Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
KR in the Wild

SRIP: A SAT-based System for Independent Set Reconfiguration

Takehide Soh[1] , **Akifumi Kuwahara**[2] , **Mutsunori Banbara**[1] , **Naoyuki Tamura**[2] ,
Yasuaki Kobayashi[3] , **Yuta Nozaki**[3,4] , **Takehiro Ito**[5]
[1]Nagoya University, Nagoya, Japan
[2]Kobe University, Kobe, Japan
[3]Hokkaido University, Sapporo, Japan
[4]SKCM[2], Hiroshima University, Hiroshima, Japan
[5]Tohoku University, Sendai, Japan
soh@i.nagoya-u.ac.jp, 232x023x@gsuite.kobe-u.ac.jp, banbara@nagoya-u.jp, tamura@kobe-u.ac.jp,
koba@ist.hokudai.ac.jp, nozaki@math.sci.hokudai.ac.jp, takehiro@tohoku.ac.jp

Abstract

We present SRIP, a SAT-based system for solving the Independent Set Reconfiguration Problem (ISRP) under the Token Jumping (TJ) rule. SRIP formulates ISRP with SAT problems employing a clique-partition-based constraint model and a set of pruning constraints that strengthen propagation and reduce the search space for reconfiguration. The resulting model is compiled into a sequence of SAT problems and solved using incremental SAT within a bounded model checking framework, enabling SRIP to compute shortest reconfiguration sequences efficiently. We evaluate SRIP on benchmark instances from the CoRe Challenge, a competition series dedicated to ISRP under TJ. SRIP finds optimal (shortest) reconfiguration sequences for 477 out of 693 instances, achieving the best results among state-of-the-art solvers on this benchmark suite.

1 Introduction

Combinatorial reconfiguration (Ito et al. 2011; Nishimura 2018) is an emerging field that has gained significant traction, particularly in the study of algorithms and complexity. Its primary objective is to analyze the structure and properties of solution spaces associated with combinatorial (search) problems. In this framework, a solution space is modeled as a graph, where each node represents a feasible solution of the associated problem and edges represent an adjacency relation (i.e., a single allowed modification) between these solutions. Common research topics include the reachability, distance, diameter, and connectivity of such solution spaces. For example, given an instance of a combinatorial problem and two of its feasible solutions, the reachability problem asks whether there exists a path between the two solutions in the corresponding solution space.

The motivation for combinatorial reconfiguration stems from a wide range of areas, including statistical physics (Mohar and Salas 2009), combinatorics (Knuth 2011), puzzles (Hearn and Demaine 2009), and industrial applications such as power distribution networks (Inoue et al. 2014). For instance, in power distribution networks, we must find a sequence of feasible switch configurations that avoids blackouts. Beyond these motivation, there is a strong theoretical interest in the field. Interestingly, many combinatorial problems that are NP-complete in their original form are known to be PSPACE-complete in their reconfiguration versions (Ito et al. 2011); examples include SAT Reconfiguration (Gopalan et al. 2009), Independent Set Reconfiguration (Kaminski, Medvedev, and Milanic 2012), Coloring Reconfiguration (Bonsma and Cereceda 2009), Clique Reconfiguration (Ito, Ono, and Otachi 2015), and Hamiltonian Cycle Reconfiguration (Takaoka 2018). This PSPACE-completeness implies that even a shortest path in the solution space needs a super-polynomial length, under the assumption that PSPACE $\neq$ NP. Furthermore, studying these solution spaces has occasionally yielded deeper insights in the analysis of the original problems themselves, such as graph colorings (Brewster et al. 2016).

Due to this inherent computational difficulty, practical aspects of combinatorial reconfiguration have recently gained momentum. This trend is driven by the CoRe Challenge 2022/2023 (Soh et al. 2024a)[1,2], an international competition series that aims to build a practical foundation for the field by providing benchmark instances, instance formats, and baseline solvers. The central problem of the CoRe Challenge is the *Independent Set Reconfiguration Problem* (ISRP), formally defined in Section 2, which is one of the most extensively studied problems in the field. The challenge released 11 benchmark families totaling 693 instances, spanning structured graphs (e.g., grid, queen, power) and random or handcrafted instances (Soh et al. 2024b). On this benchmark set, solvers from various paradigms have been proposed, including AI planning solvers, heuristic search (IDA*/BFS), SAT- and ASP-based bounded model checking (BMC), BMC with the symbolic model checker NuSMV, ZDD-based breadth-first search, and reinforcement learning, as showcased in solver descriptions (Soh et al. 2023; Soh, Okamoto, and Ito 2022). Among them, the top-performing solvers are PARIS (Christen et al. 2023) and NUASP (Yamada et al. 2024), which are AI planning and ASP-based BMC solvers, respectively. Although these two solvers are award-winning and represent the current state of the art, there remains room for further refinement.

[1]https://core-challenge.github.io/2022/
[2]https://core-challenge.github.io/2023/

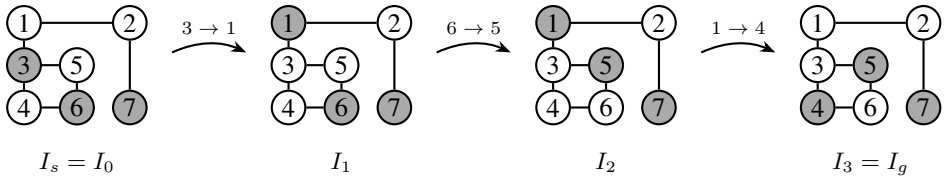


Figure 1: Example of an ISRP instance and a solution under Token Jumping. The graph has 7 vertices and 7 edges, with $I_s = \{3, 6, 7\}$ and $I_g = \{4, 5, 7\}$. The sequence $\langle I_s, I_1, I_2, I_g \rangle$ is a shortest reconfiguration sequence of length 3.

In this paper, we present SRIP, a SAT-based system for independent set reconfiguration under the *Token Jumping* rule. Our approach is based on bounded model checking (BMC; (Biere et al. 1999)), a formal verification technique that searches for a path of bounded length between states. A key technical contribution is a lightweight cardinality modeling which reduces the number of clauses required. We also introduce pruning constraints that enhance propagation in our SAT-based BMC framework. The contributions and results of this paper are summarized as follows:

1. We present SRIP, a SAT-based BMC solver for ISRP under the Token Jumping rule. Our approach leverages incremental SAT solving to efficiently explore reconfiguration sequences of increasing length. SRIP finds optimal (shortest) reconfiguration sequences for 477 out of 693 CoRe Challenge instances, achieving the best results among state-of-the-art solvers.
2. We introduce *clique-partition-based cardinality encoding*, a practical constraint encoding that exploits the structure of independent sets. This technique reduces the number of clauses for the exact-k constraint by switching from vertex-level to clique-level cardinality modeling, yielding a 3–10$\times$ clause reduction across diverse graph families (median compression ratio 0.45).
3. We integrate pruning constraints that exploit distance information from the start and goal independent sets to prune the search space for reconfiguration. Ablation studies show that combining pruning constraints with our cardinality encoding improves performance from 300 to 477 solved instances on the CoRe Challenge benchmark.
4. We provide a complete artifact package (implementation, scripts, benchmark instances, and full result logs) to support reproducibility and future comparisons, aligning with the goals of the "KR in the Wild" track.

Overall, our work demonstrates that SAT-based BMC, when carefully engineered with domain-specific techniques, can be highly competitive for ISRP.

2 Preliminaries

2.1 Independent Set Reconfiguration Problem

Let $G = (V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. A vertex subset $I \subseteq V$ is an *independent set of* G if no two vertices in I are adjacent in G. Figure 1 shows four independent sets I_s, I_1, I_2, I_g of size 3 in the same graph. Henceforth, we refer to each vertex contained in an independent set as a *token*.

Independent Set Reconfiguration Problem (ISRP) asks whether there exists a sequence of independent sets transforming a start independent set I_s into a goal independent set I_g, following prescribed reconfiguration rules. *Token Jumping* (TJ) is a well-studied reconfiguration rule, where a single reconfiguration step replaces one vertex in the current independent set I with another vertex that is not in I.

Formally, an instance of ISRP under TJ is a tuple (G, I_s, I_g), where I_s and I_g are independent sets of G such that $|I_s| = |I_g|$. A *TJ reconfiguration sequence* is a sequence $\langle I_0, I_1, \ldots, I_i, \ldots, I_\ell \rangle$ such that:

- **R1.** $I_0 = I_s$ and $I_\ell = I_g$;
- **R2.** $|I_i| = |I_s|$ for each i, $0 \leq i \leq \ell$;
- **R3.** I_i is an independent set of G for each i, $0 \leq i \leq \ell$;
- **R4.** $|I_i \triangle I_{i+1}| = 2$ for each i, $0 \leq i \leq \ell - 1$, namely, exactly one token jumps to another vertex.

Then, the *Independent Set Reconfiguration Problem* (ISRP) under TJ is to determine whether there exists a TJ reconfiguration sequence from I_s to I_g.

In this paper, we focus on the shortest variant of ISRP, which generalizes ISRP. Given an ISRP instance (G, I_s, I_g), the *shortest ISRP* under TJ aims to determine the existence of a TJ reconfiguration sequence from I_s to I_g and, if so, find one of minimum length.

Figure 1 illustrates a concrete instance with $I_s = \{3, 6, 7\}$ and $I_g = \{4, 5, 7\}$. Each step replaces exactly one vertex while preserving independence and the number of tokens, i.e., $|I_i| = |I_s|$ and $|I_i \triangle I_{i+1}| = 2$. The sequence $\langle I_s, I_1, I_2, I_g \rangle$ in the figure shows a valid TJ reconfiguration sequence of length 3, and is shortest for this instance.

2.2 Propositional Satisfiability

A *propositional variable* takes a value in $\{true, false\}$. A *literal* is a variable x (positive) or its negation $\neg x$ (negative). A *clause* is a disjunction of literals, e.g., $(x_1 \vee \neg x_2 \vee x_3)$. A formula in *conjunctive normal form* (CNF) is a conjunction of clauses. An *assignment* maps every variable to $true$ or $false$. An assignment that makes a CNF formula true is called a *satisfying assignment* (or *solution*). A formula is *satisfiable* (SAT) if a satisfying assignment exists, and *unsatisfiable* (UNSAT) otherwise. The *satisfiability problem* (SAT) asks whether a given CNF formula is satisfiable.

In what follows, we regard each $x_i \in \{\mathrm{false}, \mathrm{true}\}$ as a $0/1$ variable by identifying true with 1 and false with 0. A *cardinality constraint* $\sum_{i=1}^{n} x_i = k$ (or $\leq k$, $\geq k$) restricts the number of true variables. Such constraints can be encoded into CNF using, e.g., the sequential counter (Sinz

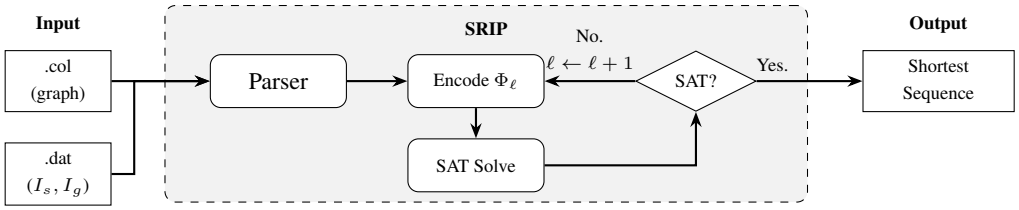


Figure 2: Architecture of SRIP. The parser reads graph and instance files, then the BMC loop iteratively encodes and solves SAT instances for increasing bounds until a solution is found or timeout.

2005) or the totalizer (Bailleux and Boufkhad 2003), both producing $O(nk)$ clauses.

In *incremental SAT solving*, clauses are added to the solver without discarding previous clauses, and each call to the solver may specify a set of *assumptions*—literals that are temporarily asserted for that call only. A common technique is to use *block literals*: a fresh (auxiliary) variable b is added as a negative literal $\neg b$ to a set of clauses, which are then activated by assuming $b = \mathit{true}$ and deactivated by assuming $b = \mathit{false}$ or adding the unit clause $(\neg b)$. This mechanism is central to the BMC loop described in Section 3.3.

Encoding the constraint $\sum x_i = k$ into CNF is a fundamental task in SAT-based combinatorial solving. The sequential counter encoding (Sinz 2005) and the totalizer encoding (Bailleux and Boufkhad 2003) both produce $O(nk)$ clauses. As we will see in Section 3, our clique-based approach is orthogonal to the choice of cardinality encoding: it reduces the number of variables fed to the encoder from $|V|$ to the number m of cliques that form a partition of V, and can be combined with any encoding.

3 SRIP Approach

SRIP (SAT-based Reconfiguration for Independent Set) uses BMC (Bounded Model Checking) (Biere et al. 1999) to solve ISRP under the Token Jumping (TJ) rule. The system overview of SRIP is illustrated in Figure 2. Given an instance (G, I_s, I_g) with $|I_s| = |I_g| = k$, SRIP iteratively constructs and solves SAT instances for increasing bounds ℓ, corresponding to the length of a TJ reconfiguration sequence, until a satisfying assignment is found or a global timeout is reached. When satisfiable, SRIP extracts a TJ reconfiguration sequence from the satisfying assignment and validates it. Let x_u^t be a Boolean variable indicating whether vertex $u \in V$ is in the independent set (and hence has a token) at time step $t \in \{0, \ldots, \ell\}$. Let X^t be the set of all such variables at time step t. For a fixed bound ℓ, we build a propositional formula Ψ_ℓ that is satisfiable if and only if there exists a TJ reconfiguration sequence of length ℓ from I_s to I_g. Its core is the following constraints.

$$\Psi_\ell \;=\; \mathit{Init}(X^0) \wedge \mathit{Goal}(X^\ell) \wedge \bigwedge_{t=1}^{\ell-1} \mathit{Token}(X^t) \wedge \bigwedge_{t=0}^{\ell-1} \mathit{Jump}(X^t, X^{t+1}).$$

The constraints $\mathit{Init}(X^0)$ and $\mathit{Goal}(X^\ell)$ ensure that the start and goal independent sets are I_s and I_g, respectively.

$$\mathit{Init}(X^0) \;=\; \bigwedge_{v \in I_s} x_v^0 \wedge \bigwedge_{v \notin I_s} \neg x_v^0,$$
$$\mathit{Goal}(X^\ell) \;=\; \bigwedge_{v \in I_g} x_v^\ell \wedge \bigwedge_{v \notin I_g} \neg x_v^\ell.$$

The constraint $\mathit{Token}(X^t)$ ensures that the set of vertices with tokens at time step t forms an independent set of G.

$$\mathit{Token}(X^t) \;=\; \bigwedge_{\{u,v\} \in E} (\neg x_u^t \vee \neg x_v^t) \wedge \sum_{v \in V} x_v^t = k.$$

The constraint $\mathit{Jump}(X^t, X^{t+1})$ ensures that exactly one token departs at each transition from time step t to $t+1$.

$$\mathit{Jump}(X^t, X^{t+1}) \;=\; \bigwedge_{v \in V} p_v^t \;\leftrightarrow\; (x_v^t \wedge \neg x_v^{t+1}) \wedge \sum_{v \in V} p_v^t = 1$$

where p_v^t is an auxiliary Boolean variable indicating whether the token at vertex v jumps out at time step t.

Proposition 1. *For a given ISRP instance (G, I_s, I_g) with $|I_s| = |I_g| = k$, there exists a TJ reconfiguration sequence of length ℓ from I_s to I_g if and only if Ψ_ℓ is satisfiable.*

Proof. ($\Rightarrow$) Suppose a TJ reconfiguration sequence $\langle I_0, I_1, \ldots, I_\ell \rangle$ exists. Set $x_v^t = \mathit{true}$ iff $v \in I_t$ for each $v \in V$ and $0 \le t \le \ell$. By **R1**, $I_0 = I_s$ and $I_\ell = I_g$, so $\mathit{Init}(X^0)$ and $\mathit{Goal}(X^\ell)$ are satisfied. By **R3**, each I_t is an independent set, and by **R2**, $|I_t| = k$, so $\mathit{Token}(X^t)$ is satisfied for all t. By **R4**, $|I_t \triangle I_{t+1}| = 2$, meaning exactly one vertex loses its token and one gains one, so setting $p_v^t = \mathit{true}$ iff $v \in I_t \setminus I_{t+1}$ satisfies $\mathit{Jump}(X^t, X^{t+1})$. Hence Ψ_ℓ is satisfied. ($\Leftarrow$) Suppose Ψ_ℓ is satisfiable with assignment σ. Define $I_t = \{v \in V \mid \sigma(x_v^t) = \mathit{true}\}$ for each $0 \le t \le \ell$. $\mathit{Init}(X^0)$ and $\mathit{Goal}(X^\ell)$ ensure **R1**. The independence and cardinality clauses in $\mathit{Token}(X^t)$ ensure **R2** and **R3**. The linking and cardinality clauses in $\mathit{Jump}(X^t, X^{t+1})$ ensure that exactly one token is removed and one is placed, giving $|I_t \triangle I_{t+1}| = 2$, which is **R4**. Hence $\langle I_0, I_1, \ldots, I_\ell \rangle$ is a valid TJ reconfiguration sequence. □

Throughout, we assume that $I_s \neq I_g$, excluding the trivial case $I_s = I_g$. In the process of searching for a shortest reconfiguration sequence by increasing the bound on the

number of reconfiguration steps one by one, satisfiability of $\Psi_1, \Psi_2, \ldots$ forms a pattern of (UNSAT)* SAT. By checking satisfiability for $\ell = 1, 2, 3, \ldots$ in increasing order, the first bound for which Ψ_ℓ is satisfiable yields the shortest TJ reconfiguration sequence.

As a limitation of BMC, clauses grow with the length of a TJ reconfiguration sequence. The overall formula Ψ_ℓ contains $O(\ell(|E|+|V|k))$ clauses, where independence clauses $O(\ell|E|)$ are binary and the cardinality constraint of exact-k (controlling the number of tokens), $\sum_{v\in V} x_v^t = k$, inside $Token(X^t)$ contributes $O(\ell|V|k)$ clauses, which could be a bottleneck. Therefore, we introduce a new technique in the next section to mitigate this bottleneck.

3.1 Clique-based Modeling for Exact-k

A major bottleneck in the above basic approach is repeatedly enforcing $\sum_{v\in V} x_v^t = k$ across time steps. Our idea is to reduce this bottleneck by modeling the cardinality constraint of exact-k at the level of cliques rather than individual vertices.

We here introduce a clique partition $\mathcal{P} = \{C_1, \ldots, C_m\}$ of a graph G. A *clique* in a graph $G = (V, E)$ is a subset $C \subseteq V$ such that every pair of distinct vertices in C is adjacent (i.e., $\forall u, v \in C, u \neq v \Rightarrow \{u, v\} \in E$). A *clique partition* of G is a collection $\mathcal{P} = \{C_1, \ldots, C_m\}$ such that

(i) C_i is a clique in G for each i, $1 \leq i \leq m$,
(ii) $C_i \cap C_j = \emptyset$ for every $i \neq j$, $1 \leq i, j \leq m$, and
(iii) $\bigcup_{i=1}^m C_i = V$.

Notice that any independent set of G can contain only one vertex from each clique C_i, $1 \leq i \leq m$.

Using a clique partition $\mathcal{P} = \{C_1, \ldots, C_m\}$ of G, we replace $Token(X^t)$ with a clique-level variant:

$$Token^{CC}(X^t, \mathcal{P}) \;=\; \bigwedge_{\{u,v\}\in E} (\neg x_u^t \vee \neg x_v^t) \wedge CC(X^t, \mathcal{P}).$$

Given $\mathcal{P} = \{C_1, \ldots, C_m\}$, at step t we introduce clique-occupancy variables c_i^t linked to vertex variables by

$$CC(X^t, \mathcal{P}) \;=\; \sum_{i=1}^m c_i^t = k \;\wedge\; \bigwedge_{i=1}^m \left(c_i^t \leftrightarrow \bigvee_{v\in C_i} x_v^t \right).$$

Proposition 2. *$Token(X^t)$ and $Token^{CC}(X^t, \mathcal{P})$ are equivalent.*

Proof. Let t be a fixed time step and $\mathcal{P} = (C_1, \ldots, C_m)$ of G be a clique partition of G. Since each C_i is a clique, for any distinct $u, v \in C_i$ we have $\{u, v\} \in E$. Hence the independence clauses $\bigwedge_{\{u,v\}\in E}(\neg x_u^t \vee \neg x_v^t)$ imply that at most one variable among $\{x_v^t \mid v \in C_i\}$ can be true. By the linking constraint $\bigwedge_{i=1}^m \left(c_i^t \leftrightarrow \bigvee_{v\in C_i} x_v^t\right)$, $c_i^t = 1$ iff at least one vertex in C_i has a token; with the "at most one" property, this is equivalent to $c_i^t = \sum_{v\in C_i} x_v^t$. Summing over all cliques and using that $\mathcal{P}$ is a partition of V,

$$\sum_{i=1}^m c_i^t = \sum_{i=1}^m \sum_{v\in C_i} x_v^t = \sum_{v\in V} x_v^t.$$

Thus $\sum_{v\in V} x_v^t = k$ iff $\sum_{i=1}^m c_i^t = k$. Both $Token(X^t)$ and $Token^{CC}(X^t, \mathcal{P})$ share the same independence clauses, so the two constraints are equivalent. □

In SRIP we enforce the cardinality constraint of exact-k on the m clique variables (with $m < |V|$), which empirically reduces the size and difficulty of the SAT instances.

Clique partition computation. Given an undirected graph $G = (V, E)$, we compute a clique partition $\mathcal{P} = (C_1, \ldots, C_m)$ of V using a greedy heuristic. Vertices are processed in decreasing order of degree and each vertex is assigned to an existing clique to which it is fully adjacent, preferring cliques whose current size is below an estimated target; if no compatible clique exists, a new singleton clique is created.

The procedure consists of two phases: an initial greedy assignment (Algorithm 1) followed by a rebalancing pass (Algorithm 2).

In Algorithm 1, by `EstimateMaxCliqueSize`, a target clique size τ is first estimated as follows: the top-s vertices by degree (we use $s = 10$) are each used as a seed, and a clique is greedily extended from each seed by adding, in decreasing degree order, vertices that are adjacent to all current members; τ is set to the maximum clique size found. Vertices are then processed in decreasing order of degree. For each vertex v, the algorithm scans all existing cliques C_i and considers only those to which v is *fully adjacent* (i.e. v is a neighbor of every vertex in C_i), so adding v preserves the clique property. Among the compatible cliques, a best-fit score s is computed: under-filled cliques ($|C_i| < \tau$) receive a positive score $\tau - |C_i|$, while filled or over-filled cliques receive a negative score $-|C_i|$. This scoring strongly prefers placing v into the most under-filled clique, encouraging all cliques to grow toward τ. The vertex is assigned to the clique with the highest score; if no compatible clique exists, a new singleton clique $\{v\}$ is created. Processing high-degree vertices first is important because they have many neighbors and are therefore more likely to be compatible with existing cliques; low-degree vertices, handled later, are more likely to require new cliques.

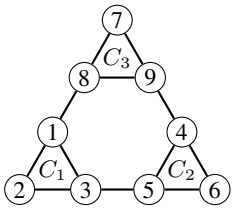
Algorithm 2 then improves the partition by consolidating small cliques. It iterates over cliques in increasing order of size. For each small clique C_s, the algorithm attempts to move each of its vertices v to a larger clique C_t such that (i) v is adjacent to all vertices of C_t and (ii) $|C_t|$ is minimized among all feasible targets with $|C_t| \geq |C_s|$. Condition (ii) ensures that vertices migrate toward moderately larger cliques rather than always gravitating to the largest one, preserving balance. This process repeats until a full pass produces no further moves. Finally, any cliques that have become empty are removed, and the remaining cliques are sorted by decreasing size.

Example. Let $V = \{1, \ldots, 9\}$ and suppose a clique partition is given by $C_1 = \{1, 2, 3\}$, $C_2 = \{4, 5, 6\}$, $C_3 = \{7, 8, 9\}$ (each of size 3). For a time step t and cardinality constraint of exact-k with $k = 2$, we add

Algorithm 1: `BestFitPartition`

Input: Graph $G = (V, E)$
Output: Clique partition $\mathcal{P} = \{C_1, \ldots, C_m\}$ of V

$\tau \leftarrow$ `EstimateMaxCliqueSize`(G);
$\mathcal{P} \leftarrow \emptyset$;
foreach $v \in V$ *in decreasing degree order* **do**
  best_index $\leftarrow \perp$;
  best_score $\leftarrow -\infty$;
  `// find best-fit clique for` v
  foreach $C_i \in \{C \in \mathcal{P} \mid \forall u \in C, \{u, v\} \in E\}$ **do**
    score $\leftarrow \tau - |C_i|$;
    if *score* $>$ *best_score* **then**
      best_index $\leftarrow i$;
      best_score $\leftarrow$ *score*;
  if *best_index* $\neq \perp$ **then**
    $C_{\textit{best\_index}} \leftarrow C_{\textit{best\_index}} \cup \{v\}$;
  else
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\{v\}\}$;
return $\mathcal{P}$;



Vertex-level: $\sum_{v \in V} x_v^t = k$
Clique-level: $\sum_{i=1}^{m} c_i^t = k$
Linking: $c_i^t \leftrightarrow \bigvee_{v \in C_i} x_v^t$

Figure 3: Example graph with three 3-vertex cliques (solid triangles) arranged as an equilateral triangle; solid edges indicate additional inter-clique adjacency.

$$c_1^t \leftrightarrow (x_1^t \vee x_2^t \vee x_3^t),$$
$$c_2^t \leftrightarrow (x_4^t \vee x_5^t \vee x_6^t),$$
$$c_3^t \leftrightarrow (x_7^t \vee x_8^t \vee x_9^t),$$

and then enforce $c_1^t + c_2^t + c_3^t = 2$. This replaces an exact-2 constraint over 9 variables with an exact-2 constraint over 3 variables while preserving correctness.

3.2 Pruning Constraints

We add redundant constraints that strengthen unit propagation and narrow the search space. Let $d = |I_s \setminus I_g|$ denote a lower bound on the shortest reconfiguration length: every token that is in I_s but not in I_g must move at least once, and each step moves exactly one token.

Token fixation. When testing bound $\ell = d$, every token on a vertex in $I_s \cap I_g$ must remain in place throughout any shortest sequence, since removing and restoring it would require at least two extra steps. Likewise, no vertex outside $I_s \cup I_g$ can hold a token. We add the unit clauses

$$\bigwedge_{v \in I_s \cap I_g} \bigwedge_{t=0}^{d} x_v^t \quad \wedge \quad \bigwedge_{v \notin I_s \cup I_g} \bigwedge_{t=0}^{d} \neg x_v^t.$$

Algorithm 2: `Rebalancing`

Input: Clique partition $\mathcal{P} = \{C_1, \ldots, C_m\}$, and Graph $G = (V, E)$
Output: Rebalanced clique partition $\mathcal{P}$

repeat
  changed $\leftarrow$ **false**;
  foreach $C_s \in \mathcal{P}$ *in increasing order of* $|C_s|$ **do**
    foreach $v \in C_s$ **do**
      $\mathcal{F} \leftarrow \{C \in \mathcal{P} \mid \forall u \in C, \{u, v\} \in E \wedge |C| \geq |C_s|\}$;
      if $\mathcal{F} \neq \emptyset$ **then**
        $C_s \leftarrow C_s \setminus \{v\}$;
        $C_t \leftarrow \arg\min_{C \in \mathcal{F}} |C|$;
        $C_t \leftarrow C_t \cup \{v\}$;
        changed $\leftarrow$ **true**;
until *changed* $=$ ***false***;
$\mathcal{P} \leftarrow \{C \in \mathcal{P} \mid C \neq \emptyset\}$;
return $\mathcal{P}$;

Monotonicity. Again at $\ell = d$, for a goal-only vertex $v \in I_g \setminus I_s$ a token that has arrived cannot leave without exceeding the lower bound; symmetrically, for a start-only vertex $v \in I_s \setminus I_g$ a departed token cannot return. We enforce

$$\bigwedge_{v \in I_g \setminus I_s} \bigwedge_{t=0}^{d-1} (x_v^t \rightarrow x_v^{t+1}) \quad \wedge \quad \bigwedge_{v \in I_s \setminus I_g} \bigwedge_{t=0}^{d-1} (\neg x_v^t \rightarrow \neg x_v^{t+1}).$$

Both constraints are sound only at $\ell = d$ and are dropped when $\ell > d$. In Algorithm 3, we write H_d for these constraints, which are active only at $\ell = d$.

Clause complexity. Table 1 summarizes clauses by component. Here $n = |V|$, m is the number of cliques in $\mathcal{P}$, $k = |I_s| = |I_g|$, $d = |I_s \setminus I_g|$, and cardinality constraints (sequential counter (Sinz 2005) / totalizer (Bailleux and Boufkhad 2003)) on n' variables with bound k' produce $O(n'k')$ clauses. The majority of clauses in the basic formula Ψ_ℓ is $O(\ell n k)$ from the exact-k token constraint at each step. The clique-based formula Ψ_ℓ^{CC} replaces this with $O(\ell m k)$, yielding a reduction factor of n/m.

3.3 Overall Algorithm

Algorithm 3 describes the main BMC loop of SRIP. The loop increments ℓ from the lower bound d until a solution is found or timeout; $\ell_{\max}$ is set large enough that the timeout is the only effective bound. The key mechanism is to exploit *block literals* and *assumptions* for incremental SAT solving.

For each bound ℓ, we introduce a fresh Boolean variable b_ℓ called a *block literal*. Every clause belonging to $Goal(X^\ell)$ at bound ℓ is augmented with $\neg b_\ell$, so that these clauses are logically active only when b_ℓ is assumed true. At iteration ℓ, the SAT solver is called with assumption b_ℓ, which activates the goal at step ℓ. If the solver returns UNSAT, we add the unit clause $(\neg b_\ell)$ as a permanent hard clause to disable the goal at ℓ, and proceed to $\ell + 1$. Since

Component	Per step	Total
Basic formula Ψ_ℓ		
$Init + Goal$	—	$O(n)$
Independence	$O(\|E\|)$	$O(\ell\|E\|)$
Exact-k (n vars)	$O(nk)$	$O(\ell\|V\|k)$
$Jump$ (link + exact-1)	$O(n)$	$O(\ell n)$
Total		$O\big(\ell(\|E\| + nk)\big)$
Clique-based formula Ψ_ℓ^{CC}		
Clique linking	$O(n)$	$O(\ell n)$
Exact-k (m vars)	$O(mk)$	$O(\ell mk)$
Total		$O\big(\ell(\|E\| + n + mk)\big)$
Pruning constraints H_d		
Token fixation	$O(n)$	$O(nd)$
Monotonicity	$O(d)$	$O(d^2)$

Table 1: Clause counts by constraint component.

clauses from previous iterations persist, only the new step's variables, transition constraints, and blocked goal clauses need to be added—no re-encoding is required.

The pruning constraints H_d use a dedicated block literal h_d that is assumed true at $\ell = d$ and permanently set to false (via a unit clause $\neg h_d$) at $\ell = d + 1$, ensuring that H_d is active only during the first iteration.

In the implementation, written in Rust, we use CaDiCaL (Biere et al. 2024) (ver. 3.0) as the backend SAT solver. We use the sequential counter encoding (Sinz 2005) for exact-1 constraints and the totalizer encoding (Bailleux and Boufkhad 2003) for clique-level exact-k constraints, as these choices empirically perform best in our setting.

Algorithm 3: Overall Implementation of SRIP

Input: ISRP instance (G, I_s, I_g) with $|I_s| = |I_g| = k$
Output: Shortest sequence, or UNSAT

$\mathcal{P} \leftarrow$ `BestFitPartition`(G)
$\mathcal{P} \leftarrow$ `Rebalancing`$(\mathcal{P}, G)$
$d \leftarrow |I_s \setminus I_g|$ `// lower bound`
$\Psi \leftarrow Init(X^0) \wedge \bigwedge_{t=0}^{d-1} \big(Token^{CC}(X^t, \mathcal{P}) \wedge Jump(X^t, X^{t+1})\big)$
Create block literal h_d
for $\ell \leftarrow d$ **to** $\ell_{\max}$ **do**
  Create block literal b_ℓ
  $\Psi_\ell \leftarrow \Psi \wedge Token^{CC}(X^\ell, \mathcal{P}) \wedge Jump(X^{\ell-1}, X^\ell) \wedge \big(b_\ell \rightarrow Goal(X^\ell)\big)$
  $\mathcal{A} \leftarrow \{b_\ell\}$ `// set assumption`
  if $\ell = d$ **then**
    $\Psi_\ell \leftarrow \Psi_\ell \wedge (h_d \rightarrow H_d)$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{h_d\}$ `// add assumption`
  if $\ell = d + 1$ **then**
    $\Psi_\ell \leftarrow \Psi_\ell \wedge (\neg h_d)$ `// drop` H_d
  result $\leftarrow$ SolveSAT($\mathcal{A}$, Ψ_ℓ)
  if *result* = SAT **then**
    return extract a TJ reconfiguration sequence from the model found
  $\Psi \leftarrow \Psi_\ell \wedge (\neg b_\ell)$ `// drop` $Goal(X^\ell)$
return UNSAT

4 Experimental Evaluation

This section presents an experimental evaluation of SRIP. First, we provide a pre-analysis of the clique-partition-based cardinality encoding. Then, we provide ablation studies to demonstrate the effectiveness of our proposed techniques. Finally, we compare SRIP against state-of-the-art ISRP solvers on the CoRe Challenge benchmark.

4.1 Experimental Setup

Environment. We carry out all experiments on a cluster with Intel(R) Xeon(R) Platinum 8480+ (2.00GHz) and 128GB RAM, running Ubuntu 20.04. We allow solvers to use a single core per instance. Timelimit is 1800 seconds.

Benchmark: Clique. We create a family of *clique* instances parameterized by $n \in \{5, 10, 15, \ldots, 200\}$ (40 instances total). For each n, the graph $G_n = (V_n, E_n)$ consists of n cliques of size n, connected by a cycle:

- $V_n = \{v_{i,j} \mid 0 \le i < n, 0 \le j < n\}$.
- $E_n = E_{\text{clique}} \cup E_{\text{cycle}}$ where:

$$E_{\text{clique}} = \{\{v_{i,j}, v_{i,k}\} \mid 0 \le i < n, 0 \le j < k < n\},$$
$$E_{\text{cycle}} = \{\{v_{i,0}, v_{(i+1 \bmod n),1}\} \mid 0 \le i < n\}.$$

As the vertex set $\{v_{i,j} \mid 0 \le j < n\}$ induces a clique in G_n, we refer to it as clique i. The cycle edges connect "hub" vertices $v_{i,0}$ and $v_{i,1}$ across adjacent cliques, each having degree n (internal $n-1$ plus external 1). The ISRP instance has:

- Start: $I_s = \{v_{i,0} \mid 0 \le i < n\}$.
- Goal: $I_g = \{v_{0,2}\} \cup \{v_{i,1} \mid 1 \le i < n\}$.

For I_s, each clique contributes one hub vertex, forming an independent set of size n. For I_g, clique 0 contributes non-hub vertex $v_{0,2}$, while each other clique contributes one hub vertex, forming an independent set of size n. This construction ensures $I_s \cap I_g = \emptyset$ and both are independent sets of size n. Figure 3 illustrates the $n = 3$ case.

Benchmark: CoRe Challenge We use the CoRe Challenge benchmark (Soh et al. 2024a), which consists of 693 TJ instances across 11 series: `color04` (from DIMACS graph coloring instances), `queen` (from queen's graphs corresponding to N-queens problems), `grid` (square grid graphs with checkered-flag patterns), `handcrafted` (small YES/NO instances for testing), `ph-isr` (from pigeon hole problems), `square` and `power` (quadratic and exponential reconfiguration lengths, respectively), `sp` (converted from shortest path reconfiguration problems with

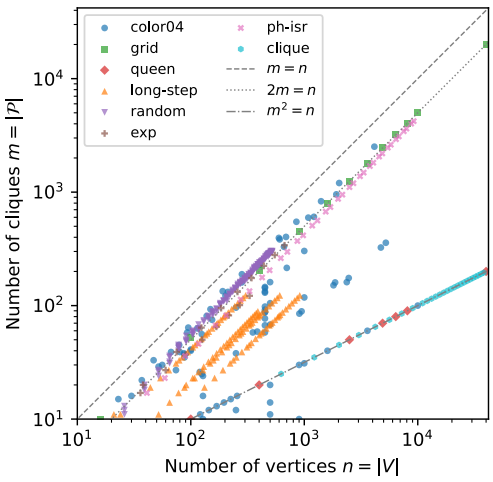


Figure 4: Vertices n vs. cliques m for BestFit+Rebalancing on official CoRe Challenge and handcrafted `clique` instances (log–log scale). Each point is a unique (n, m) pair; points below the dashed $m = n$ line indicate compression (median $m/n \approx 0.45$).

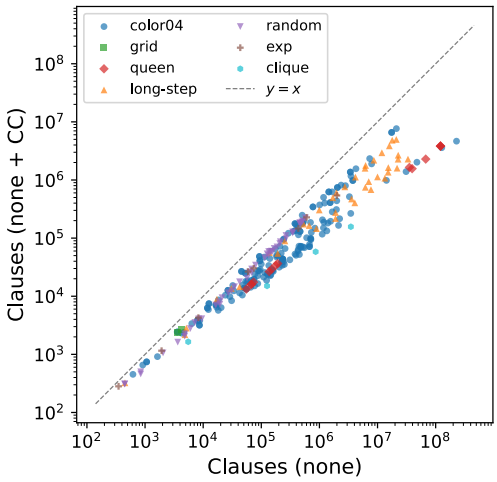


Figure 5: Clause count comparison: none vs. +CC on 296 instances solved by both (including 4 handcrafted `clique` instances). Points below the diagonal indicate clause reduction by the clique partition.

exponential lengths), `exp` (converted from hard 3-SAT and list coloring reconfiguration problems with exponential lengths), `wide` (high branching factor), and `random` (randomly generated graphs). In our evaluation (Table 2), we group these into 7 categories: `grid` includes `handcrafted`, and `long-step` aggregates `square`, `power`, `sp`, and `wide`—series whose reconfiguration lengths grow rapidly with instance size.

Solvers compared. We compare SRIP against three state-of-the-art ISRP solvers: PARIS (Christen et al. 2023) (planning-based), NUASP (Yamada et al. 2024) (ASP-based BMC), and RECONF. (Froleyks, Yu, and Biere 2022) (SAT-based). PARIS behaves differently from SRIP, NUASP, and RECONF. in that it does not always guarantee optimality. Specifically, PARIS outputs a provably optimal solution once optimality is certified; otherwise, it returns its best solution before the time limit (*best-so-far*). Following the official CoRe Challenge 2022/2023 shortest-track rule[1,2], we define `#best` as the number of instances for which a solver attains the minimum length of reconfiguration sequence within the solver set considered in each comparison. For `#best`, we compare, for each instance, the lengths of reconfiguration sequences returned by all solvers within the time limit, and assign $+1$ to every solver attaining the minimum (ties are counted for all tied solvers). Since PARIS returns its *best-so-far* solution at the time limit when optimality is not certified, that returned solution is included in this comparison. For the ablation study, we also compare four SRIP variants: +H_d+CC (all techniques), +CC (without H_d), +H_d (without CC), and none (no technique).

4.2 Pre-analysis: Clique Compression

In this subsection, we analyze the official CoRe Challenge instances together with the handcrafted `clique` instances.

Reduction ratio. Figure 4 plots the number of vertices n against the number of cliques m produced by our greedy heuristic on CoRe Challenge instances and the handcrafted `clique` instances. Each point corresponds to a unique (n, m) pair. The median ratio is $m/n \approx 0.45$; on the largest instances ($n = 40\,000$) the ratio drops to $m/n = 0.005$, giving a $200\times$ reduction in the cardinality constraint size.

Clause reduction. Figure 5 compares the number of SAT clauses between the baseline (none) and the clique-partition-based cardinality encoding (+CC) for the 296 instances solved by both, including 4 handcrafted `clique` instances. All points lie below the diagonal, confirming that the clique partition consistently reduces the clause count in our formulation. The median reduction ratio is 0.28 for color04, 0.19 for queen, 0.25 for long-step, and 0.094 for clique instances, demonstrating that CC compresses the exact-k cardinality encoding by a factor of 3–10$\times$ across diverse families.

Sensitivity to partition quality. To assess how sensitive end-to-end performance is to the quality of the clique partition, we replaced BestFitPartition + Rebalancing with a naive greedy partition while keeping the rest of SRIP unchanged. On the official 693-instance benchmark (1800 s timeout), the number of solved instances dropped from 477 to 465. The preprocessing cost was negligible in both settings (about 0.002 s in our logs), so the main effect appears in the SAT encoding/solving phase. BestFitPartition + Rebalancing produced clique partitions with fewer cliques on 546 out of 693 instances, with median clique count 82 versus 95 for the naive greedy partition, and substantially more balanced clique sizes, with mean within-instance clique-size standard deviation 1.53 versus 2.78. This suggests that partition quality matters, including balance beyond clique count.

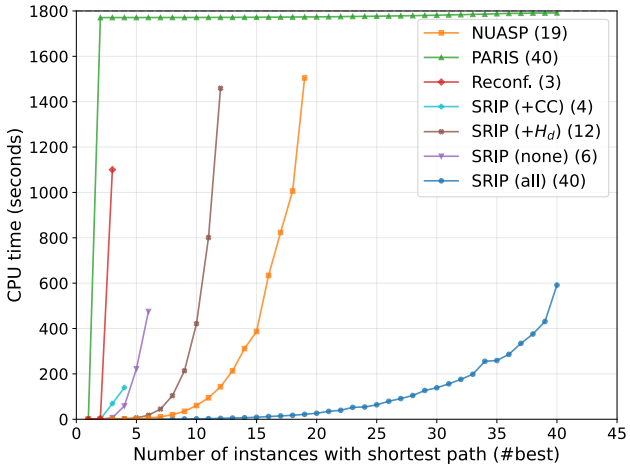


Figure 6: Ablation study on 40 `clique` instances. Removing techniques from "all" progressively degrades performance. Each point counts an instance on which the solver attains the shortest length (#best, with ties counted for all tied solvers).

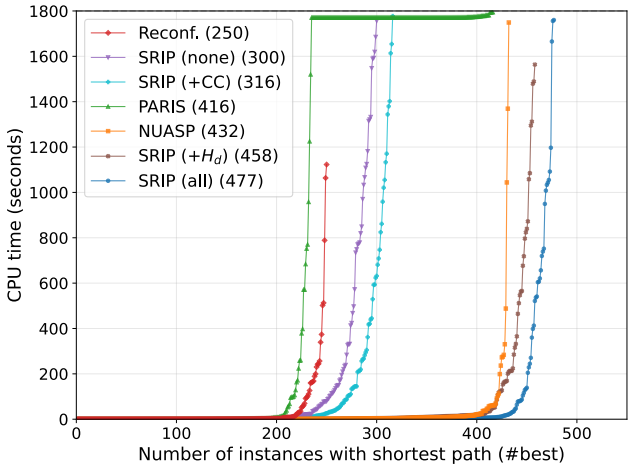


Figure 7: Cactus plot on the CoRe Challenge benchmark. SRIP (all) uses both techniques (+H_d and CC); (+H_d) uses only pruning constraints; (+CC) uses only clique-partition-based cardinality encoding; (none) uses neither technique. Each point is an instance where a solver attained the shortest reconfiguration sequence (ties count for all tied solvers).

Exact minimum clique partitions on `color04`. We also compared our heuristic with an exact minimum clique partition formulation using OR-Tools CP-SAT for complement-graph coloring on all 101 unique graphs in the `color04` benchmark set under a 1800-second timeout. The exact solver proved optimality on only 21 graphs; for the remaining graphs, it returned feasible but unverified partitions on 68 and timed out or exceeded memory limits on 12. At the same time, BestFitPartition + Rebalancing was not uniformly worse: on 45 graphs it either matched the clique count returned by CP-SAT (19), found fewer cliques than CP-SAT's best feasible partition within the time limit (14), or was the only method to return any partition within the time and memory limits (12). These results confirm that minimizing the number of cliques exactly is expensive and not necessarily the right objective for the full pipeline, where clique-size balance and preprocessing cost also matter; bounded-exact or multi-objective partitioning methods are an interesting direction for future work.

4.3 Ablation Studies on `clique`

We first conduct a proof-of-concept for our proposed techniques (H_d and CC) on `clique` which consists of connected cliques and the length of shortest reconfiguration sequences are at the lower bound d.

Results. We evaluated the effect of each technique on 40 dedicated `clique` instances. Figure 6 shows the cactus plot comparing different ablation variants. The all-techniques configuration +H_d+CC achieves the shortest path on all 40 instances, while removing techniques progressively degrades performance. PARIS also solved all 40 instances, but the optimality of its returned solutions is not guaranteed.

The results reveal several key findings. First, +H_d+CC achieves the shortest path on all 40 instances, demonstrating that the combination of the pruning constraints and clique-partition-based cardinality encoding is highly effective on this benchmark family. Second, removing CC dramatically degrades performance: +H_d achieves only 12 and none achieves only 6 #best instances, demonstrating the critical importance of the clique-partition-based cardinality encoding on graphs with dense clique structure. Conversely, +CC without H_d achieves only 4 #best instances, showing that both techniques are necessary. Third, NUASP achieves 19 #best instances, outperforming variants without both techniques but falling short of +H_d+CC. Finally, RECONF. achieves only 3 #best instances, suggesting that its encoding is not well-suited for this benchmark family.

4.4 Comparison with State-of-the-Art Solvers on CoRe Challenge Benchmark

Results as cactus plot. Figure 7 shows a cactus plot comparing the four main solvers along with SRIP ablation variants on the CoRe Challenge benchmark. We can say that SRIP (+H_d+CC) is the fastest solver on the largest number of instances, followed by NUASP, PARIS, and RECONF.. As noted above, PARIS returns *best-so-far* just before the time limit.

Breakdowns by category, size, and path length. We further break down results by benchmark category, vertex count, and the length of reconfiguration sequences. Let L denote the minimum length of reconfiguration sequence attained within the compared solver set (SRIP, NUASP, PARIS, and RECONF.). The following tables are generated from the full logs; for L we consider only instances for which at least one solver produced a validated solution.

Table 2 shows a breakdown analysis by benchmark category. SRIP achieves the best results on long-step and random categories, while PARIS performs best on queen in-

Category	#Inst	RECONF.	PARIS	NUASP	SRIP
color04	202	177	196	**200**	**200**
grid	49	**2**	**2**	**2**	**2**
queen	48	24	**46**	24	21
long-step	138	8	22	7	**47**
random	200	34	140	194	**199**
exp	20	5	**9**	5	8
ph-isr	36	0	**1**	0	0
Total	693	250	416	432	**477**

Table 2: Breakdown by benchmark category (#best).

$\|V\|$	#Inst	RECONF.	PARIS	NUASP	SRIP
$\leq$100	136	101	**123**	107	**123**
101–500	373	99	208	249	**278**
501–1000	68	21	30	35	**36**
1001–5000	72	27	37	**38**	34
>5000	44	2	**18**	3	6

Table 3: Breakdown by vertex count (#best).

Length	#Inst	RECONF.	PARIS	NUASP	SRIP
1	61	**61**	**61**	**61**	**61**
2–5	79	**79**	**79**	78	**79**
6–10	60	**60**	**60**	**60**	**60**
11–20	53	43	49	**53**	50
21–50	91	5	**78**	73	69
>50	169	2	89	107	**158**

Table 4: Breakdown by transition length (#best).

$\|V\| \times L$	#Inst	RECONF.	PARIS	NUASP	SRIP
$\leq$100	24	**24**	**24**	**24**	**24**
101–500	69	**69**	**69**	**69**	**69**
501–1000	60	**60**	**60**	**60**	**60**
1001–5000	101	73	97	98	**100**
5001–10000	42	5	34	35	**38**
>10000	217	19	132	146	**186**

Table 5: Breakdown by $|V| \times L$ (#best).

stances. The long-step category, which contains instances with long reconfiguration sequences, highlights SRIP's strength: it achieves #best on 47 instances compared to 22 for PARIS and only 7–8 for NUASP and RECONF..

Table 3 shows a breakdown analysis by the number of vertices. SRIP achieves the best results on medium-sized instances ($101 \leq |V| \leq 1000$), whereas PARIS attains more #best results on very large instances ($|V| > 5000$), likely because PARIS can return a *best-so-far* solution even when optimality cannot be certified within the time limit. This apparent advantage on very large instances, however, is concentrated mainly in the `queen` family rather than being a general effect of instance size. Of the 18 large instances on which PARIS attains #best, 16 belong to `queen`, and the remaining two are `wap04a` instances on which SRIP and NUASP attain the same shortest length. Moreover, all 14 large instances solved by PARIS but not by SRIP are queen instances. Beyond these #best cases, the only other non-queen large instances solved by PARIS are the two `qg.order100` instances; there, PARIS returns *best-so-far* solutions, whereas SRIP certifies the shortest sequences and therefore attains #best. These observations suggest that the apparent advantage of PARIS on very large instances is driven primarily by the queen family.

Table 4 shows a breakdown analysis by the length of reconfiguration sequences. All solvers perform similarly on instances that admit short reconfiguration sequences ($L \leq 10$), but SRIP shows a clear advantage on instances requiring longer reconfiguration sequences ($L > 50$), achieving #best on 158 out of 169 instances.

Table 5 reports a combined scalarization by $|V| \times L$ (the number of vertices times the length of reconfiguration sequences) as a simple proxy of instance difficulty. All solvers perform equally on easier instances ($|V| \times L \leq 1000$). SRIP achieves the best results on the most difficult instances ($|V| \times L > 10000$), achieving #best on 186 out of 217 instances.

Ablation study on CoRe Challenge Benchmark. We evaluate the contribution of pruning constraints (H_d: token fixation and monotonicity) and clique-partition-based cardinality encoding (CC) on the 693-instance benchmark. Table 6 shows the number of instances solved for each of the four combinations.

The baseline (none) solves 300 instances using only the core incremental BMC loop with lower-bound initialization. H_d alone provides a substantial gain of +158 instances, while CC alone adds only 16. However, when combined, H_d+CC solves 477 instances (+19 over H_d alone), demonstrating a synergistic effect: CC reduces the number of clauses (Figure 5), making the SAT instances easier for the solver to handle within the time limit.

5 Related Work

Several solver paradigms have been applied to ISRP. PARIS (Christen et al. 2023) models ISRP as an automated planning problem by encoding token movements as PDDL (Planning Domain Description Language) actions. In its single-engine configuration for the shortest track, it uses an anytime landmark-based planner that runs greedy best-first search with a landmark count heuristic, followed by weighted A* with decreasing weights to find progressively shorter plans. NUASP (Yamada et al. 2024) formulates ISRP as an Answer Set Programming (ASP) problem built on the *clingo* solver and uses a BMC-style incremental grounding–solving loop. RECONF. (ReconfAIGERation) (Froleyks, Yu, and Biere 2022) encodes ISRP as an AIGER circuit and solves it using hardware model checkers: the bounded model checker CaMiCaL (with the incremental SAT solver CaDiCaL (Biere et al. 2024)) for shortest-path finding, and ABC (Brayton and Mishchenko 2010) (which implements IC3/PDR (Bradley 2011)) for reacha-

	none	+CC	+H_d	+H_d+CC
$\ell = d$	208	215	367	374
$\ell > d$	92	101	91	103
Total	300	316	458	477

Table 6: Ablation study on the CoRe Challenge benchmark.

bility and unsolvability detection. ZDD-based methods (Ito et al. 2023) represent the family of reachable independent sets as a zero-suppressed binary decision diagram (ZDD) and iteratively expand it by one-step token jumps; this approach can both find shortest paths and prove unsolvability, but its memory footprint limits scalability to large graphs. SRIP differs from all of the above by combining incremental SAT solving, clique-partition-based cardinality encoding, and pruning constraints in a single framework.

6 Artifact and Reproducibility

To support reproducibility and future comparisons, we release a complete artifact package comprising the full Rust implementation of SRIP, the 693 CoRe Challenge instances, 40 handcrafted clique-cycle instances, experiment scripts for running and parsing experiments and generating tables and figures, full raw logs, and a Dockerfile. The artifact is publicly available at:
https://github.com/TakehideSoh/KR2026.

7 Conclusion and Future Work

We have presented SRIP, a SAT-based bounded model checking system for the Independent Set Reconfiguration Problem (ISRP) under the Token Jumping rule. Our key technical contribution is two-fold: (i) *clique-partition-based cardinality encoding*, which reduces the exact-k cardinality constraint from $O(\ell|V|k)$ to $O(\ell mk)$ clauses by enforcing cardinality at the clique level, and (ii) pruning constraints (H_d), which strengthen propagation and reduce the search space. Working together with incremental SAT solving and block literals, these two techniques yield a clear synergistic effect: SRIP finds shortest reconfiguration lengths on 477 out of 693 CoRe Challenge instances, exceeding the previous best solver NUASP by 45 instances. Ablation results support this view, showing that both components contribute and that their combination performs best.

Several directions remain for future work. First, SRIP currently targets shortest reconfiguration sequences; extending it to prove *unreachability* (i.e., no reconfiguration sequence exists) is an important next step. Second, supporting the *Token Sliding* rule, which restricts moves to adjacent vertices, would broaden the applicability of the approach. Third, it would be interesting to study bounded-exact or multi-objective clique partitioning methods that balance clique count, clique-size regularity, and preprocessing cost.

Acknowledgements

We thank the reviewers for their careful reading and helpful comments. This work was initiated under the collaborative research supported by JSPS KAKENHI (JP23K11047, 23K12974, JP24H00686, JP24H00690, JP25K03097).

AI Declaration

AI tools were used to assist with English polishing, LaTeX editing, and the maintenance of figure/table generation scripts during the preparation of the camera-ready version. All technical content and experimental results are the original work of the authors. The final wording was reviewed and validated by the authors, who take full responsibility for the paper.

References

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF encoding of Boolean cardinality constraints. In *Proc. of the 9th International Conference on Principles and Practice of Constraint Programming (CP 2003)*, volume 2833 of *LNCS*, 108–122. Springer.

Biere, A.; Cimatti, A.; Clarke, E. M.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In Cleaveland, R., ed., *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99*, volume 1579 of *Lecture Notes in Computer Science*, 193–207. Springer.

Biere, A.; Faller, T.; Fazekas, K.; Fleury, M.; Froleyks, N.; and Pollitt, F. 2024. CaDiCaL 2.0. In Gurfinkel, A., and Ganesh, V., eds., *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, 133–152. Springer.

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

Bradley, A. R. 2011. SAT-based model checking without unrolling. In Jhala, R., and Schmidt, D. A., eds., *Verification, Model Checking, and Abstract Interpretation, 12th International Conference, VMCAI 2011*, volume 6538 of *Lecture Notes in Computer Science*, 70–87. Springer.

Brayton, R. K., and Mishchenko, A. 2010. ABC: an academic industrial-strength verification tool. In Touili, T.; Cook, B.; and Jackson, P., eds., *Computer Aided Verification, 22nd International Conference, CAV 2010*, volume 6174 of *Lecture Notes in Computer Science*, 24–40. Springer.

Brewster, R. C.; McGuinness, S.; Moore, B. R.; and Noel, J. A. 2016. A dichotomy theorem for circular colouring reconfiguration. *Theoretical Computer Science* 639:1–13.

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., eds., *Proceedings of the 26th European*

Conference on Artificial Intelligence (ECAI 2023), volume 372 of *Frontiers in Artificial Intelligence and Applications*, 453–460. IOS Press.

Froleyks, N.; Yu, E.; and Biere, A. 2022. ReconfAIGERation entering core challenge 2022. In Soh, T.; Okamoto, Y.; and Ito, T., eds., *Core Challenge 2022: Solver and Graph Descriptions*. arXiv. 29–31. Available at arXiv:2208.02495.

Gopalan, P.; Kolaitis, P. G.; Maneva, E. N.; and Papadimitriou, C. H. 2009. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing* 38(6):2330–2355.

Hearn, R. A., and Demaine, E. D. 2009. *Games, puzzles and computation*. A K Peters.

Inoue, T.; Takano, K.; Watanabe, T.; Kawahara, J.; Yoshinaka, R.; Kishimoto, A.; Tsuda, K.; Minato, S.; and Hayashi, Y. 2014. Distribution loss minimization with guaranteed error bound. *IEEE Transactions on Smart Grid* 5(1):102–111.

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 Ciré, A. A., ed., *Proceedings of the 20th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2023)*, volume 13884 of *Lecture Notes in Computer Science*, 167–183. Springer.

Ito, T.; Ono, H.; and Otachi, Y. 2015. Reconfiguration of cliques in a graph. In Jain, R.; Jain, S.; and Stephan, F., eds., *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation (TAMC 2015)*, volume 9076 of *Lecture Notes in Computer Science*, 212–223. Springer.

Kaminski, M.; Medvedev, P.; and Milanic, M. 2012. Complexity of independent set reconfigurability problems. *Theoretical Computer Science* 439:9–15.

Knuth, D. 2011. *The art of computer programming Vol.4A, Combinatorial algorithms, Part 1*. Addison-Wesley.

Mohar, B., and Salas, J. 2009. A new Kempe invariant and the (non)-ergodicity of the Wang–Swendsen–Kotecký algorithm. *Journal of Physics A: Mathematical and Theoretical* 42(22):225204.

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

Sinz, C. 2005. Towards an optimal CNF encoding of boolean cardinality constraints. In van Beek, P., ed., *Proc. of the 11th International Conference on Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *LNCS*, 827–831. Springer.

Soh, T.; Tanjo, T.; Okamoto, Y.; and Ito, T. 2023. Core challenge 2023: Solver and graph descriptions. *CoRR* abs/2310.17136.

Soh, T.; Tanjo, T.; Okamoto, Y.; and Ito, T. 2024a. CoRe challenge 2022/2023: Empirical evaluations for independent set reconfiguration problems (extended abstract). In Felner, A., and Li, J., eds., *Proceedings of the Seventeenth International Symposium on Combinatorial Search (SOCS 2024)*, 285–286. AAAI Press.

Soh, T.; Watanabe, T.; Kawahara, J.; Suzuki, A.; and Ito, T. 2024b. Scalable hard instances for independent set reconfiguration. In Liberti, L., ed., *Proc. The 22nd International Symposium on Experimental Algorithms, SEA 2024*, volume 301 of *LIPIcs*, 26:1–26:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

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

Takaoka, A. 2018. Complexity of Hamiltonian cycle reconfiguration. *Algorithms* 11(9):140.

Yamada, Y.; Banbara, M.; Inoue, K.; Schaub, T.; and Uehara, R. 2024. Combinatorial reconfiguration with answer set programming: Algorithms, encodings, and empirical analysis. In Uehara, R.; Yamanaka, K.; and Yen, H., eds., *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024)*, volume 14549 of *Lecture Notes in Computer Science*, 242–256. Springer.