# Scalable Hard Instances for Independent Set Reconfiguration

**Takehide Soh**
Information Infrastructure and Digital Transformation Initiatives Headquaters, Kobe University, Japan

**Takumu Watanabe**
Graduate School of Information Sciences, Tohoku University, Japan

**Jun Kawahara**
Graduate School of Informatics, Kyoto University, Japan

**Akira Suzuki**
Graduate School of Information Sciences, Tohoku University, Japan

**Takehiro Ito**
Graduate School of Information Sciences, Tohoku University, Japan

## Abstract

The Token Jumping problem, also known as the independent set reconfiguration problem under the token jumping model, is defined as follows: Given a graph and two same-sized independent sets, determine whether one can be transformed into the other via a sequence of independent sets. Token Jumping has been extensively studied, mainly from the viewpoint of algorithmic theory, but its practical study has just begun. To develop a practically good solver, it is important to construct benchmark datasets that are scalable and hard. Here, "scalable" means the ability to change the scale of the instance while maintaining its characteristics by adjusting the given parameters; and "hard" means that the instance can become so difficult that it cannot be solved within a practical time frame by a solver. In this paper, we propose four types of instance series for Token Jumping. Our instance series is scalable in the sense that instance scales are controlled by the number of vertices. To establish their hardness, we focus on the numbers of transformation steps; our instance series requires exponential numbers of steps with respect to the number of vertices. Interestingly, three types of instance series are constructed by importing theories developed by algorithmic research. We experimentally evaluate the scalability and hardness of the proposed instance series, using the SAT solver and award-winning solvers of the international competition for Token Jumping.

**2012 ACM Subject Classification** Computing methodologies → Discrete space search

**Keywords and phrases** Combinatorial reconfiguration, Benckmark dataset, Graph Algorithm, PSPACE-complete

**Digital Object Identifier** 10.4230/LIPIcs.SEA.2024.26

**Supplementary Material** *Dataset (Benchmark Instances, Solvers and Logs)*: `https://github.com/TakehideSoh/Scalable-Hard-ISR`
archived at `swh:1:dir:c7100376b39143cf94f55312233283a4ced13440`

**Funding** *Takehide Soh*: JSPS KAKENHI Grant Numbers JP21K11828, JP22K11973, JP23K11047
*Jun Kawahara*: JSPS KAKENHI Grant Numbers JP20H00605, JP20H05794, JP23H04383
*Akira Suzuki*: JSPS KAKENHI Grant Numbers JP20K11666, JP20H05794
*Takehiro Ito*: JSPS KAKENHI Grant Numbers JP19K11814, JP20H05793



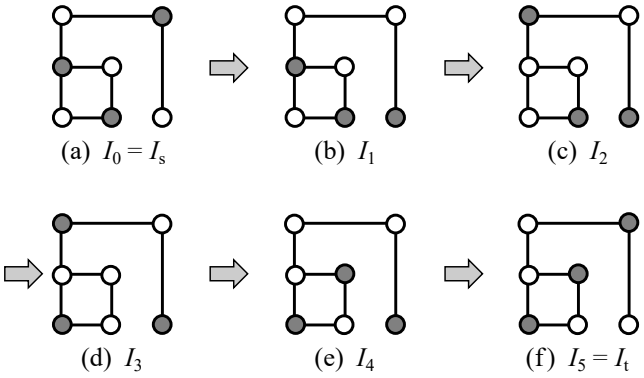



**Figure 1** A sequence $\langle I_0, I_1, \ldots, I_5 \rangle$ of independent sets between $I_s = I_0$ and $I_t = I_5$, where tokens (i.e., the vertices in an independent set) are colored with gray.

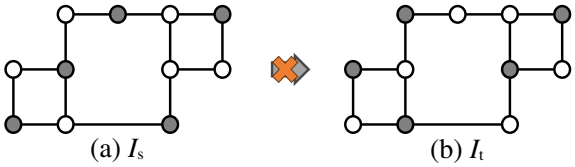



**Figure 2** An example of instances to which the answer is no.

## 1 Introduction

Combinatorial reconfiguration [22, 37, 21] is a family of problems that involve finding a step-by-step transformation between two given feasible solutions of a combinatorial (search) problem such that all intermediate solutions are also feasible and each step respects a prescribed reconfiguration rule. One of the most well-studied reconfiguration problems is the Token Jumping problem, which is also known as the Independent Set Reconfiguration problem under the token jumping model [28]. Recall that an *independent set* $I$ of a graph $G$ is a vertex subset of $G$ such that no two vertices are adjacent in $G$. Imagine that a *token* is placed on each vertex in $I$. Given two independent sets (token placements) of $G$, we are asked whether there is a transformation from one into the other by moving a single token at a time while always maintaining independent sets of $G$. (See Figure 1 for a yes-instance, and Figure 2 for a no-instance.) Token Jumping has been extensively studied from the viewpoint of algorithmic theory (see the surveys in [11, 37]), and is known to be PSPACE-complete [28]. Since NP $\subseteq$ PSPACE, this implies that there is a yes-instance such that even a shortest transformation requires a super-polynomial number of steps under the assumption of NP $\neq$ PSPACE; otherwise, we can use a transformation (of polynomial steps) as the witness. Indeed, examples that actually require exponential numbers of steps have been constructed for some reconfiguration problems [17, 27, 9]. In the literature, standard algorithmic techniques designed for NP-complete problems are rarely extended to reconfiguration problems, and significant algorithmic developments are required. We think this is one of the main reasons why Token Jumping and other combinatorial reconfiguration problems have attracted much attention from the theoretical algorithms research community.

Token Jumping has been used to prove the PSPACE-completeness of several other reconfiguration problems. In this sense, Token Jumping is a theoretically central problem and is thus important, similar to the SAT problem for NP-complete problems. Indeed, in

addition to the theoretical studies, recently, there has been a growing focus on practical studies for Token Jumping. In particular, initiated by the international solver competitions for Token Jumping held in 2022 [1] and 2023 [2], named CoRe Challenge, general-purpose solvers for Token Jumping are studied. Christen et al. [13] developed a solver that utilizes AI planning methods [16]. Yamada et al. [42] proposed a solver based on answer set programming [3]. Ito et al. [23] proposed a solver based on a data structure for a family of sets, called zero-suppressed binary decision diagram [32]. Notice that all the studies mentioned above are published in 2023 and 2024.

To accelerate these practical studies, it is crucial to construct benchmark datasets that are scalable and hard. Here, "scalable" means the ability to change the scale of the instance while maintaining its characteristics by adjusting the given parameters; and "hard" means that the instance can become so difficult that it cannot be solved within a practical time frame by a solver. To develop general-purpose solvers that do not rely on prior problem-specific knowledge, these properties are desirable for benchmark instances, as we have seen in history, e.g., for instances for SAT solvers. For example, the tower of Hanoi is easy to solve if we use problem-specific knowledge; however, interestingly, it is recognized and often used as a scalable and hard benchmark instance for general-purpose solvers such as SAT/CSP solvers (refer to CSP/SAT benchmark database [3,4] and the literature [36]).

It is important for solver developments to construct datasets reflecting the characteristic property of combinatorial reconfiguration, which gives a clear difference from NP-complete problems. To construct such datasets for Token Jumping, we focus on the numbers of required transformation steps. This is because, as mentioned above, one of the characteristic properties of Token Jumping is that there is a yes-instance such that even a shortest transformation requires a super-polynomial number of steps under the assumption of NP $\neq$ PSPACE. On the other hand, interestingly, Token Jumping is solvable in polynomial time if the number of transformation steps is bounded by a constant [35]. We thus think that the number of transformation steps gives a strong influence on the "hardness" of instances. One may think that it is not so difficult to construct datasets for Token Jumping, because there are various graphs that can be used from publicly available datasets, e.g., the DIMACS Challenge [25]. However, from the results of CoRe Challenge 2022 [5], despite the wide range of vertex numbers, from 11 to 10000, such instances do not need many transformation steps: instances with less than 10 steps occur in 80% of all DIMACS instances, and the longest step is only 112. Indeed, those instances were often easily solved by several solvers (including the SAT solver for an NP-complete problem).

In this paper, we propose four types of instance series for Token Jumping. Our instance series is scalable in the sense that instance scales are controlled by the number of vertices. We establish their hardness by ensuring that they require exponential numbers of steps with respect to the number of vertices. Interestingly, we constructed three of them by importing theories [17, 27, 9] developed by algorithmic research (and the remaining one is from scratch). We experimentally evaluate the scalability and hardness of the proposed instance series using the SAT solver and award-winning solvers of the international competition CoRe Challenge [39]. By comparing with randomly generated instances, we will confirm that the proposed instances are scalable and hard enough.

[1] https://core-challenge.github.io/2022/
[2] https://core-challenge.github.io/2023/
[3] https://www.xcsp.org/instances/
[4] https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html
[5] https://core-challenge.github.io/2022result/

The paper is organized as follows. Section 2 introduces terminology used in the following sections. Section 3 explains how to create the proposed Token Jumping instance series. Section 4 explains three kinds of Token Jumping solvers with distinct characteristics. Section 5 shows empirical experiments demonstrating the scalability and hardness of the proposed instance series. Section 6 concludes this paper.

## 2 Preliminaries

In this section, we formally define the Token Jumping problem and its related terminologies.

Let $G = (V, E)$ be an unweighted and undirected graph. A vertex subset $I \subseteq V$ is an *independent set* of $G$ if no two vertices in $I$ are adjacent in $G$. For a positive integer $k$, the *solution space* $\mathcal{S}_k(G)$ is a graph such that each node in $\mathcal{S}_k(G)$ is an independent set $I$ of $G$ with $|I| = k$ and two nodes are joined by an edge in $\mathcal{S}_k(G)$ if and only if the corresponding independent sets $I$ and $I'$ satisfy $|I \setminus I'| = |I' \setminus I| = 1$. A path in $\mathcal{S}_k(G)$ connecting two nodes $I$ and $I'$ is called a *reconfiguration sequence* between $I$ and $I'$. The *length* of the reconfiguration sequence is defined to be the number of edges in the path.

Given a graph $G$ and two independent sets $I_\mathrm{s}$ and $I_\mathrm{t}$ of $G$ such that $|I_\mathrm{s}| = |I_\mathrm{t}| = k$, *the Token Jumping problem* is to determine whether or not there exists a reconfiguration sequence between $I_\mathrm{s}$ and $I_\mathrm{t}$ in $\mathcal{S}_k(G)$. Note that Token Jumping is a decision problem, and hence we are not asked for an actual reconfiguration sequence as an output. Throughout this paper, we denote by $(G, I_\mathrm{s}, I_\mathrm{t})$ an instance of Token Jumping.

To evaluate the "hardness" of instances of Token Jumping, we introduce the notion of the "distance" of an instance. Let $(G, I_\mathrm{s}, I_\mathrm{t})$ be an instance of Token Jumping. Then, the *distance of* $(G, I_\mathrm{s}, I_\mathrm{t})$ is defined as the shortest length of any reconfiguration sequence between $I_\mathrm{s}$ and $I_\mathrm{t}$; it is defined as $+\infty$ if the answer to $(G, I_\mathrm{s}, I_\mathrm{t})$ is no. We say that an instance of Token Jumping is *hard* if its distance is exponential in the input size.

## 3 Scalable Hard Instances

In this section, we give four types of hard instance series for Token Jumping: three series are given in Section 3.1 by reductions from well-studied reconfiguration problems; and the last one in Section 3.2 is our original.

### 3.1 Hard instances based on reductions

In this subsection, we provide three types of hard instance series for Token Jumping, by introducing polynomial-time reductions from three well-studied reconfiguration problems. We will show (in Theorems 1, 2, and 3) that the distances of the corresponding instances of Token Jumping are at least those of source instances of the reductions. Then, because hard instance series are known for the three source reconfiguration problems of the reductions, we will obtain hard instance series for Token Jumping.

#### 3.1.1 SAT-based series

Gopalan et al. [17] introduced the *SAT Reconfiguration problem*, as follows: we are given two satisfying truth assignments for a CNF formula $\phi$, and are asked to determine whether or not we can transform one into the other by flipping a truth assignment of a single variable at a time so that all intermediate results remain satisfying truth assignments for $\phi$. This problem is PSPACE-complete [17], and the complexities of SAT Reconfiguration and its related problems have been studied very precisely [17, 30, 31, 34]. Ito et al. [22, Theorem 2] gave the following reduction.

▶ **Theorem 1** ([22]). *Let $\phi$ be a* 3-CNF*-formula with $N$ variables and $M$ clauses, as an instance of* SAT Reconfiguration*. Then, there is a corresponding instance of* Token Jumping *such that the input size is $O(NM)$ and its distance is at least that of the original instance.*

We note in passing that the distance of the corresponding instance of Token Jumping is at most $M$ times that of the original instance of SAT Reconfiguration.

Gopalan et al. [17, Lemma 3.7] constructed an instance series for SAT Reconfiguration such that formulas are 3-CNF and the distances of the instances are exponential in the input sizes. Then, by taking such SAT Reconfiguration instances, Theorem 1 gives a hard instance series for Token Jumping, which we call *the SAT series* in this paper.

### 3.1.2 Shortest-Path-based series

Kamiński et al. [27] introduced the *Shortest Path Reconfiguration problem*, as follows: we are given two shortest paths connecting two specified vertices $s$ and $t$ in an unweighted graph, and are asked to determine whether or not we can transform one into the other by exchanging a single vertex in a shortest path at a time so that all intermediate results remain shortest paths connecting $s$ and $t$. Surprisingly, this problem is PSPACE-complete [7, 41], and polynomial-time algorithms have been developed for restricted graph classes [1, 2, 7, 8, 15]. Kamiński et al. [28, Theorem 3] gave the following reduction.

▶ **Theorem 2** ([28]). *Let $G'$ be a graph with $N$ vertices, as an instance of* Shortest Path Reconfiguration*. Then, there is a corresponding instance of* Token Jumping *such that the input size is $O(N^2)$ and its distance is equal to that of the original instance.*

Kamiński et al. [27, Theorem 1] constructed an instance series for Shortest Path Reconfiguration such that the distances of the instances are exponential in the input sizes. Then, by taking such Shortest Path Reconfiguration instances, Theorem 2 gives a hard instance series for Token Jumping, which we call *the SP series* in this paper.

### 3.1.3 List-Coloring-based series

Let $C$ be a set of $k$ colors. For a graph $G' = (V', E')$, assume that each vertex $v$ in $V'$ has a *list* $L(v) \subseteq C$ of colors. Then, a *list coloring* of $G$ is to assign a color in $L(v)$ to each vertex $v$ in $V'$ so that no two adjacent vertices receive the same color. Then, the *List Coloring Reconfiguration problem* is defined as follows: we are given two list colorings of $G$, and are asked to determine whether or not we can transform one into the other by recoloring a single vertex at a time, so that all intermediate results remain list colorings of $G$. This problem and its related problems appear in several research fields, such as Glauber dynamics in statistical physics (e.g., see [24]).

List Coloring Reconfiguration has been studied also in the field of theoretical computer science. The problem is PSPACE-complete [9, 19, 41], and there are some tractable cases [6, 12, 19, 20, 26]. A standard reduction from list colorings to independent sets in graphs (e.g., see [33]) gives the following reduction also for reconfiguration problems.

▶ **Theorem 3.** *As an instance of* List Coloring Reconfiguration*, let $G'$ be a graph with $N$ vertices and $M$ edges, and let $|C| = k$. Then, there is a corresponding instance of* Token Jumping *such that the input size is $O(k^2N + kM)$ and its distance is equal to that of the original instance.*

**Proof.** We construct a graph $G$ for Token Jumping, as follows: For each vertex $v$ in $G'$, we construct a new clique of $|L(v)|$ vertices; each vertex in the clique corresponds to a color in $L(v)$. For each edge $uv$ in $G'$, if $L(u) \cap L(v) \neq \emptyset$, then we join the two vertices in the cliques for $u$ and $v$ that correspond to each color $c \in L(u) \cap L(v)$. Then, the theorem follows from the fact that a one-to-one correspondence holds between list colorings of $G'$ and maximum independent sets of $G$. ◀

Bonsma and Cereceda [9, Theorem 21] constructed an instance series for List Coloring Reconfiguration such that the distances of the instances are exponential in the input size. Then, by taking such List Coloring Reconfiguration instances, Theorem 3 gives a hard instance series for Token Jumping, which we call *the LC series* in this paper.

## 3.2 Original instance series

In this subsection, we construct our original hard instance series for Token Jumping, which we call *the IS series*[6] in this paper. We give the following theorem.

▶ **Theorem 4.** *Let $x \geq 3$, $y \geq 1$, $z \geq 1$ be any three integers such that $x$ is odd. Then, there is an instance of Token Jumping such that*

- *the number of vertices is $5xy + 2z$,*
- *the number of edges is $8xy - y + 2z - 1$, and*
- *its distance is $\Omega(z \cdot x^y)$.*

*More specifically, its distance is equal to*

$$\frac{x(x-1)^y - 2}{x-2} z + \frac{x(x-1)^y - (2x-4)y - x}{(x-2)^2}(3x-2) + 2y.$$

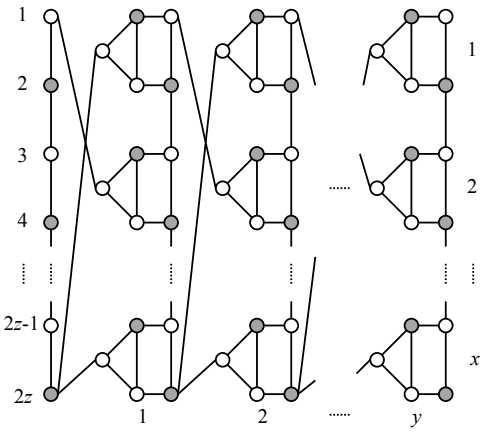
**Proof sketch.** We here sketch the construction of our instance, and roughly explain why it requires an exponential distance. The graph $G$ of the instance is shown in Figure 3. It consists of a *path* of $2z$ vertices (shown in the left of the figure) and $y$ *villages* (shown in Figure 4), where each village has $x$ *houses* (shown in Figure 5). We order the houses from top to bottom, and the villages from left to right, as labeled in Figure 3.

We say that a house is in the *north-position* (resp., *south-position*) if the two tokens in the house are placed as shown in Figure 5(a) (resp., in Figure 5(b)). For an independent set $I$ of $G$ and each $i \in \{1, 2, \ldots, y\}$, we denote by $s_i(I)$ the number of houses in the south-position in the $i$-th village; thus $s_i(I) \in \{0, 1, \ldots, x\}$. In the initial independent set $I_\mathrm{s}$, all the houses in all the villages are in the north-position, and hence $s_i(I_\mathrm{s}) = 0$ for all $i \in \{1, 2, \ldots, y\}$. In the target independent set $I_\mathrm{t}$, all the houses in the $y$-th village are in the south-position, and the others are in the north-position; that is, $s_y(I_\mathrm{t}) = x$ and $s_i(I_\mathrm{t}) = 0$ for all $i \in \{1, 2, \ldots, y-1\}$. Therefore, any reconfiguration sequence from $I_\mathrm{s}$ to $I_\mathrm{t}$ must change the value $s_y$ for the $y$-th village from 0 to $x$. The structure of $G$ forces the following properties:

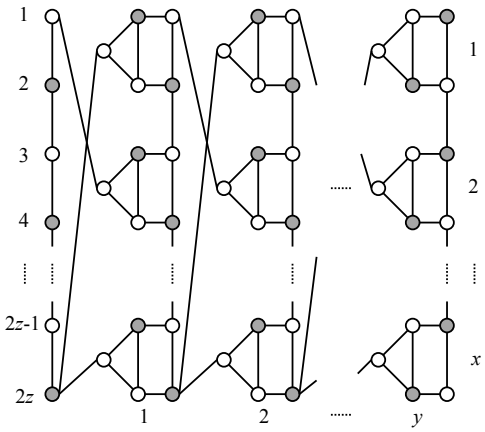
- to increase the value $s_y$ from $j$ to $j+1$ for even $j$, we need to change the value $s_{y-1}$ for the $(y-1)$-st village from 0 to $x$; and
- to increase the value $s_y$ from $j$ to $j+1$ for odd $j$, we need to change the value $s_{y-1}$ for the $(y-1)$-st village from $x$ to 0.

During the changes of $s_y$ from 0 to $x$, the above changes for $s_{y-1}$ happen $x$ times alternatively. Furthermore, $y$ villages are recursively connected in $G$, and this recursive structure yields the exponential factor $\Omega(x^y)$ into the distance of our instance. ◀

[6] IS stands for Independent Sets, since this instance series is designed originally for Token Jumping.




**(a)** An initial independent set $I_s$.




**(b)** A target independent set $I_t$.

**Figure 3** The construction of an instance in the IS series, where $x$ is odd.

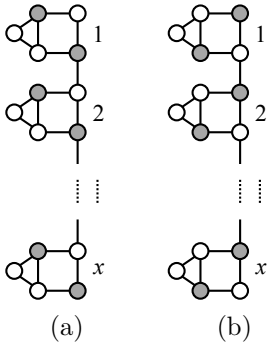



**Figure 4** The village gadgets used in the IS series. (a) The number of houses in the south-position in the village is 0. (b) The number of houses in the south-position in the village is $x$.

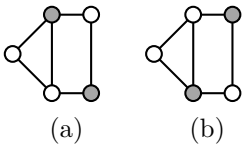



**Figure 5** The house gadgets used in the IS series. (a) A north-position. (b) A south-position.

## 4 Solvers Used for Evaluation

In this section, we describe three solvers that will be used for the evaluations in Section 5: SAT-based solver, ZDD-based solver, and IDA-BFS-based solver. The authors develop the SAT-based solver for this paper to perform the most fundamental evaluation, which will be available as described in Conclusions. The ZDD-based and IDA-BFS-based solvers have participated in CoRe Challenge 2022 [39]. The ZDD-based solver solved the instance with the largest shortest reconfiguration sequence in the challenge. The IDA-BFS-based solver solved the most instances and got the first place in the shortest track of the overall solver category. Both competition solvers are publicly available from the competition repository[7].

[7] `https://github.com/core-challenge/2022solver-showcase`

## 4.1 SAT-based solver

Let $(G, I_\mathrm{s}, I_\mathrm{t})$ be an instance of Token Jumping, where $G = (V, E)$. Our SAT-based solver employs the method of the bounded model checking [4]. More specifically, for a bound (integer) $\ell$, the solver determines whether there is a reconfiguration sequence of length $\ell$ between $I_\mathrm{s}$ and $I_\mathrm{t}$; and then we increment the bound $\ell$.

First, for an integer $i \in \{0, 1, \ldots, \ell\}$ and a vertex $u \in V$, let $p_{i,u}$ be a propositional variable such that it is true if and only if a token is placed on the vertex $u$ at Step $i$; we define the initial independent set $I_\mathrm{s}$ as Step 0. To represent an independent set at Step $i$, we introduce a function $\mathsf{Token}(i)$ as follows:

$$\mathsf{Token}(i) = \begin{cases} \bigwedge_{u \in I_\mathrm{s}} p_{i,u} \wedge \bigwedge_{u \notin I_\mathrm{s}} \neg p_{i,u} & \text{if } i = 0, \\ \bigwedge_{\{u,v\} \in E} (\neg p_{i,u} \vee \neg p_{i,v}) & \text{if } 0 < i < \ell, \\ \bigwedge_{u \in I_\mathrm{t}} p_{i,u} \wedge \bigwedge_{u \notin I_\mathrm{t}} \neg p_{i,u} & \text{if } i = \ell. \end{cases}$$

Next, we introduce two kinds of propositional variables $q^{10}_{i,u}$ and $q^{01}_{i,u}$ to represent a token movement at a vertex $u \in V$ between Steps $i$ and $i+1$: $q^{10}_{i,u}$ (resp., $q^{01}_{i,u}$) is true if and only if a token is removed from $u$ (resp., placed to $u$) between Steps $i$ and $i+1$. Then, the following constraint $\mathsf{Jump}(i)$ represents a single token movement between Steps $i$ and $i+1$:

$$\begin{aligned} \mathsf{Jump}(i) \quad = \quad & \left( \bigwedge_{u \in V} (q^{10}_{i,u} \leftrightarrow (p_{i,u} \wedge \neg p_{i+1,u})) \right) \\ & \wedge \left( \bigwedge_{u \in V} (q^{01}_{i,u} \leftrightarrow (\neg p_{i,u} \wedge p_{i+1,u})) \right) \\ & \wedge \left( \sum_{u \in V} q^{10}_{i,u} = 1 \right) \wedge \left( \sum_{u \in V} q^{01}_{i,u} = 1 \right). \end{aligned}$$

Note that the arithmetic constraint of $\sum_i x_i = 1$ can be encoded into propositional clauses by using the sequential counter [38].

Finally, we check whether there is a reconfiguration sequence of length $\ell$ by computing the satisfiability of the following formula $\Psi_\ell$:

$$\Psi_\ell = \bigwedge_{i=0}^{\ell} \mathsf{Token}(i) \wedge \bigwedge_{i=0}^{\ell-1} \mathsf{Jump}(i)$$

Our SAT-based solver increases $\ell$ one by one and performs satisfiability testing of $\Psi_\ell$. The method stops once the formula $\Psi_\ell$ becomes satisfiable, and outputs its model as a shortest reconfiguration sequence. In addition, incremental SAT solving [14] is used to speed up the computation by reusing learned clauses of SAT solvers. We implemented the proposed SAT-based method in the Scala language and used CaDiCaL [5] as its backend SAT solver.

## 4.2 ZDD-based solver

A zero-suppressed binary decision diagram (ZDD) is a data structure that efficiently represents the family of sets. Ito et al. [23] developed a solver that solves various reconfiguration problems (including Token Jumping) using ZDDs. In CoRe Challenge 2022 [39], the ZDD-based solver solved Token Jumping instances requiring longer shortest reconfiguration sequences that the other participants could not solve.

We briefly describe the algorithm of the ZDD-based solver for Token Jumping. The solver conducts the breadth-first search in the solution space $\mathcal{S}_k(G)$ starting from the node corresponding to $I_s$. (Recall the definition of $\mathcal{S}_k(G)$ given in Section 2.) Let $\mathcal{Z}^i$ be the family of *all* independent sets in $\mathcal{S}_k(G)$ that are distance at $i$ from $I_s$. We represent $\mathcal{Z}^i$ as a ZDD in a compressed form and construct $\mathcal{Z}^0, \mathcal{Z}^1, \ldots$ in turn, where $\mathcal{Z}^0 = \{I_s\}$. After constructing $\mathcal{Z}^i$, we check whether $\mathcal{Z}^i$ contains the target independent set $I_t$. If so, we can conclude that there is a reconfiguration sequence of length $i$ between $I_s$ and $I_t$. Ito et al. [23] designed an algorithm that efficiently constructs $\mathcal{Z}^i$ from $\mathcal{Z}^{i-1}$ using ZDD manipulation methods.

### 4.3 IDA-BFS-based solver

Turau and Weyer developed a solver for Token Jumping, which got the first place in the shortest track of the overall solver category in CoRe Challenge 2022 [39]. Their solver is a hybrid one carrying both the iteratively deepening A* algorithm (IDA*) [29] and the breadth-first search (BFS) on the solution space $\mathcal{S}_k(G)$. In this paper, we call their solver the IDA-BFS-based solver.

Roughly speaking, the IDA-BFS-based solver regards Token Jumping as the standard graph problem that searches a path in a graph $\mathcal{S}_k(G)$ connecting two nodes $I_s$ and $I_t$. Thus, we can make use of A* search. However, A* search needs a huge amount of memory because $\mathcal{S}_k(G)$ is usually large and it needs to store the visited nodes. To save the memory usage, the IDA-BFS-based solver employs the method that combines the depth-first search (DFS) with A* search, called IDA* [29]. In addition, when the answer to $(G, I_s, I_t)$ is no, IDA* searches nodes in $\mathcal{S}_k(G)$ many times. To avoid such a situation, the IDA-BFS-based solver uses BFS together with IDA*, which run in parallel in separate threads. Details are given in the first section of the description of CoRe Challenge 2022 [40].

## 5 Empirical Evaluations

In this section, we evaluate the instance series proposed in Section 3 by the solvers in Section 4. In the following, all experiments were conducted on a machine equipped with a 3.2 GHz CPU and 64GB of memory. The time limit is 30 minutes for each instance.

### 5.1 Benchmark instances

We generated benchmark instances from the series of SAT (Section 3.1.1), SP (Section 3.1.2), LC (Section 3.1.3), and IS (Section 3.2). In addition, for the purpose of comparisons, we generated random instances as follows:

1. Generate a random graph by specifying the numbers of vertices and edges[8]. These numbers were chosen to be approximately the same as those from benchmark instances of SAT, SP, LC, and IS for the purpose of comparison.
2. For the generated graph, compute two maximal independent sets of the same size as initial and target independent sets.
3. Check if there exists a reconfiguration sequence for the generated instance. If it exists, return it as a random instance; otherwise, attempt another pair of maximal independent sets. If a given number (this time 100) of attempts is unsuccessful, return to Step 1.

[8] Specifically, the `gnm_random_graph` method in the NetworkX graph library [18] is used, which chooses a graph uniformly at random from all the graphs with $n$ vertices and $m$ edges for specified $n$ and $m$.

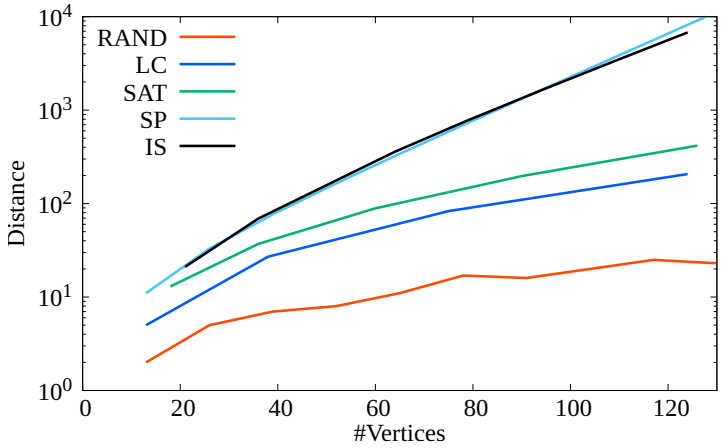



**Figure 6** Distance of instances with respect to the number of vertices.

For each $n$ and $m$, we generated five random instances. In the following, all values for random instances are their median values. For each series, instances with 130 vertices or fewer were generated. As we will describe in Conclusions, all benchmark instances and their generators are available online.

## 5.2 Evaluation

### 5.2.1 Distances of benchmark instances

First, we evaluated the benchmark instances by their distances, which are computed by one of the three solvers in Section 4. Figure 6 shows the distances, where the horizontal axis represents the number of vertices, and the vertical axis in a logarithmic scale represents the distance.

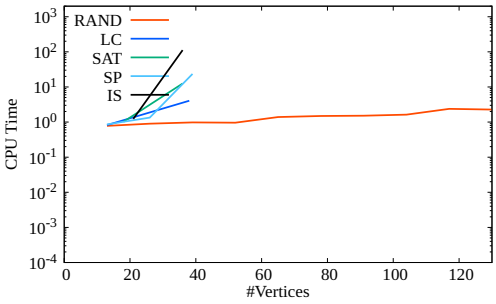
Recall that we have theoretically shown in Section 3 that distances of SAT, SP, LC and IS series increase exponentially with the numbers of vertices. This fact was also confirmed by our experiments as shown in Figure 6 (notice that the vertical axis uses a logarithmic scale): the plots of SP and IS draw steeply straight lines, and the plots of SAT and LC draw comparably mild straight lines.

On the other hand, the line of RAND no longer increases significantly when the number of nodes is over 100. This implies the difficulty of constructing scalable hard instances for Token Jumping in a straightforward way.

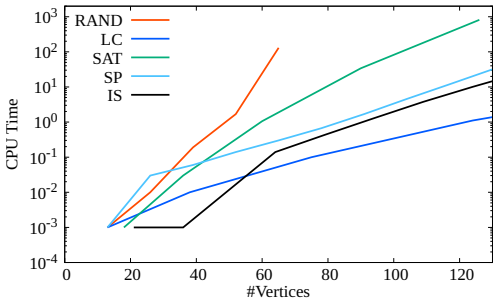
### 5.2.2 Evaluation by CPU time

Next, we evaluated the benchmark instances by CPU times of three solvers in Section 4. Recall that the SAT-based solver (Section 4.1) was developed by the authors for this paper to perform the most fundamental evaluation. On the other hand, the ZDD-based and IDA-BFS-based solvers (Sections 4.2 and 4.3, respectively) have participated in CoRe Challenge 2022 [39], and they are designed for Token Jumping. Figures 7(a), 7(b), and 7(c) show the CPU times of the SAT-based, ZDD-based, and IDA-BFS-based solvers, respectively. In each figure, the horizontal axis represents the number of vertices, and the vertical axis in a logarithmic scale represents the CPU time.

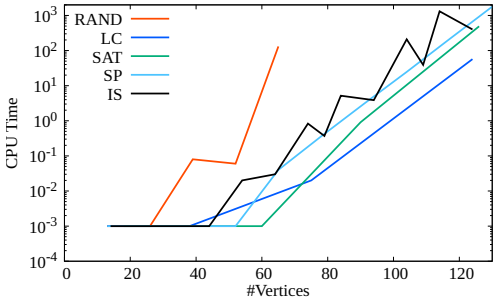
For the SAT-based solver (see Figure 7(a)), the CPU time increases sharply according to the increase in the number of vertices for all hard instance series SAT, SP, LC, and IS. In particular, hard instances having more than 50 vertices become unsolvable for the




**(a)** SAT-based solver.




**(b)** ZDD-based solver.




**(c)** IDA-BFS-based solver.

**Figure 7** CPU time with respect to the number of vertices.

SAT-based solver. In contrast, all instances in RAND are solved within 10 seconds, even when the number of vertices increases to more than 100. We remark that the reason why the computation time of the SAT solver exceeds one second for all instances is believed to be due to the use of the Scala programming language on the Java Virtual Machine for implementation, which results in significant overhead during startup.

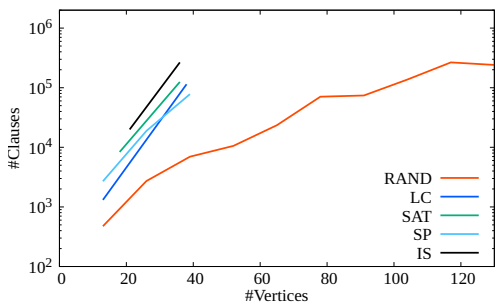
The ZDD-based and IDA-BFS-based solvers show similar behaviors, as shown in Figures 7(b) and 7(c), respectively. The plots of CPU time for all hard instance series SAT, SP, LC, and IS draw straight lines, namely showing the exponential growths of CPU times. Compared to the SAT-based solver, lines of SAT, SP, LC, and IS are much more gradual, and indeed the ZDD-based and IDA-BFS-based solvers can solve hard instances having more than 100 vertices. In this sense, our hard instance series are "nicely" hard to develop and improve a solver for Token Jumping.

We note that the lines of RAND for the ZDD-based and IDA-BFS-based solvers are different from that for the SAT-based solver: the ZDD-based and IDA-BFS-based solvers experience rapid increases in CPU times according to the numbers of vertices in those instances. We discuss the reason for this behavior in the following section, regarding the SAT-based and ZDD-based solvers.

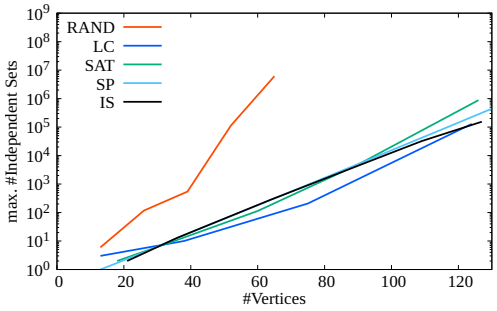
### 5.2.3 Analyses of solver performance

To understand the behavior of solvers, we add some more analyses for the SAT-based and ZDD-based solvers.

We analyzed the number of clauses in the SAT formula for the SAT-based solver because this is a standard metric for the difficulty of SAT instances. Figure 8(a) shows the growth of the number of clauses according to the number of vertices. As this figure shows, the number of clauses exponentially grows for all hard instance series SAT, SP, LC, and IS; while it




**(a)** Clauses in the SAT formula with respect to the number of vertices.




**(b)** Maximum number of independent sets in $\mathcal{Z}^i$ with respect to the number of vertices.

**Figure 8** Analyses of solvers' behavior.

becomes stable for RAND even when the number of vertices is over 100. These behaviors are almost the same as the increases in the distances of instances, as shown in Figure 6. Recall that our SAT-based solver employs the bounded model checking which depends on distances of instances, as explained in Section 4.1.

For the ZDD-based solver, we analyzed the number of independent sets that appear in the computation. Recall that $\mathcal{Z}^i$ is the family of *all* independent sets in the solution space $\mathcal{S}_k(G)$ that are distance at $i$ from $I_s$, and that the ZDD-based solver constructs $\mathcal{Z}^i$ from $\mathcal{Z}^{i-1}$ using ZDD manipulation methods. Thus, as a metric, we computed the number of independent sets in $\mathcal{Z}^i$, and take the maximum among $i = 0, 1, \ldots, \ell$, where $\ell$ is the distance of the instance. Figure 8(b) shows the growth of this maximum number of independent sets computed by the ZDD-based solver, according to the number of vertices. Then, compared to the line of RAND, those of SAT, SP, LC, and IS are much more gradual, and hence they are exponentially small numbers. This seems to be a natural property, because there are at most $2^{|V|}$ independent sets in a graph $G = (V, E)$, and hence the number of independent sets at distance $i$ would be small if the distance $\ell$ is exponential. Because the ZDD-based solver conducts the breadth-first search on $\mathcal{S}_k(G)$, it becomes more effective for hard instances and less effective for random instances in RAND.

### 5.2.4 Evaluation summary

We summarize the evaluations. As theoretically shown in Section 3, all the proposed instance series SAT, SP, LC, and IS result in exponential growths in the distances according to the number of vertices. On the other hand, as shown in Figure 6, we could not observe such growth in the distance for random instances in RAND. In addition, as mentioned in Introduction, recall that instances in CoRe Challenge 2022 [39] made from graphs in the DIMACS challenge do not have long distances: despite the wide range of vertex numbers from 11 to 10000, 80% of such instances have distance less than 10, and the longest distance is only 112. Therefore, we confirmed the effectiveness of theoretical approaches to creating benchmark instances, particularly when we require scalability. In this context, there is a related work [10] that analyzes the asymptotic behavior of longest distances of instances among all graphs having prescribed numbers of vertices and tokens, although their intention is not to focus on the construction of hard instances.

In addition, we analyzed how this distance property affects the performances of solvers. We confirmed that the required CPU times by three solvers also show scalability and exponential growth according to the number of vertices. This implies that solvers adept at handling long distances demonstrate good performance, resulting in identical rankings for each solver.

## 6 Conclusions

In this paper, we proposed four types of instance series for Token Jumping, to accelerate practical studies of general-purpose solvers that do not rely on prior problem-specific knowledge. Our instance series is scalable in the sense that instance scales are controlled by the numbers of vertices of input graphs. We focused on the distances of instances to establish the hardness, and theoretically show that our instance series require exponential numbers of steps with respect to the number of vertices. We confirmed their scalability and hardness by three distinct solvers: SAT-based solver, ZDD-based solver, and IDA-BFS-based solver. We emphasize again that constructing such scalable and hard instances is not a trivial task. In addition, our experiments demonstrated that the randomly generated instances can be sufficiently hard for two types of solvers, but not for the SAT-based solver. All benchmark instances, experimental result logs, and solver programs are available on a GitHub repository[9].

As a future work, there would be other ways to establish the hardness of instances for Token Jumping. For example, there may be instances where the distance is short but listing all candidate independent sets for the next step is difficult. The difficulty of such instances may depend on other parameters such as the size of independent sets. Future work also includes creating benchmark datasets for other reconfiguration problems.

## References


1 John Asplund, Kossi D. Edoh, Ruth Haas, Yulia Hristova, Beth Novick, and Brett Werner. Reconfiguration graphs of shortest paths. *Discrete Mathematics*, 341(10):2938–2948, 2018. doi:10.1016/j.disc.2018.07.007.

2 John Asplund and Brett Werner. Classification of reconfiguration graphs of shortest path graphs with no induced 4-cycles. *Discrete Mathematics*, 343(1):111640, 2020. doi:10.1016/j.disc.2019.111640.

3 Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

4 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, 1999.

5 Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In Tomas Balyo, Nils Froleyks, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*, volume B-2020-1 of *Department of Computer Science Report Series B*, pages 51–53. University of Helsinki, 2020.

6 Paul Bonsma, Amer E. Mouawad, Naomi Nishimura, and Venkatesh Raman. The complexity of bounded length graph recoloring and CSP reconfiguration. In *Parameterized and Exact Computation - 9th International Symposium, IPEC 2014, Wroclaw, Poland, September 10-12, 2014. Revised Selected Papers*, pages 110–121, 2014. doi:10.1007/978-3-319-13524-3_10.

7 Paul S. Bonsma. The complexity of rerouting shortest paths. *Theoretical Computer Science*, 510:1–12, 2013. doi:10.1016/j.tcs.2013.09.012.

8 Paul S. Bonsma. Rerouting shortest paths in planar graphs. *Discrete Applied Mathematics*, 231:95–112, 2017. doi:10.1016/j.dam.2016.05.024.


[9] https://github.com/TakehideSoh/Scalable-Hard-ISR

9 Paul S. Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410(50):5215–5226, 2009. doi:10.1016/j.tcs.2009.08.023.

10 Nicolas Bousquet, Bastien Durain, Théo Pierron, and Stéphan Thomassé. Extremal independent set reconfiguration. *Electronic Journal of Combinatorics*, 30(3):P3.8, 2023. doi:10.37236/11771.

11 Nicolas Bousquet, Amer E. Mouawad, Naomi Nishimura, and Sebastian Siebertz. A survey on the parameterized complexity of the independent set and (connected) dominating set reconfiguration problems. *CoRR*, abs/2204.10526, 2022. doi:10.48550/arXiv.2204.10526.

12 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011. doi:10.1002/jgt.20514.

13 Remo Christen, Salomé Eriksson, Michael Katz, Christian Muise, Alice Petrov, Florian Pommerening, Jendrik Seipp, Silvan Sievers, and David Speck. PARIS: Planning algorithms for reconfiguring independent sets. In *ICAPS 2023 Heuristics and Search for Domain-Independent Planning Workshop*, 2023. URL: https://openreview.net/forum?id=LE8nB7aHV4.

14 Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4), 2003.

15 Kshitij Gajjar, Agastya Vibhuti Jha, Manish Kumar, and Abhiruk Lahiri. Reconfiguring shortest paths in graphs. In *Proceedings of AAAI 2022*, pages 9758–9766. AAAI Press, 2022. URL: https://ojs.aaai.org/index.php/AAAI/article/view/21211.

16 Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning - Theory and Practice*. Elsevier, 2004.

17 Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of Boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009. doi:10.1137/07070440X.

18 Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

19 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. The list coloring reconfiguration problem for bounded pathwidth graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E98.A(6):1168–1178, 2015. doi:10.1587/transfun.E98.A.1168.

20 Tatsuhiko Hatanaka, Takehiro Ito, and Xiao Zhou. Parameterized complexity of the list coloring reconfiguration problem with graph parameters. *Theoretical Computer Science*, 739:65–79, 2018. doi:10.1016/j.tcs.2018.05.005.

21 Jan van den Heuvel. The complexity of change. In *Surveys in Combinatorics 2013*, volume 409 of *London Mathematical Society Lecture Note Series*, pages 127–160. Cambridge University Press, 2013. doi:10.1017/CBO9781139506748.005.

22 Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12–14):1054–1065, 2011. doi:10.1016/j.tcs.2010.12.005.

23 Takehiro Ito, Jun Kawahara, Yu Nakahata, Takehide Soh, Akira Suzuki, Junichi Teruyama, and Takahisa Toda. ZDD-based algorithmic framework for solving shortest reconfiguration problems. In Andre A. Cire, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 167–183, Cham, 2023. Springer Nature Switzerland.

24 Mark Jerrum. *Counting, Sampling and Integrating: Algorithms and Complexity*. Birkhäuser Verlag, Basel, 2003.

25 David S. Johnson and Michael A. Trick. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993*, volume 26. American Mathematical Society, 1996.

26 Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Finding shortest paths between graph colourings. *Algorithmica*, 75(2):295–321, 2016. doi:10.1007/s00453-015-0009-7.

**27** Marcin Kamiński, Paul Medvedev, and Martin Milanič. Shortest paths between shortest paths. *Theoretical Computer Science*, 412(39):5205–5210, 2011. doi:10.1016/j.tcs.2011.05.021.

**28** Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. doi:10.1016/j.tcs.2012.03.004.

**29** Richard E. Korf. Iterative-deepening-A*: An optimal admissible tree search. In Aravind K. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 1985*, pages 1034–1036. Morgan Kaufmann, 1985.

**30** Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. On the Boolean connectivity problem for Horn relations. *Discrete Applied Mathematics*, 158(18):2024–2030, 2010. doi:10.1016/j.dam.2010.08.019.

**31** Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. An exact algorithm for the Boolean connectivity problem for $k$-CNF. *Theoretical Computer Science*, 412(35):4613–4618, 2011. doi:10.1016/j.tcs.2011.04.041.

**32** Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th ACM/IEEE design automation conference*, pages 272–277, 1993. doi:10.1145/157485.164890.

**33** Cristopher Moore and Stephan Mertens. *The Nature of Computation*. Oxford University Press, Inc., New York, 2011.

**34** Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of Boolean formulas. *SIAM Journal on Discrete Mathematics*, 31(3):2185–2200, 2017. doi:10.1137/16M1065288.

**35** Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. *Algorithmica*, 78(1):274–297, 2017. URL: https://doi.org/10.1007/s00453-016-0159-2, doi:10.1007/S00453-016-0159-2.

**36** Artur Niewiadomski, Piotr Switalski, Teofil Sidoruk, and Wojciech Penczek. Applying modern SAT-solvers to solving hard problems. *Fundamenta Informaticae*, 165(3-4):321–344, 2019. doi:10.3233/FI-2019-1788.

**37** Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4):Paper id 52, 2018. doi:10.3390/a11040052.

**38** Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, 2005. doi:10.1007/11564751_73.

**39** Takehide Soh, Yoshio Okamoto, and Takehiro Ito. Core challenge 2022: Solver and graph descriptions. *CoRR*, abs/2208.02495, 2022. arXiv:2208.02495, doi:10.48550/arXiv.2208.02495.

**40** Volker Turau and Christoph Weyer. Finding shortest reconfigurations sequences of independent sets. In *Core Challenge 2022: Solver and Graph Descriptions*, pages 3–14, 2022.

**41** Marcin Wrochna. Reconfiguration in bounded bandwidth and tree-depth. *Journal of Computer and System Sciences*, 93:1–10, 2018. doi:10.1016/j.jcss.2017.11.003.

**42** Yuya Yamada, Mutsunori Banbara, Katsumi Inoue, Torsten Schaub, and Ryuhei Uehara. Combinatorial reconfiguration with answer set programming: Algorithms, encodings, and empirical analysis. In Ryuhei Uehara, Katsuhisa Yamanaka, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation - 18th International Conference and Workshops on Algorithms and Computation, WALCOM 2024, Kanazawa, Japan, March 18-20, 2024, Proceedings*, volume 14549 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2024. doi:10.1007/978-981-97-0566-5_18.