec 2.3.

Let us consider what variables of ZDDs are. A path on a graph is uniquely specified by its edge set. Therefore, we can represent the solution space by the family of edge sets each of which consists of a path. Our first direction is to make the edges of G the variables of ZDDs, which we call *edge variables*.

We can consider another direction of ZDD variables by using the characteristics of shortest paths. Once the input graph G and the input vertices s and t are fixed, the shortest distance from s to each vertex can be easily determined by ordinary breadth first search. We partition V into $V_0, \ldots, V_{n-1}$, where V_i is the set of vertices to which the length of a shortest path from s is i for $i = 0, \ldots, d$ (recall that we assume that the weights of all the edges of G are 1). Then, any s-t shortest path P can be uniquely specified by $v_0 = s, v_1, \ldots, v_{d-1}, v_d = t$, where d is the length of any shortest s-t path, and $v_i \in V_i$ for $i = 0, \ldots, d$. This implies that the set $\{v_0, v_1, \ldots, v_d\}$ can uniquely represent an s-t path. Our second direction is to make the vertices of G the variables of ZDDs, which we call *vertex variables*.

In the setting of edge variables, we can use frontier-based search for representing various objects such as s-t paths, cycles, spanning trees, matchings, as ZDDs (Kawahara et al., 2017). Therefore, the edge variable direction can solve various reconfiguration problems. On the other hand, the vertex variable direction highly depends on the structure of shortest s-t paths, and thus it is specialized to solve the shortest s-t path reconfiguration problem. As we will show in the experiment section, the vertex variable direction is faster than the edge variable direction for almost all instances, although the results are omitted due to the space limitation. The rest of the section is devoted to solving the shortest s-t path reconfiguration using edge variable ZDDs and vertex variable ZDDs.

3.1 Algorithms on Edge Variables

Kawahara et al. (Kawahara et al., 2017) proposed an algorithm to construct a ZDD representing the family of all the s-t paths of a given graph G and given vertices s, t of G, where ZDD variables are edge variables, which we denote by $\mathcal{Z}_{\text{path}}$. It is easy to construct a ZDD representing the family of all the sets whose cardinality is exactly k, which we denote by $\mathcal{Z}_{=k}$ (Knuth, 2011). Therefore, by computing $\mathcal{Z}_{\text{path}} \cap \mathcal{Z}_{=d}$, we obtain the solution space ZDD $\mathcal{Z}_{\text{sol}}$, which represents the family of all the s-t shortest paths of G, where '$\cap$' is the intersection operation of ZDDs (Bryant, 1986), described in Sec. 2.2.

Next, we show how to construct the ZDD Z^i from Z^{i-1}. First, we design a naive algorithm. We consider a cycle x, u, y, v, x of G with length four. We extract all the sets including xu and uy from $\mathcal{S}(Z^{i-1})$, remove xu and uy from each extracted set, and add xv and vy to each of them. Each obtained edge set consists of a shortest s-t path transformed from a path in $\mathcal{S}(\mathcal{Z}^{i-1})$. We conduct the process for all cycles with length four and take the union of all of them. Thus, we obtain

$$\mathcal{Z}^i = \bigcup_{xu,uy,xv,vy \in E} \mathsf{multip}(\mathsf{multip}(\mathsf{subset1}(\mathsf{subset1}(\mathcal{Z}^{i-1}, xu), uy), xv), vy),$$

where multip and subset1 are described in Sec. 2.2. Note that for a ZDD $\mathcal{Z}$, $\mathsf{subset1}(\mathcal{Z}, a)$ conducts both the extraction of the sets containing a from $\mathcal{S}(\mathcal{Z})$ and removing a. The symbol '$\cup$' is the union operation of ZDDs, described in Sec. 2.2. Since the above equation computes the union $\mathrm{O}(n^4)$ times, the computation is not efficient. In the following, we propose five algorithms to compute it. Their efficiency will be compared by computer experiments in Sec. 4.

Remove-Add Algorithm. It is natural to consider the use of $\mathsf{remove}(\mathcal{Z})$ and $\mathsf{add}_I(\mathcal{Z})$, which removes and adds one arbitrary element from and to $\mathcal{S}(\mathcal{Z})$, respectively. In the setting of edge variables, to transform a shortest s-t path, we remove two edges and add another two edges. One might think that if we remove arbitrary two edges by calling remove twice and add arbitrary two edges by calling add twice, some generated results would not consist of a (shortest) path. However, such not (shortest) paths can be eliminated by the intersection operation of $\mathcal{Z}_{\text{sol}}$ (the ZDD for the family of edge sets consisting of shortest s-t paths). Therefore, we have

$$\mathcal{Z}^i = \mathsf{add}_E(\mathsf{add}_E(\mathsf{remove}(\mathsf{remove}(\mathcal{Z}^{i-1})))) \cap \mathcal{Z}_{\text{sol}}.$$

Moreover, we can design new operations that conduct remove k times at once, and that conduct add_I k times at once:

$$\mathsf{remove}(\mathcal{F}, k) = \{F - X \mid X \subseteq F \in \mathcal{F}, |X| = k\},$$
$$\mathsf{add}_I(\mathcal{F}, k) = \{F \cup X \mid F \in \mathcal{F}, X \cap F = \emptyset, X \subseteq I, |X| = k\}.$$

Algorithms that compute $\mathsf{remove}(\mathcal{F}, k)$ and $\mathsf{add}_I(\mathcal{F}, k)$ will be designed later. Using them, we obtain

$$\mathcal{Z}^i = \mathsf{add}_E(\mathsf{remove}(\mathcal{Z}^{i-1}, 2), 2) \cap \mathcal{Z}_{\text{sol}},$$

which we call the *remove-add* algorithm. Note that $\mathcal{S}(\mathcal{Z}^i)$ includes paths each of which is completely equal to a path in $\mathcal{S}(\mathcal{Z}^{i-1})$, which violates our reconfiguration rule, but it does not affect the decision of the existence of a reconfiguration sequence and computing a shortest reconfiguration sequence.

Remove-Restrict and Remove-Restrict-k Algorithms. In the remove-add algorithm, we add arbitrary two edges after removing two edges. It generates a huge number of edge sets consisting of non-paths (although they are compactly represented as a ZDD). The algorithm proposed here adds two edges so that the resulting edge sets consist of paths. This can be performed using the restrict operation, described in Sec. 2.2, as follows:

$$\mathcal{Z}^i = \mathsf{restrict}(\mathcal{Z}_{\mathrm{sol}}, \mathsf{remove}(\mathcal{Z}^{i-1}, 2)),$$

which we call the *remove-restrict* algorithm. Recall that the definition of restrict. Each edge set $E' \in \mathcal{S}(\mathsf{restrict}(\mathcal{Z}_{\mathrm{sol}}, \mathsf{remove}(\mathcal{Z}^{i-1}, 2)))$ is a member of $\mathcal{S}(\mathcal{Z}_{\mathrm{sol}})$ such that there exists an edge set $X \in \mathcal{S}(\mathsf{remove}(\mathcal{Z}^{i-1}, 2))$ satisfying $X \subseteq E'$. This means that we first remove two edges from $\mathcal{S}(\mathcal{Z}^{i-1})$ (the resulting set is X), and then add edges to X (the resulting set is E') so that E' is in $\mathcal{S}(\mathcal{Z}_{\mathrm{sol}})$; i.e., E' consists of a shortest *s*-*t* path.

We know the size of E' is larger than that of X by two. By using the fact, we expect to accelerate the computation. We propose a new operation:

$$\mathsf{restrict}(\mathcal{F}, \mathcal{G}, k) = \{F \in \mathcal{F} \mid G \in \mathcal{G}, \\ G \subseteq F, |F - G| = k\}.$$

The computation of it and the reason this makes the computation faster will be shown later. By using the extended restrict operation, we write

$$\mathcal{Z}^i = \mathsf{restrict}(\mathcal{Z}_{\mathrm{sol}}, \mathsf{remove}(\mathcal{Z}^{i-1}, 2), 2),$$

which we call the *remove-restrict-k* algorithm.

Dist Algorithm. In the remove-restrict and remove-restrict-k algorithms, we first call $\mathsf{remove}(\mathcal{Z}^{i-1}, 2)$. Here, we consider to compute remove and restrict at once. We propose a new operation, called dist:

$$\mathsf{dist}(\mathcal{F}, \mathcal{G}, k) = \{F \in \mathcal{F} \mid G \in \mathcal{G}, |F \Delta G| = k\},$$

where $F \Delta G = (F - G) \cup (G - F)$. This operation extracts the sets F from $\mathcal{S}(\mathcal{F})$ such that the hamming distance between F and a set G in $\mathcal{S}(\mathcal{G})$ is k. Using the operation, we compute $\mathcal{Z}^i$ as follows:

$$\mathcal{Z}^i = \mathsf{dist}(\mathcal{Z}_{\mathrm{sol}}, \mathcal{Z}^{i-1}, 4),$$

which we call the *dist* algorithm.

Delta Algorithm. Knuth (Knuth, 2011) proposed in his book "The art of computer programming" a ZDD operation, called delta:

$$\mathsf{delta}(\mathcal{F}, \mathcal{G}) = \{F \Delta G \mid F \in \mathcal{F}, G \in \mathcal{G}\}.$$

Using delta, we can compute $\mathcal{Z}^i$. First, we construct the ZDD $\mathcal{C}$ representing the family of edge sets each of which consists of a cycle with length four by frontier-based search (Kawahara et al., 2017). Then, we compute

$$\mathcal{Z}^i = \mathcal{Z}_{\mathrm{sol}} \cap \mathsf{delta}(\mathcal{Z}^{i-1}, \mathcal{C}),$$

which we call the *delta* algorithm.

3.2 Algorithms on Vertex Variables

First, we construct the solution space ZDD $\mathcal{Z}_{\mathrm{sol}}$ when the ZDD variables are the vertices of the input graph $G = (V, E)$. For $i = 0, \ldots, d$, exactly one vertex in V_i is included in a shortest *s*-*t* path. Let $\mathcal{V}_i$ be the ZDD for the family of sets containing exactly one vertex in V_i and arbitrary vertices in $V - V_i$; that is,

$$\mathcal{S}(\mathcal{V}_i) = \{X \subseteq V_i \mid |X| = 1\} \cup \{X \mid X \subseteq V - V_i\}.$$

Two vertices that are not adjacent on G cannot be included in a shortest *s*-*t* path. Let $\mathcal{X}_{uv}$ be the ZDD for the family of sets containing both u and v; that is, $\mathcal{S}(\mathcal{X}_{uv}) = \{X \subseteq V \mid X \supseteq \{u, v\}\}$.

$$\mathcal{Z}_{\mathrm{sol}} = \left(\bigcap_{i=0,\ldots,d} \mathcal{V}_i \right) \setminus \left(\bigcup_{u,v \in V, uv \notin E} \mathcal{X}_{uv} \right).$$

The ZDDs $\mathcal{V}_i$ and $\mathcal{X}_{uv}$ can be easily constructed and the above equation can be computed by ZDD operations. Therefore, we can construct the ZDD $\mathcal{Z}_{\mathrm{sol}}$.

Next, we describe how to construct $\mathcal{Z}^i$ from $\mathcal{Z}^{i-1}$. This is similar to the edge variable algorithms. The main difference of them is to remove and add one variable in the setting of vertex variables. We propose the following algorithms:

- Naive algorithm:
$$\mathcal{Z}^i = \left(\bigcup_{u,v \in V} \mathsf{multip}(\mathsf{subset1}(\mathcal{Z}^{i-1}, u), v) \right) \cap \mathcal{Z}_{\mathrm{sol}},$$
- Remove-add algorithm:
$$\mathcal{Z}^i = \mathsf{add}_V(\mathsf{remove}(\mathcal{Z}^{i-1})) \cap \mathcal{Z}_{\mathrm{sol}},$$
- Remove-restrict algorithm:
$$\mathcal{Z}^i = \mathsf{restrict}(\mathcal{Z}_{\mathrm{sol}}, \mathsf{remove}(\mathcal{Z}^{i-1}, 1)),$$
- Remove-restrict-k algorithm:
$$\mathcal{Z}^i = \mathsf{restrict}(\mathcal{Z}_{\mathrm{sol}}, \mathsf{remove}(\mathcal{Z}^{i-1}, 1), 1),$$
- Dist algorithm: $\mathcal{Z}^i = \mathsf{dist}(\mathcal{Z}_{\mathrm{sol}}, \mathcal{Z}^{i-1}, 2)$,
- Delta algorithm: $\mathcal{Z}^i = \mathcal{Z}_{\mathrm{sol}} \cap \mathsf{delta}(\mathcal{Z}^{i-1}, \mathcal{C}')$,

where $\mathcal{C}'$ is the ZDD for the family of all the subsets of V whose cardinality is two.

3.3 Recursive Operations

In this subsection, we show how to conduct the recursive operations described in the previous subsections. First, we explain $\mathsf{remove}(\mathcal{F},k) = \{F - X \mid X \subseteq F \in \mathcal{F}, |X| = k\}$. If $k = 0$, $\mathsf{remove}(\mathcal{F},0) = \mathcal{F}$. We consider the case of $k > 0$. Let $x = \mathsf{root}(\mathcal{F})$. We create a node labeled x and make its 0-arc and 1-arc point at the ZDDs $\mathcal{F}_0$ and $\mathcal{F}_1$, explained below, respectively.

$\mathcal{F}_0$ and $\mathcal{F}_1$ can be constructed by

$$\begin{aligned}\mathcal{F}_0 \leftarrow\ & \mathsf{remove}(\mathsf{child}_0(\mathcal{F}),k)\\ & \cup \mathsf{remove}(\mathsf{child}_1(\mathcal{F}),k-1),\\ \mathcal{F}_1 \leftarrow\ & \mathsf{remove}(\mathsf{child}_1(\mathcal{F}),k).\end{aligned}$$

The second term $\mathsf{remove}(\mathsf{child}_1(\mathcal{F}),k-1)$ in the first equation means that x is removed and $k-1$ elements other than x will be removed in the recursive call. We omit the explanation of the termination in the recursion.

The $\mathsf{add}_I(\mathcal{F},k) = \{F \cup X \mid F \in \mathcal{F}, X \cap F = \emptyset, X \subseteq I, |X| = k\}$ operation can be computed similarly. We show just $\mathcal{F}_0$ and $\mathcal{F}_1$ as follows: If $x \notin I$, we have

$$\begin{aligned}\mathcal{F}_0 &\leftarrow \mathsf{add}_I(\mathsf{child}_0(\mathcal{F}),k),\\ \mathcal{F}_1 &\leftarrow \mathsf{add}_I(\mathsf{child}_1(\mathcal{F}),k).\end{aligned}$$

If $x \in I$, we obtain

$$\begin{aligned}\mathcal{F}_0 \leftarrow\ & \mathsf{add}_{I-x}(\mathsf{child}_0(\mathcal{F}),k),\\ \mathcal{F}_1 \leftarrow\ & \mathsf{add}_{I-x}(\mathsf{child}_1(\mathcal{F}),k)\\ & \cup \mathsf{add}_{I-x}(\mathsf{child}_0(\mathcal{F}),k-1).\end{aligned}$$

Next, we consider

$$\begin{aligned}\mathsf{restrict}(\mathcal{F},\mathcal{G},k) = \{F \in \mathcal{F} \mid\ & G \in \mathcal{G},\\ & G \subseteq F, |F - G| = k\}.\end{aligned}$$

$\mathcal{F}_0$ and $\mathcal{F}_1$ can be obtained by

$$\begin{aligned}\mathcal{F}_0 \leftarrow\ & \mathsf{restrict}(\mathsf{child}_0(\mathcal{F}),\mathsf{child}_0(\mathcal{G}),k),\\ \mathcal{F}_1 \leftarrow\ & \mathsf{restrict}(\mathsf{child}_1(\mathcal{F}),\mathsf{child}_1(\mathcal{G}),k)\\ & \cup \mathsf{restrict}(\mathsf{child}_1(\mathcal{F}),\mathsf{child}_0(\mathcal{G}),k-1).\end{aligned}$$

Finally, we consider

$$\mathsf{dist}(\mathcal{F},\mathcal{G},k) = \{F \in \mathcal{F} \mid G \in \mathcal{G}, |F \Delta G| = k\}.$$

$\mathcal{F}_0$ and $\mathcal{F}_1$ can be obtained by

$$\begin{aligned}\mathcal{F}_0 \leftarrow\ & \mathsf{dist}(\mathsf{child}_0(\mathcal{F}),\mathsf{child}_0(\mathcal{G}),k)\\ & \cup \mathsf{dist}(\mathsf{child}_0(\mathcal{F}),\mathsf{child}_1(\mathcal{G}),k-1),\\ \mathcal{F}_1 \leftarrow\ & \mathsf{dist}(\mathsf{child}_1(\mathcal{F}),\mathsf{child}_1(\mathcal{G}),k)\\ & \cup \mathsf{dist}(\mathsf{child}_1(\mathcal{F}),\mathsf{child}_0(\mathcal{G}),k-1).\end{aligned}$$

4 COMPUTER EXPERIMENTS

We conducted computer experiments on $N \times N$ grid graph instances, the instances provided by the paper (Kamiński et al., 2011), and randomly generated instances. For grid graph instances, we implemented and measured a polynomial-time solution method for planar graphs as a comparison with the proposed method (Bonsma, 2017). The $N \times N$ grid graph instance, denoted by `grid`N, has N^2 vertices, $2N(N-1)$ edges, and a shortest reconfiguration sequence with length $(N-1)^2$. We set s and t to be the upper-left and lower-right corners of a grid graph. We set the start and goal shortest s-t paths to be the path consisting of the uppermost and rightmost edges via the upper right corner, and the path consisting of the leftmost and lowermost edges via the lower left corner, respectively. The instances provided by the paper (Kamiński et al., 2011) are used for proving that the shortest s-t reconfiguration is PSPACE-complete. Each of the instances has a parameter N, which we denote by `exp`N. It has $13N+2$ vertices, $36N-11$ edges, and a shortest reconfiguration sequence with length $11(2^N-1)$. As for the randomly generated instances, we generate random graphs by randomly adding vertices to $V_0, V_1, \ldots, V_d$ for a given integer d and randomly connecting vertices between V_i and V_{i+1}. We pick up the generated instances whose execution time was the fifth longest, denoted by `random5`, `random6`, `random9`, `random10`, `random11`. The features of the instances are shown in Table 1.

Table 1: Features of input graphs. The length of a shortest s-t path reconfiguration sequence is shown as ℓ.

Instance	$\lvert V\rvert$	$\lvert E\rvert$	ℓ
grid10x10	100	180	81
grid20x20	400	760	361
grid30x30	900	1740	841
grid40x40	1600	3120	1521
grid50x50	2500	4900	2401
grid60x60	3600	7080	3481
exp6	80	205	891
exp8	106	277	3696
exp10	132	349	14949
exp12	158	421	59994
exp14	184	493	240207
exp16	210	565	961092
random5	470	1887	113
random6	482	1938	111
random9	385	2971	38
random10	437	3423	34
random11	458	3676	41

Due to the memory limitation, we did not restore a shortest reconfiguration sequence, but measured the

Table 2: Results for vertex variable (in seconds). '$-$' means timeout (exceeding 1000 seconds).

Instance	Naive	Remove-add	Remove-restrict	Remove-restrict-k	Dist	Delta
grid10x10	0.1	0.1	<0.1	<0.1	<0.1	<0.1
grid20x20	67.4	4.5	1.3	0.4	0.5	1.7
grid30x30	$-$	47.5	14.7	5.0	6.5	17.0
grid40x40	$-$	208.3	69.3	37.4	55.7	89.3
grid50x50	$-$	571.4	241.7	173.9	323.1	331.8
grid60x60	$-$	$-$	684.2	509.4	$-$	945.5
exp6	0.6	0.3	0.2	0.1	0.1	0.1
exp8	6.9	2.1	2.0	0.3	0.4	0.7
exp10	48.3	12.3	11.9	2.6	2.1	4.1
exp12	323.1	66.1	61.4	14.7	12.1	22.8
exp14	$-$	314.6	300.7	68.6	60.0	105.4
exp16	$-$	$-$	$-$	351.8	291.3	500.1
random5	$-$	143.7	390.0	34.1	731.4	346.1
random6	$-$	19.6	45.4	7.2	189.5	35.9
random9	$-$	$-$	$-$	49.7	61.5	269.2
random10	$-$	$-$	$-$	323.6	777.6	$-$
random11	$-$	656.3	$-$	73.8	199.8	53.2

time taken to find the length of a shortest sequence. However, since the polynomial-time method for planar graphs cannot find the length of a shortest reconfiguration sequence, we measured the time taken to determine the reachability. The time required to embed a planar graph into a plane was not measured in the polynomial-time method for planar graphs. Due to the space limitation, we omit the experimental results for the edge variable method. Results for the vertex variable method and the polynomial-time algorithm (Bonsma, 2017) are shown in Tables 2, and 3, respectively. The values in the table represent the execution time in seconds, in particular, <0.1 means that the solution was completed in less than 0.1 second.

For the vertex variable setting, remove-restrict-k was the fastest for both instances. The remove-restrict was almost as fast as the dist algorithm in the grid graph instance, but in the exponential instance, remove-restrict was about three times slower in the largest case, and the delta algorithm was faster in the largest case. The remove-add algorithm was the fastest after rem+restrict and delta, and the slowest was the naive algorithm. Comparing the speed of the same type of algorithms between vertex and edge variables, the vertex variable was faster in both cases.

The polynomial-time algorithm for planar graphs are considerably faster than other algorithms. However, considering that it is not possible to find the shortest reconfiguration sequence length or the shortest reconfiguration sequence, and that it can only be used on planar graphs, the proposed algorithms without these restrictions can also be considered valuable.

The reason why the remove-restrict, remove-restrict-k, and dist algorithms are relatively fast is that they find $\mathcal{Z}^i$ by ZDD operations with $\mathcal{Z}_{\rm sol}$, which has a small number of nodes. The remove-add and delta algorithms are fast once the remove-add and delta algorithms generate a set that includes combinations that may not necessarily be a shortest *s*-*t* path and then intersect with $\mathcal{Z}_{\rm sol}$, which may result in many unnecessary (non shortest *s*-*t* path) combinations. However, for the delta algorithm, it is easy to reduce the number of unnecessary combinations by limiting the vertex variables to pairs of u, v in the same V_i or limiting the edge variables to four edges that form a cycle. The remove-add algorithm is considered to be slower because it takes into account a large number of unnecessary combinations, especially when k, the argument of add, is large.

In both cases, the remove-restrict-k algorithm is faster than the remove-restrict algorithm. This is because the $\mathsf{restrict}(\mathcal{F}, \mathcal{G}, k)$ operation is equivalent to the product set operation at $k = 0$. When $k > 0$, the child pointed to by the 1-arc of $\mathsf{restrict}(\mathcal{F}, \mathcal{G}, k)$ is $\mathsf{restrict}(\mathcal{F}_1, \mathcal{G}_1, k) \cup \mathsf{restrict}(\mathcal{F}_1, \mathcal{G}_0, k-1)$. In particular, when $k = 1$, the second term is replaced by the product set $\mathcal{F}_1 \sqcap \mathcal{G}_0$. On the other hand, the child pointed to by the 1-arc in the $\mathsf{restrict}(\mathcal{F}, \mathcal{G})$ operation is computed as $\mathsf{restrict}(\mathcal{F}_1, \mathcal{G}_1) \cup \mathsf{restrict}(\mathcal{F}_1, \mathcal{G}_0)$. In general, the $\mathsf{restrict}(\mathcal{F}, \mathcal{G}, k)$ operation is faster than the $\mathsf{restrict}(\mathcal{F}, \mathcal{G})$ operation because the product set operation is lighter than the $\mathsf{restrict}(\mathcal{F}, \mathcal{G})$ operation. However, the addition of k to the argument may reduce the cache hit ratio. In other words, the larger k is, the smaller the speedup effect is, and it should be noted that the $\mathsf{restrict}(\mathcal{F}, \mathcal{G})$ operation may be more efficient in some cases.

In conclusion, the remove-restrict-k and dist algorithms in the vertex variable seem to be the best for the shortest *s*-*t* path reconfiguration problem.

Table 3: Results for the polynomial time algorithm for planar graphs.

Instance	Time (sec.)
`grid20x20`	< 0.1
`grid40x40`	< 0.1
`grid60x60`	0.1
`grid80x80`	0.3
`grid100x100`	0.7

5 CONCLUSION

In this paper, we have proposed several algorithms for the shortest *s*-*t* path reconfiguration problem. All of them are based on DD operations. Computer experiments were conducted to compare these algorithms.

We believe that the value of this study is that it showed that ZDD operations can solve combinatorial reconfiguration problems under various rules. Future work includes considering what rules of combinatorial reconfiguration problems our method can be applied to.

ACKNOWLEDGEMENTS

This work was partially supported by JSPS KAKENHI Grant Numbers JP20H00605, JP20H05794, JP20H05964, and JP23H04383.

REFERENCES

Bonsma, P. and Cereceda, L. (2009). Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226.

Bonsma, P. S. (2017). Rerouting shortest paths in planar graphs. *Discrete Applied Mathematics*, 231:95–112.

Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

Christen, R., Eriksson, S., Katz, M., Muise, C., Petrov, A., Pommerening, F., Seipp, J., Sievers, S., and Speck, D. (2023). PARIS: planning algorithms for reconfiguring independent sets. In Gal, K., Nowé, A., Nalepa, G. J., Fairstein, R., and Radulescu, R., editors, *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 453–460. IOS Press.

Hayase, K., Sadakane, K., and Tani, S. (1995). Output-size sensitiveness of OBDD construction through maximal independent set problem. In Du, D.-Z. and Li, M., editors, *Computing and Combinatorics*, pages 229–234, Berlin, Heidelberg. Springer Berlin Heidelberg.

Ito, T., Demaine, E. D., Harvey, N. J. A., Papadimitriou, C. H., Sideri, M., Uehara, R., and Uno, Y. (2011). On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065.

Ito, T., Kawahara, J., Nakahata, Y., Soh, T., Suzuki, A., Teruyama, J., and Toda, T. (2023). ZDD-based algorithmic framework for solving shortest reconfiguration problems. In *20th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2023)*, pages 167–183.

Kamiński, M., Medvedev, P., and Milanič, M. (2011). Shortest paths between shortest paths. *Theoretical Computer Science*, 412(39):5205–5210.

Kawahara, J., Inoue, T., Iwashita, H., and Minato, S. (2017). Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E100-A(9):1773–1784.

Knuth, D. E. (2011). *The Art of Computer Programming, Volume 4A, Combinatorial Algorithms, Part 1.* Addison-Wesley Professional, 1st edition.

Minato, S. (1993). Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th ACM/IEEE design automation conference*, pages 272–277.

Nishimura, N. (2018). Introduction to reconfiguration. *Algorithms*, 11(4):52.

Okuno, H. G., Minato, S., and Isozaki, H. (1998). On the properties of combination set operations. *Information Processing Letters*, 66(4):195–199.

Soh, T., Okamoto, Y., and Ito, T. (2022). Core challenge 2022: Solver and graph descriptions. *CoRR*, abs/2208.02495.

Toda, T., Ito, T., Kawahara, J., Soh, T., Suzuki, A., and Teruyama, J. (2023). Solving reconfiguration problems of first-order expressible properties of graph vertices with boolean satisfiability. In *The 35th IEEE International Conference on Tools with Artificial Intelligence*, pages 294–302.

Yamada, Y., Banbara, M., Inoue, K., and Schaub, T. (2023). Recongo: Bounded combinatorial reconfiguration with answer set programming. In *Logics in Artificial Intelligence: 18th European Conference, JELIA 2023, Dresden, Germany, September 20–22, 2023, Proceedings*, pages 278–286, Berlin, Heidelberg. Springer-Verlag.