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] Takehiro Ito[4]

July 22, 2026

[1]Nagoya University, Nagoya, Japan
[2]Kobe University, Kobe, Japan
[3]Hokkaido University, Sapporo, Japan
[4]Tohoku University, Sendai, Japan

International Conference on Principles of Knowledge Representation and Reasoning (KR 2026)
KR in the Wild — Session: SAT Encoding

What is Combinatorial Reconfiguration (CR)?

Input

1. Combinatorial problem P
2. Reconfiguration rule
3. X_s: solution of P
4. X_g: solution of P

Output

A sequence of solutions of P from X_s to X_g while satisfying a given reconfiguration rule.

What is Combinatorial Reconfiguration (CR)?

Input

1. Combinatorial problem P
2. Reconfiguration rule
3. X_s: solution of P
4. X_g: solution of P

Output

A sequence of solutions of P from X_s to X_g while satisfying a given reconfiguration rule.

Solution Space Graph

- vertices: solutions of P
- edges: reconfiguration rule

What is Combinatorial Reconfiguration (CR)?

Input

1. Combinatorial problem P
2. Reconfiguration rule
3. X_s: solution of P
4. X_g: solution of P

Output

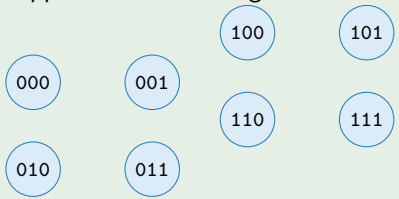
A sequence of solutions of P from X_s to X_g while satisfying a given reconfiguration rule.

Solution Space Graph

- vertices: solutions of P
- edges: reconfiguration rule

Example

Suppose that P has eight solutions.



What is Combinatorial Reconfiguration (CR)?

Input

1. Combinatorial problem P
2. Reconfiguration rule
3. X_s: solution of P
4. X_g: solution of P

Output

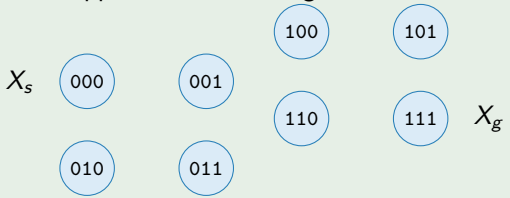
A sequence of solutions of P from X_s to X_g while satisfying a given reconfiguration rule.

Solution Space Graph

- vertices: solutions of P
- edges: reconfiguration rule

Example

Suppose that P has eight solutions.



What is Combinatorial Reconfiguration (CR)?

Input

1. Combinatorial problem P
2. Reconfiguration rule
3. X_s: solution of P
4. X_g: solution of P

Output

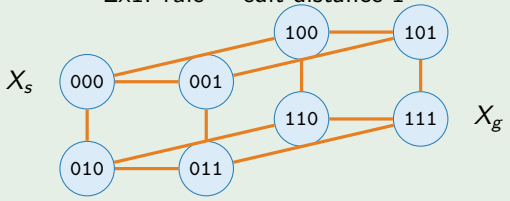
A sequence of solutions of P from X_s to X_g while satisfying a given reconfiguration rule.

Solution Space Graph

- vertices: solutions of P
- edges: reconfiguration rule

Example

Ex1. rule = edit distance 1



Valid sequence: 000 → 001 → 101 → 111

What is Combinatorial Reconfiguration (CR)?

Input

1. Combinatorial problem P
2. Reconfiguration rule
3. X_s: solution of P
4. X_g: solution of P

Output

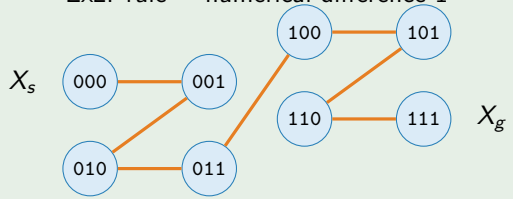
A sequence of solutions of P from X_s to X_g while satisfying a given reconfiguration rule.

Solution Space Graph

- vertices: solutions of P
- edges: reconfiguration rule

Example

Ex2. rule = numerical difference 1



Valid sequence: 000 → 001 → 010 → 011 → 100 → 101 → 110 → 111

Many Reconfiguration Problems Are PSPACE-Complete

Many NP-complete problems have PSPACE-complete reconfiguration counterparts.

- SAT Reconfiguration [Gopalan+, 2009, Mouawad+, 2017]
- Coloring Reconfiguration [Bonsma+, 2009, Brewster+, 2016, Cereceda+, 2011]
- Dominating Set Reconfiguration [Haddadan+, 2016, Suzuki+, 2016]
- Clique Reconfiguration [Ito+, 2015]
- Hamiltonian Cycle Reconfiguration [Takaoka, 2018]
- Set Covering Reconfiguration [Ito+, 2011]
- Independent Set Reconfiguration [Ito+, 2011, Kaminski+, 2012]

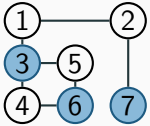
Many Reconfiguration Problems Are PSPACE-Complete

Many NP-complete problems have PSPACE-complete reconfiguration counterparts.

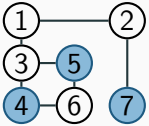
- SAT Reconfiguration [Gopalan+, 2009, Mouawad+, 2017]
- Coloring Reconfiguration [Bonsma+, 2009, Brewster+, 2016, Cereceda+, 2011]
- Dominating Set Reconfiguration [Haddadan+, 2016, Suzuki+, 2016]
- Clique Reconfiguration [Ito+, 2015]
- Hamiltonian Cycle Reconfiguration [Takaoka, 2018]
- Set Covering Reconfiguration [Ito+, 2011]
- Independent Set Reconfiguration [Ito+, 2011, Kaminski+, 2012]

Among them, the Independent Set Reconfiguration Problem (ISRP) is one of the most extensively studied problems in the field.

Independent Set Reconfiguration under Token Jumping Rule (ISRP)



I_s

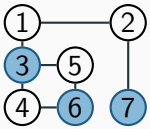


I_g

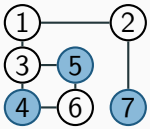
Input of ISRP

- Independent Set Problem: $G = (V, E)$
- Token jumping
  (exact one token can jump)
- I_s: independent set
- I_g: independent set

Independent Set Reconfiguration under Token Jumping Rule (ISRP)



$I_s = I_0$



$I_3 = I_g$

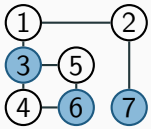
Input of ISRP

- Independent Set Problem: $G = (V, E)$
- Token jumping
  (exact one token can jump)
- I_s: independent set
- I_g: independent set

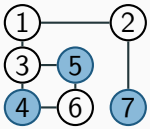
Query.

Is there a solution sequence from I_s to I_g?

Independent Set Reconfiguration under Token Jumping Rule (ISRP)



$I_s = I_0$



$I_3 = I_g$

Input of ISRP

- Independent Set Problem: $G = (V, E)$
- Token jumping
  (exact one token can jump)
- I_s: independent set
- I_g: independent set

Output

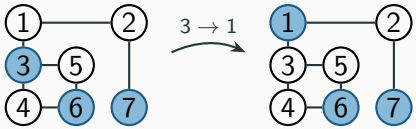
$I_s = I_0$ $\{3, 6, 7\}$ 0010011

$I_g = I_3$ $\{4, 5, 7\}$ 0001101

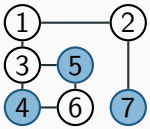
Query.

Is there a solution sequence from I_s to I_g?

Independent Set Reconfiguration under Token Jumping Rule (ISRP)



$I_s = I_0$ I_1



$I_3 = I_g$

Input of ISRP

- Independent Set Problem: $G = (V, E)$
- Token jumping
  (exact one token can jump)
- I_s: independent set
- I_g: independent set

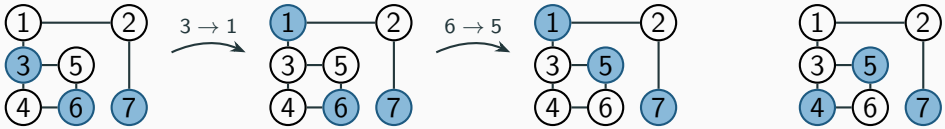
Output

$I_s = I_0$	$\{3, 6, 7\}$	0010011
I_1	$\{1, 6, 7\}$	1000011
$I_g = I_3$	$\{4, 5, 7\}$	0001101

Query.

Is there a solution sequence from I_s to I_g?

Independent Set Reconfiguration under Token Jumping Rule (ISRP)



$I_s = I_0$ I_1 I_2 $I_3 = I_g$

Input of ISRP

- Independent Set Problem: $G = (V, E)$
- Token jumping
  (exact one token can jump)
- I_s: independent set
- I_g: independent set

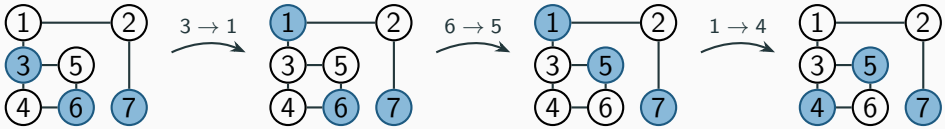
Output

$I_s = I_0$	$\{3, 6, 7\}$	0010011
I_1	$\{1, 6, 7\}$	1000011
I_2	$\{1, 5, 7\}$	1000101
$I_g = I_3$	$\{4, 5, 7\}$	0001101

Query.

Is there a solution sequence from I_s to I_g?

Independent Set Reconfiguration under Token Jumping Rule (ISRP)



$I_s = I_0$ I_1 I_2 $I_3 = I_g$

Input of ISRP

- Independent Set Problem: $G = (V, E)$
- Token jumping
  (exact one token can jump)
- I_s: independent set
- I_g: independent set

Output

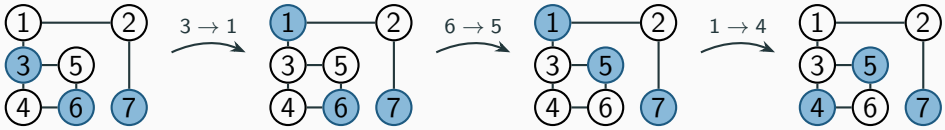
- Yes.

$I_s = I_0$	$\{3, 6, 7\}$	0010011
I_1	$\{1, 6, 7\}$	1000011
I_2	$\{1, 5, 7\}$	1000101
$I_g = I_3$	$\{4, 5, 7\}$	0001101

Query.

Is there a solution sequence from I_s to I_g?

Independent Set Reconfiguration under Token Jumping Rule (ISRP)



Input of ISRP

- Independent Set Problem: $G = (V, E)$
- Token jumping
  (exact one token can jump)
- I_s: independent set
- I_g: independent set

Query.

Is there a solution sequence from I_s to I_g?

Output

- Yes.

$I_s = I_0$	$\{3, 6, 7\}$	0010011
I_1	$\{1, 6, 7\}$	1000011
I_2	$\{1, 5, 7\}$	1000101
$I_g = I_3$	$\{4, 5, 7\}$	0001101

ISRP is PSPACE-complete.

CoRe Challenges Established ISRP Benchmarks and Solvers

- Until around 2020, studies on ISRP focused mainly on theoretical complexity.
- The international CoRe Challenges were held in 2022 and 2023, establishing shared benchmark instances and state-of-the-art practical solvers.
- The competitions compared solvers based on AI planning [Christen+, 2023], SAT-based BMC [Froleyks+, 2022], ASP-based BMC (recongo) [Yamada+, 2024], decision diagrams [Ito+, 2023], and BFS-based search.

In 2023, recongo produced a shortest reconfiguration sequence for 433 instances and ranked first in the single-engine #Shortest category.

Goal: Find Shortest ISRP Sequences at Scale

Goal

Develop a high-performance ISRP solver for finding shortest reconfiguration sequences.

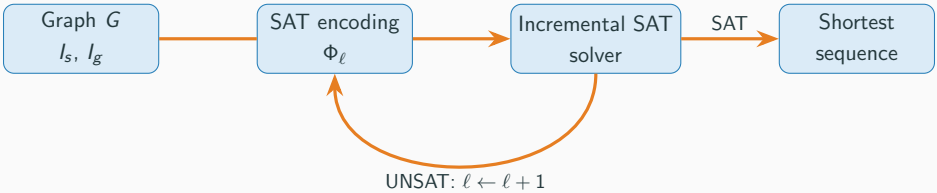
Methodology

- Use bounded model checking (BMC) with incremental SAT solving.
- Incremental SAT alone is insufficient: the encoding suffers from clause explosion as the bound and graph size increase.

Key ideas

1. Exploit the graph's clique structure to reduce the number of clauses.
2. Add optional sound constraints that prune redundant moves at the lower bound.

SRIP Uses Incremental SAT-Based BMC



1. Encode the input (G, I_s, I_g) as Φ_ℓ and pass it to the incremental SAT solver.
2. If UNSAT, increment ℓ, extend the encoding, and repeat while reusing clauses and learned information.
3. If SAT, extract a sequence at the current bound; the first SAT answer is shortest.

SAT encoding for a sequence of length ℓ

Let x_v^t be true iff vertex v has a token at time step t.

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

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

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

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

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

SAT encoding for a sequence of length ℓ

Let x_v^t be true iff vertex v has a token at time step t.

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

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

It ensures the initial and goal states.

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

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

SAT encoding for a sequence of length ℓ

Let x_v^t be true iff vertex v has a token at time step t.

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

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

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

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

It ensures that $|I_s| = |I_g| = k$ tokens consist an independent set in every state.

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

SAT encoding for a sequence of length ℓ

Let x_v^t be true iff vertex v has a token at time step t.

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

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

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

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

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

It enforces the token-jumping rule.

SAT encoding for a sequence of length ℓ

Let x_v^t be true iff vertex v has a token at time step t.

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

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

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

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

This part requires $O(\ell|V|k)$ clauses.

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

The Exact-k Constraint Dominates Clause Growth

Let n be the number of vertices $|V|$, and m be the number of elements of clique partition of G.

Component	Per step	Total
Basic formula Φ_ℓ		
Init + *Goal*	—	$O(n)$
Independence	$O(\|E\|)$	$O(\ell\|E\|)$
Exact-k (n vars)	$O(nk)$	$O(\ell nk)$
Jump (link + exact-1)	$O(n)$	$O(\ell n)$
Total		$O\big(\ell(\|E\| + nk)\big)$

The Exact-k Constraint Dominates Clause Growth

Let n be the number of vertices $|V|$, and m be the number of elements of clique partition of G.

Component	Per step	Total
Basic formula Φ_ℓ		
Init + *Goal*	—	$O(n)$
Independence	$O(\|E\|)$	$O(\ell\|E\|)$
Exact-k (n vars)	$O(nk)$	$O(\ell nk)$
Jump (link + exact-1)	$O(n)$	$O(\ell n)$
Total		$O\big(\ell(\|E\| + nk)\big)$
Clique-based formula Φ_ℓ^{CC} **Idea 1**		
Clique linking	$O(n)$	$O(\ell n)$
Exact-k (m vars)	$O(mk)$	$O(\ell mk)$
Total		$O\big(\ell(\|E\| + n + mk)\big)$

replace n by m

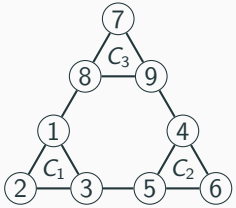
The Exact-k Constraint Dominates Clause Growth

Let n be the number of vertices $|V|$, and m be the number of elements of clique partition of G.

Component	Per step	Total
Basic formula Φ_ℓ		
Init + *Goal*	—	$O(n)$
Independence	$O(\|E\|)$	$O(\ell\|E\|)$
Exact-k (n vars)	$O(nk)$	$O(\ell nk)$
Jump (link + exact-1)	$O(n)$	$O(\ell n)$
Total		$O\big(\ell(\|E\| + nk)\big)$
Clique-based formula Φ_ℓ^{CC} **Idea 1**		
Clique linking	$O(n)$	$O(\ell n)$
Exact-k (m vars)	$O(mk)$	$O(\ell mk)$
Total		$O\big(\ell(\|E\| + n + mk)\big)$

replace n by m

Idea 1: Count Occupied Cliques Instead of Vertices



- $G = (V, E)$ is given, where $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Suppose that a clique partition $\mathcal{P} = \{C_1, C_2, C_3\}$ is obtained, where $C_1 = \{1, 2, 3\}$, $C_2 = \{4, 5, 6\}$, and $C_3 = \{7, 8, 9\}$.

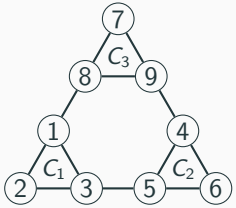
Basic formula Φ_ℓ

$$x_1^t + x_2^t + x_3^t + \cdots + x_7^t + x_8^t + x_9^t = k.$$

Clique-based formula Φ_ℓ^{CC}

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

Idea 1: Count Occupied Cliques Instead of Vertices



- $G = (V, E)$ is given, where $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
- Suppose that a clique partition $\mathcal{P} = \{C_1, C_2, C_3\}$ is obtained, where $C_1 = \{1, 2, 3\}$, $C_2 = \{4, 5, 6\}$, and $C_3 = \{7, 8, 9\}$.

Basic formula Φ_ℓ

$$x_1^t + x_2^t + x_3^t + \cdots + x_7^t + x_8^t + x_9^t = k.$$

We do not need to count tokens over all vertices; count occupied cliques instead!

Clique-based formula Φ_ℓ^{CC}

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

A Fast Heuristic Builds Small, Balanced Clique Partitions

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**
    $C_{\text{best\_index}} \leftarrow C_{\text{best\_index}} \cup \{v\}$;
  else
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\{v\}\}$;
return $\mathcal{P}$;

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 $\mathcal{P}$;

Together, these algorithms produce clique partitions with relatively few cliques and well-balanced clique sizes.

A Fast Heuristic Builds Small, Balanced Clique Partitions

Algorithm 3: `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 be… -fit clique for v
  foreach C_i … $\{C \in \mathcal{P} \mid \forall u \in C_i : \{u, v\} \in$ …
    score … $\tau - |C_i|$;
    if *sco…* > *best_score* **then**
      best_index $\leftarrow i$;
      best_score $\leftarrow$ *score*;
  if *best_index* $\neq \perp$ **then**
    $C_{best\_index} \leftarrow C_{best\_index} \cup \{v\}$;
  else
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\{v\}\}$;
return $\mathcal{P}$;

Algorithm 4: `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**
      $\mathcal{P} \leftarrow \{C \in \mathcal{P} \mid \forall u \in C : \{u, v\} \in$ …
      $|C| \geq |C_s|\}$;
      if … **then**
        $C_s \leftarrow C_s \setminus \{v\}$;
        $C_t \leftarrow \arg\min_{C \in \mathcal{P}} |C|$;
        $C_t \leftarrow C_t \cup \{v\}$;
        changed $\leftarrow$ **true**;
until *changed* $=$ **false**;
$\mathcal{P} \leftarrow \{C \in \mathcal{P} \mid C \neq \emptyset\}$;
return $\mathcal{P}$;

If you are interested, please see the paper for details.

Together, these algorithms produce clique partitions with relatively few cliques and well-balanced clique sizes.

Idea 2: Prune Redundant Moves at the Lower Bound

- Let $d = |I_s \setminus I_g|$. At BMC step $\ell = d$, no redundant move is possible.
- That is, tokens are just moved from $I_s \setminus I_g$ to $I_g \setminus I_s$ with an appropriate order.

Token fixation

Tokens on $I_s \cap I_g$ remain fixed at every step. No vertex outside $I_s \cup I_g$ can hold a token.

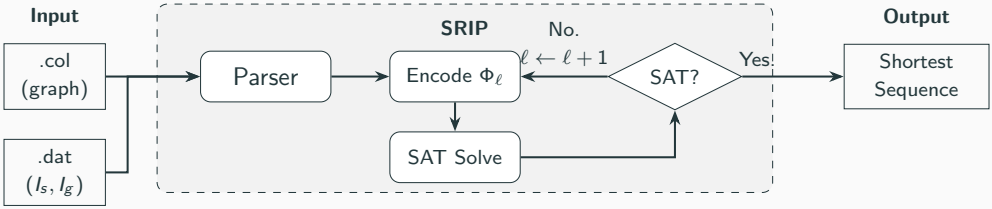
$$\bigwedge_{v \in I_s \cap I_g} \bigwedge_{t=0}^{d} x_v^t \wedge \bigwedge_{v \notin I_s \cup I_g} \bigwedge_{t=0}^{d} \neg x_v^t.$$

Monotonicity

A token that arrives at a goal-only vertex cannot leave; a token leaving a start-only vertex cannot return.

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

SRIP Implementation: Incremental SAT with CaDiCaL



- Rust implementation with CaDiCaL 3.0 as the SAT backend
- We use sequential counters [Sinz, 2005] to encode exact-1 constraints.
- We use totalizers [Bailleux+, 2003] to encode clique-level exact-k.
- Extract and validate a reconfiguration sequence from the SAT model

Experiments and Setup

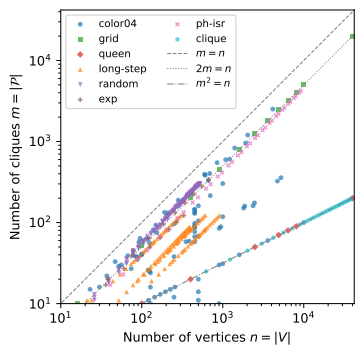
Experiments

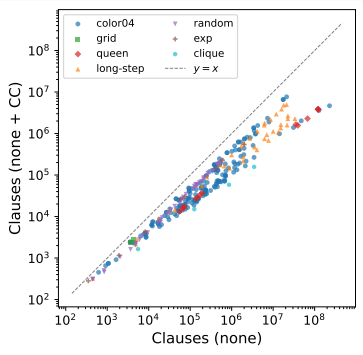
- Exp1. Evaluation of the reduction of the clique-based formula
- Exp3. Comparison with SOTA on CoRe Challenge 693 instances.
  - BMC solvers: recongo [Yamada+, 2024], ReconfAIGERation [Froleyks+, 2022]
  - AI planner: PARIS [Christen+, 2023]
- *Supplemental: Exp2. Ablation on 40 handcrafted clique-rich instances*

Set up

- Environment: Intel Xeon Platinum 8480+ (2.00 GHz), 128 GB RAM, Ubuntu 20.04; one core per instance
- 1800-second timeout
- #best: number of instances where a solver finds the minimum returned length
  - SRIP guarantees optimality; some SOTA solvers return best-so-far solutions.

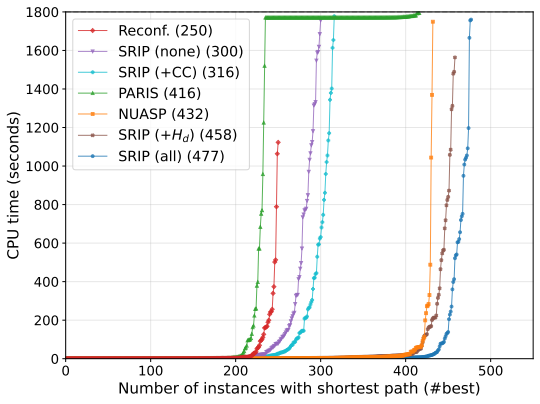
Exp1. Clique partitioning consistently reduces encoding size





- Median compression: $m/n \approx 0.45$; on the largest instances, $m/n = 0.005$ (a 200× reduction).
- At each shortest solved bound, clauses decrease on all 296 comparable instances by about 3–10× at the median.

Exp3. SRIP Finds Shortest Sequences on 477/693 Instances



Result

SRIP with $+H_d + CC$ finds shortest reconfiguration sequences for 477 of 693 instances, the highest count among the compared solvers.

Conclusion and future work

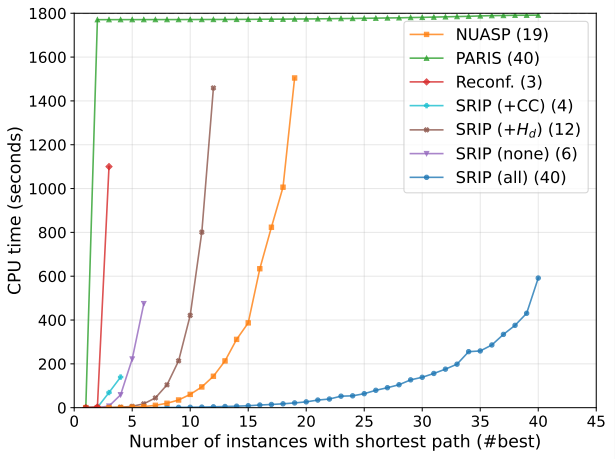
1. SRIP solves ISRP under Token Jumping with **SAT-based BMC** and returns shortest reconfiguration sequences.
2. **Clique-partition-based cardinality encoding** replaces vertex-level exact-k constraints with clique-level constraints, substantially reducing clauses.
3. **Token fixation and monotonicity** constraints strongly prune the search at the lower bound.
4. SRIP finds shortest sequences for **477 of 693** CoRe Challenge instances, the highest number in our comparison.

Future work

Better clique partitions, sound pruning beyond $\ell = d$, and applications to other reconfiguration problems.

Supplemental Slides

Exp2. Ablation: the two techniques are complementary



#best on 40 `clique` instances

Configuration	#best
$+H_d + CC$	40
$+H_d$	12
$+CC$	4
baseline	6

With both clique-based encoding (CC) and pruning constraints (H_d), SRIP obtains shortest sequences on every instance.

Backup: How much does partition quality matter?

BestFit + Rebalancing vs. naive greedy

	Proposed	Naive
Median number of cliques	82	95
Mean size std. deviation	1.53	2.78
Instances solved (of 693)	477	465

Cost

Preprocessing was negligible for both methods: about 0.002 seconds. The performance difference therefore appears in SAT encoding and solving.

Why not minimize the clique count exactly?

On 101 `color04` graphs, OR-Tools CP-SAT:

- proved optimality on only 21,
- returned unverified partitions on 68,
- failed within the limits on 12.

Answer

Partition quality matters, including balance beyond clique count, but exact minimization is too expensive as preprocessing.

Backup: What happens on unreachable instances?

Current behavior

An UNSAT result at bound ℓ rules out a sequence up to the bounds tested. SRIP then increases ℓ; without a practical completeness bound, an unreachable instance normally reaches the global timeout.

Role of H_d

Token fixation and monotonicity are active only at $\ell = d$. They strengthen the first UNSAT test, but they do not constitute a general proof of unreachability.

Future direction

Combine BMC with diameter or invariant bounds, IC3/PDR, or symbolic reachability. ReconfAIGERation and ZDD-based solvers already provide useful reference approaches for proving unsolvability.

Backup: Can the ideas generalize?

- Incremental BMC with block literals is problem-independent once states, goals, and transitions have SAT encodings.
- Clique-based cardinality applies when a state is a fixed-size subset with pairwise conflicts; examples include set packing and matching via a line graph.
- For Token Sliding, the state and clique encodings remain unchanged; the transition additionally requires the source and destination to be adjacent.
- Lower-bound pruning can transfer when each move makes at most one unit of progress toward the goal, but its constraints must be derived for each reconfiguration rule.

Limitation

Problems such as coloring have different state structure and require a different compression argument; the current clique encoding is not plug-and-play.

References

References i

[Bailleux+, 2003] Bailleux, Olivier and Boufkhad, Yacine (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*, pages 108–122. Springer.

[Bonsma+, 2009] Bonsma, Paul S. and Cereceda, Luis (2009).
Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances.
Theoretical Computer Science, 410(50):5215–5226.

References ii

[Brewster+, 2016] Brewster, Richard C., McGuinness, Sean, Moore, Benjamin R. , and Noel, Jonathan A. (2016).
A dichotomy theorem for circular colouring reconfiguration.
Theoretical Computer Science, 639:1–13.

[Cereceda+, 2011] Cereceda, Luis, van den Heuvel, Jan , and Johnson, Matthew (2011).
Finding paths between 3-colorings.
Journal of Graph Theory, 67(1):69–82.

[Christen+, 2023] Christen, Remo, Eriksson, Salomé, Katz, Michael, Muise, Christian, Petrov, Alice, Pommerening, Florian, Seipp, Jendrik, Sievers, Silvan , and Speck, David (2023).

PARIS: planning algorithms for reconfiguring independent sets.

In Gal, Kobi, Nowé, Ann, Nalepa, Grzegorz J., Fairstein, Roy , and Radulescu, Roxana, editors, *Proceedings of the 26th European Conference on Artificial Intelligence (ECAI 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, pages 453–460. IOS Press.

[Froleyks+, 2022] Froleyks, Nils, Yu, Emily , and Biere, Armin (2022).

ReconfAIGERation entering core challenge 2022.

In Soh, Takehide, Okamoto, Yoshio , and Ito, Takehiro, editors, *Core Challenge 2022: Solver and Graph Descriptions*, pages 29–31. arXiv.

Available at arXiv:2208.02495.

[Gopalan+, 2009] Gopalan, Parikshit, Kolaitis, Phokion G., Maneva, Elitza N. , and Papadimitriou, Christos H. (2009).

The connectivity of boolean satisfiability: Computational and structural dichotomies.

SIAM Journal on Computing, 38(6):2330–2355.

References v

[Haddadan+, 2016] Haddadan, Arash, Ito, Takehiro, Mouawad, Amer E., Nishimura, Naomi, Ono, Hirotaka, Suzuki, Akira , and Tebbal, Youcef (2016).
The complexity of dominating set reconfiguration.
Theoretical Computer Science, 651:37–49.

[Ito+, 2011] Ito, Takehiro, Demaine, Erik D., Harvey, Nicholas J. A., Papadimitriou, Christos H., Sideri, Martha, Uehara, Ryuhei , and Uno, Yushi (2011).
On the complexity of reconfiguration problems.
Theoretical Computer Science, 412(12-14):1054–1065.

[Ito+, 2023] Ito, Takehiro, Kawahara, Jun, Nakahata, Yu, Soh, Takehide, Suzuki, Akira, Teruyama, Junichi , and Toda, Takahisa (2023).
ZDD-based algorithmic framework for solving shortest reconfiguration problems.
In Ciré, André A., editor, *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*, pages 167–183. Springer.

[Ito+, 2015] Ito, Takehiro, Ono, Hirotaka , and Otachi, Yota (2015).
Reconfiguration of cliques in a graph.
In Jain, Rahul, Jain, Sanjay , and Stephan, Frank, editors, *Proceedings of the 12th Annual Conference on Theory and Applications of Models of Computation (TAMC 2015)*, volume 9076 of *Lecture Notes in Computer Science*, pages 212–223. Springer.

[Kaminski+, 2012] Kaminski, Marcin, Medvedev, Paul , and Milanic, Martin (2012).
Complexity of independent set reconfigurability problems.
Theoretical Computer Science, 439:9–15.

[Mouawad+, 2017] Mouawad, Amer E., Nishimura, Naomi, Pathak, Vinayak , and Raman, Venkatesh (2017).

Shortest reconfiguration paths in the solution space of boolean formulas.

SIAM Journal on Discrete Mathematics, 31(3):2185–2200.

[Sinz, 2005] Sinz, Carsten (2005).

Towards an optimal CNF encoding of boolean cardinality constraints.

In van Beek, Peter, editor, *Proc. of the 11th International Conference on Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *LNCS*, pages 827–831. Springer.

[Suzuki+, 2016] Suzuki, Akira, Mouawad, Amer E. , and Nishimura, Naomi (2016).
Reconfiguration of dominating sets.
Journal of Combinatorial Optimization, 32(4):1182–1195.

[Takaoka, 2018] Takaoka, Asahi (2018).
Complexity of Hamiltonian cycle reconfiguration.
Algorithms, 11(9):140.

[Yamada+, 2024] Yamada, Yuya, Banbara, Mutsunori, Inoue, Katsumi, Schaub, Torsten , and Uehara, Ryuhei (2024).
Combinatorial reconfiguration with answer set programming: Algorithms, encodings, and empirical analysis.

In Uehara, Ryuhei, Yamanaka, Katsuhisa , and Yen, Hsu-Chun, editors, *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM 2024)*, volume 14549 of *Lecture Notes in Computer Science*, pages 242–256. Springer.