Combinatorial Reconfiguration with Answer Set Programming: Algorithms, Encodings, and Empirical Analysis

Yuya Yamada[1] Mutsunori Banbara[1] Katsumi Inoue[2]
Torsten Schaub[3] Ryuhei Uehara[4]

[1]Nagoya University

[2]National Institute of Informatics

[3]Universität Potsdam

[4]Japan Advanced Institute of Science and Technology

WALCOM2024@Kanazawa
March 19th, 2023

Outline

- **Combinatorial Reconfiguration Problems** (CRPs) are defined as the task of deciding, for a given combinatorial problem and two among its feasible solutions, whether or not one is reachable from another via a sequence of adjacent feasible solutions.
- **Answer Set Programming** (ASP) is a declarative programming paradigm for knowledge representation and reasoning.

Main contributions

1. We develop the **bounded combinatorial reconfiguration** for solving combinatorial reconfiguration problems.
2. The resulting *recongo* system is an ASP-based CRP solver.
3. We present a collection of ASP encodings for solving **independent set reconfiguration problems**.

Combinatorial reconfiguration

Theoretical aspects

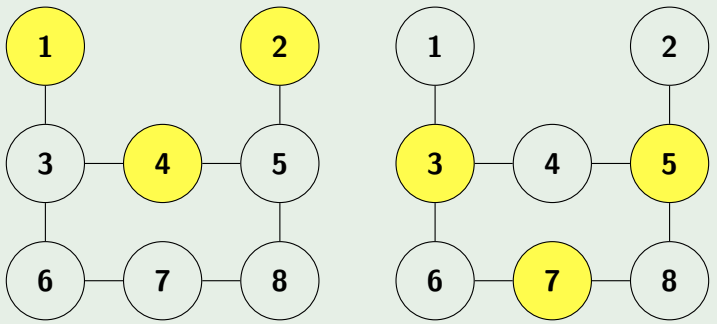
- A great effort has been made in the field of theoretical computer science over the last decade.
- For many NP-complete problems, their reconfigurations have been shown to be **PSPACE-complete**:
  - SAT reconfiguration [Gopalan+,'09]
  - **Independent set reconfiguration** [Ito+,'11], and many others.

Practical aspects

- A great attention has been paid since 2020.
- International competitions, **CoRe Challenge**, were held to stimulate research and development on practical CRP solving in 2022 and 2023.
- Some CRP solvers have been proposed: *PARIS* (planning) and *ddreconf* (ZDD) as well as *recongo* (ASP).

Independent Set Reconfiguration Problem (ISRP)

An input example

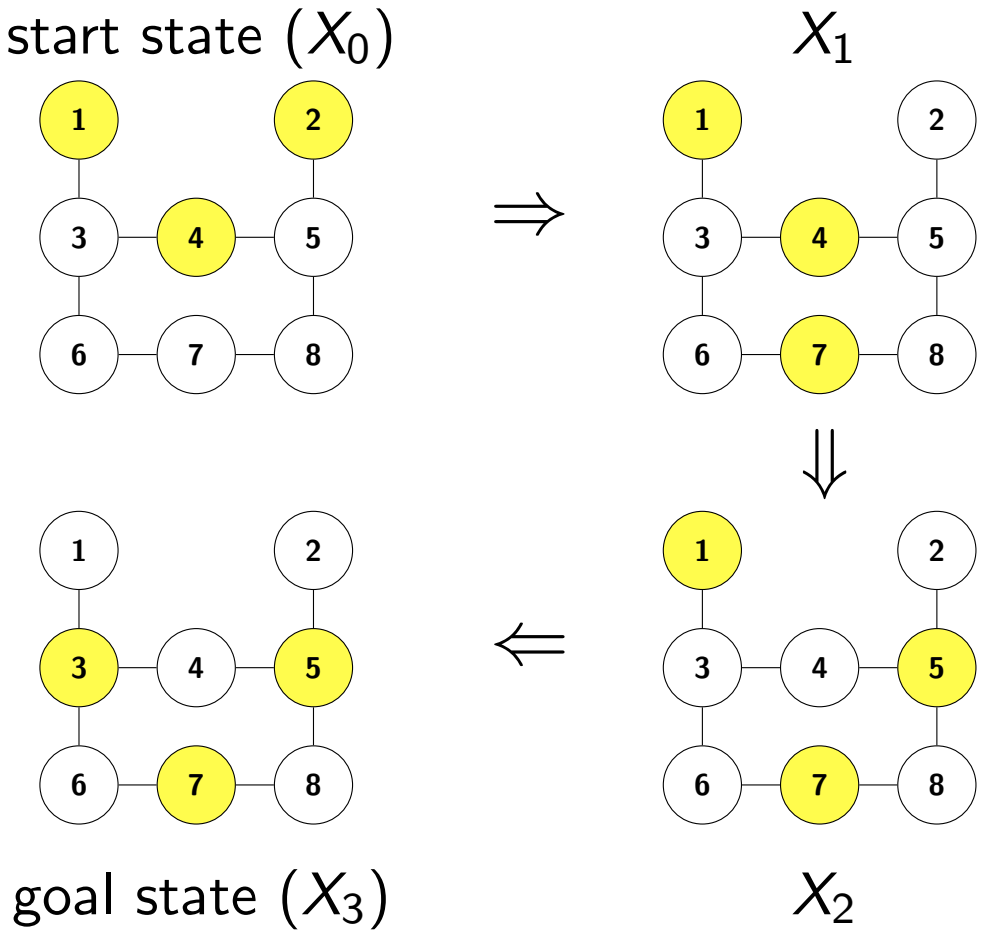


start state goal state

- The example consists of a graph having 8 nodes and 8 edges, a size of independent sets where $k = 3$, and 2 feasible solutions.
- The independent sets are highlighted in yellow.

- We consider an **adjacency relation**: **Token Jumping**.
  - A single node included in an independent set in state X at step t moves to any other node in state X' at step $t + 1$.

Example of the ISRP



- The start state can reach the goal state with length $\ell = 3$.
- Each state corresponds an independent set.
- Each transition satisfies the adjacency relation.

Answer Set Programming (ASP)

- ASP is a declarative programming paradigm with first-order logic, widely used in artificial intelligence.
- ASP system computes answer sets based on a stable model semantics [Gelfond and Lifschitz,'88] from logic programs.
- Logic programs are finite sets of rules.

Major advantages of using ASP for the CRP

- ASP can concisely represent many combinatorial problems by expressive language, and can easily extend them to its reconfiguration.
- Recent advances in multi-shot ASP solving allow for efficient reachability checking of combinatorial reconfiguration problems.

Overview

We present an approach for solving CRPs based on ASP.

1. **Bounded Combinatorial Reconfiguration (BCR)**
   - The BCR is inspired by bounded model checking [Biere,'09].
2. **CRP solver recongo**
   - The system uses multi-shot ASP solving.
3. **Collection of ASP encodings for solving ISRPs**
   - It includes one heuristics and four hint constraints.
4. **Experiments on a benchmark set of CoRe Challenge 2022**
   - Its results show that our approach can be highly effective for CRP solving.

Awards

recongo **ranked first** at a metric of the CoRe Challenge 2022, and at 5 metrics of the CoRe Challenge 2023.

Bounded Combinatorial Reconfiguration

The BCR is an approach to solve CRPs by deciding whether or not there are reconfiguration sequences of **bounded length** ℓ.

- A set of variables $\boldsymbol{x}^t$ represents each state X at step t.
- To check the existence of a reconfiguration sequence of bounded length ℓ, we can use

$$\varphi_\ell = S(\boldsymbol{x}^0) \wedge \bigwedge_{t=0}^{\ell} C(\boldsymbol{x}^t) \wedge \bigwedge_{t=1}^{\ell} T(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t) \wedge G(\boldsymbol{x}^\ell).$$

  - $S(\boldsymbol{x}^0)$ specifies conditions on the start state.
  - $C(\boldsymbol{x}^t)$ represents the constraints of combinatorial problem.
  - $T(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t)$ represents each adjacency relation.
  - $G(\boldsymbol{x}^\ell)$ specifies conditions on the goal state.
- If φ_ℓ is satisfiable, there is a reconfiguration sequence of length ℓ.
- Otherwise, we keep on reconstructing a successor, for instance $\varphi_{\ell+1}$, and checking its satisfiability until a reconfiguration sequence is found.

BCR Algorithm powered by multi-shot ASP solving

$$\varphi_\ell = S(\boldsymbol{x}^0) \wedge \bigwedge_{t=0}^{\ell} C(\boldsymbol{x}^t) \wedge \bigwedge_{t=1}^{\ell} T(\boldsymbol{x}^{t-1}, \boldsymbol{x}^t) \wedge G(\boldsymbol{x}^\ell)$$

- Obviously it is inefficient to fully reconstruct φ_ℓ in each transition because of **the expensive grounding**.

Key idea

We can **incrementally construct $\varphi_{\ell+1}$ from its predecessor φ_ℓ** by

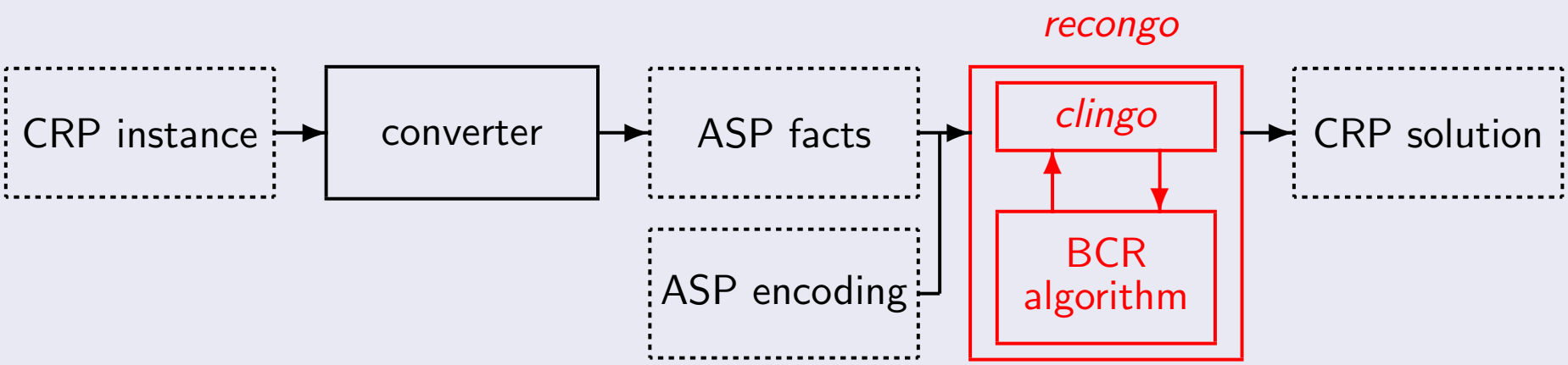
- **adding** $C(\boldsymbol{x}^{\ell+1})$, $T(\boldsymbol{x}^\ell, \boldsymbol{x}^{\ell+1})$, and $G(\boldsymbol{x}^{\ell+1})$,
- **deactivating** $G(\boldsymbol{x}^\ell)$

with a multi-shot ASP solving.

Main advantages

- Reducing the cost of grounding
- Reusing learnt clauses gained in solving predecessors

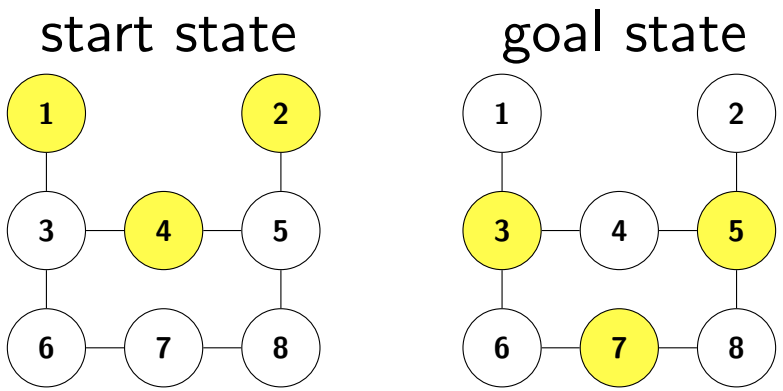
The architecture of recongo



1. *recongo* reads a CRP instance converted into ASP facts.
2. These facts are combined with an ASP encoding for CRP solving, which are afterward solved by the BCR algorithm powered by an efficient ASP solver, in our case *clingo*.

- The BCR algorithm can be implemented by using *clingo*'s multi-shot ASP solving.
- The current implementation of *recongo* **covers all metrics of CoRe Challenge** : existent, shortest, and longest.

ASP fact format of ISRP




```
k(3).
node(1).   node(2).   node(3).   node(4).
node(5).   node(6).   node(7).   node(8).
edge(1,3). edge(2,5). edge(3,4). edge(3,6).
edge(4,5). edge(5,8). edge(6,7). edge(7,8).
start(1).  start(2).  start(4).
goal(3).   goal(5).   goal(7).
```

ASP encoding for ISRP

```
 1  #program base.
 2  :- not in(X,0), start(X).
 3
 4  #program step(t).
 5  K { in(X,t): node(X) } K :- k(K).
 6  :- in(X,t), in(Y,t), edge(X,Y).
 7
 8  moved_from(X,t) :- in(X,t-1), not in(X,t), t > 0.
 9  :- not 1 { moved_from(X,t) } 1, t > 0.
10
11  #program check(t).
12  :- not in(X,t), goal(X), query(t).
```

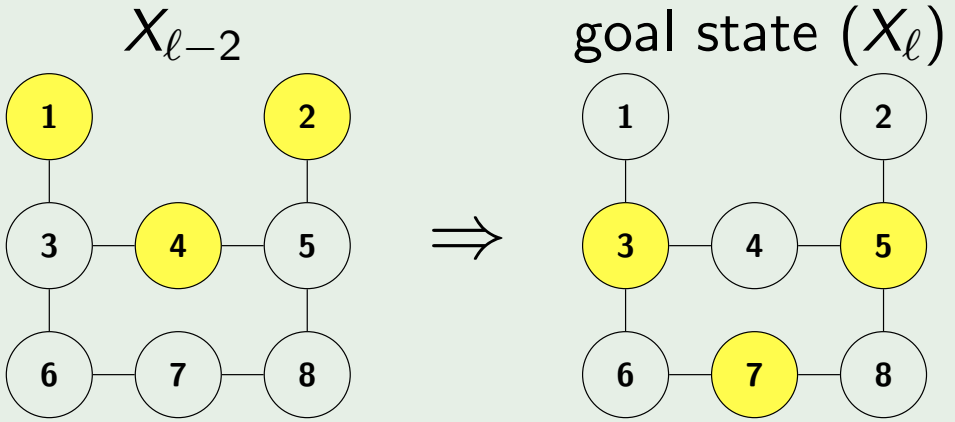
- (2): Conditions on the start state
- (5)–(6): The constraints of the independent set problem
- (8)–(9): The adjacency relation (Token Jumping)
- (12): Conditions on the goal state

Hint constraints for ISRP

We present 5 hint constraints (3 types) to accelerate ISRP solving.

1. **Distance constraint hints (`d1`, `d2`)**
2. **Forbidding redundant moves (`t1`, `t2`)**
3. **Maximal independent set heuristics (`h`)**

An example of invalid reconfiguration sequences forbidden by `d2`

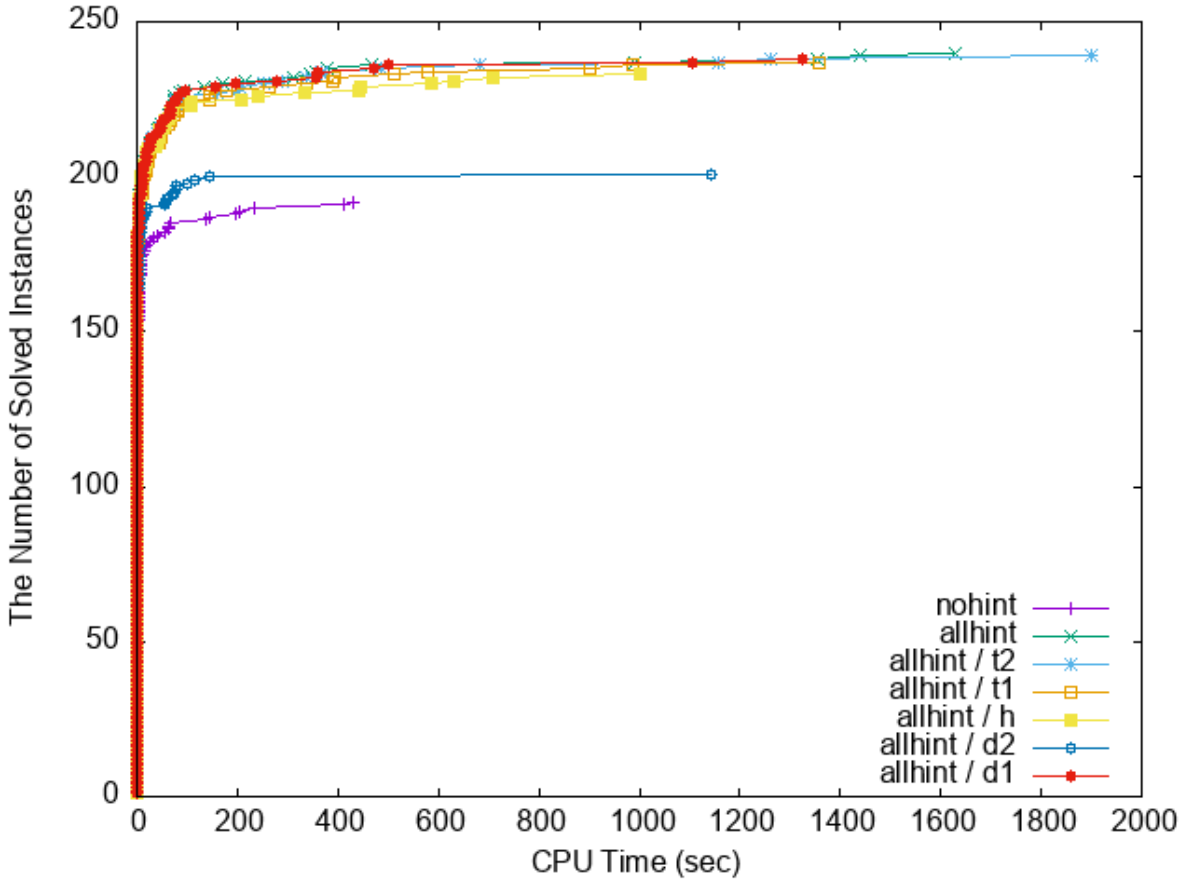


- In the sequence, the lower bound of distance between $X_{\ell-2}$ and X_ℓ is 3, but it is greater than the possible number of transitions, namely 2.

Experiments: existent

- We use a benchmark set of CoRe Challenge 2022.
  - **369 ISRP instances** in a total
- We compare **7 encodings**.
  - `no-hint`: the ISRP encoding without any hints
  - `all-hints`: the ISRP encoding with all 5 hints
  - `all-hints/`X: the `all-hints` except one hint $X \in \{\texttt{d1}, \texttt{d2}, \texttt{t1}, \texttt{t2}, \texttt{h}\}$
- We use *recongo* 0.2 powered by *clingo* 5.
- A time limit is 1 hour for each instance.
- Mac OS machine with Intel Xeon W 12-core 3.3 GHz processors and 96 GB RAM

Cactus plot: existent



- The horizontal axis (x-axis) indicates CPU times in seconds, and the vertical axis (y-axis) indicates #solved instances.
- The distance hint d2 shows the best performance because the difference of `all-hints/d2` from the `all-hints` is largest.

CoRe Challenge 2023

CoRe Challenge 2023 is the second international competition of combinatorial reconfiguration.

- The competition has three metrics:
  - **existent**, **shortest**, and **longest**
- Each metric is evaluated by 4 indices:
  - single-engine solvers or overall solvers
  - on CPU time or wall clock time
- The benchmark set consists of **693 ISRP instances** in a total.
- Submitted solvers are evaluated by the organizers.
- 15 solvers (4 teams) participated:
  - ASP, planning, ZDD, and BFS.

recongo ranked first at five metrics and second at seven metrics out of twelve total.

Single-engine solvers shortest metric (CPU)

benchmark family	#instances	max length	*PARIS*	*recongo*	*sano*	*ddreconf*
color04	202	112	198	**200**	154	76
grid	49	8	**2**	**2**	**2**	**2**
handcraft	6	69	**5**	**5**	**5**	**5**
power	17	442,175	4	0	2	**11**
queen	48	94	**46**	25	9	8
sp	30	360,437	9	2	6	**15**
square	17	1,722	6	0	3	**17**
exp_instance	20	4,239	9	5	7	**17**
power_wide	17	221,003	0	0	0	**10**
power_wider	17	110,423	0	0	0	**9**
square_wide	17	1,722	0	0	0	**17**
square_wider	17	1,722	0	0	0	**17**
ph-isr	36	31	**1**	0	0	**1**
random_instance	200	115	140	**194**	17	36
total	693		420	**433**	205	241

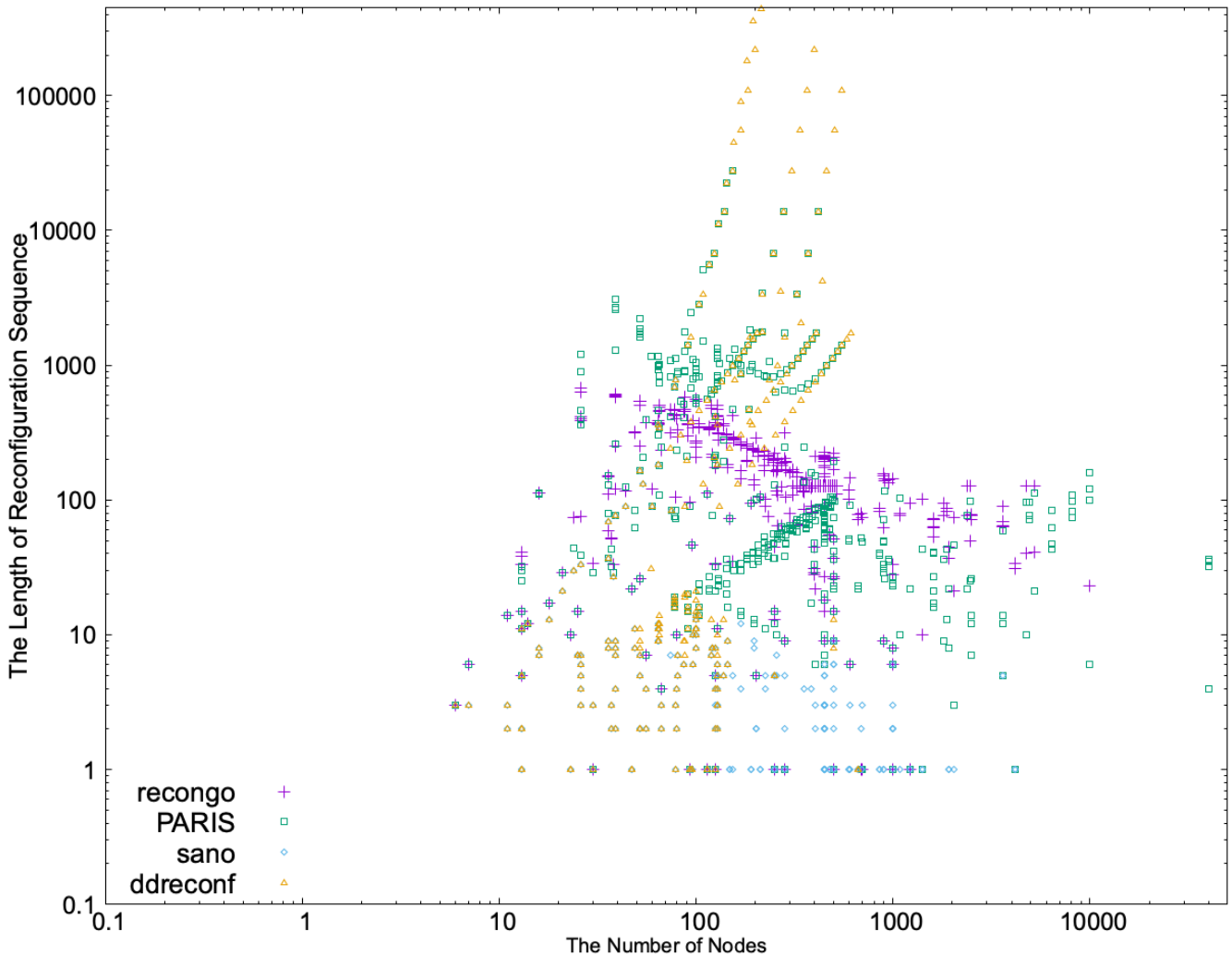
- The score in shortest metric represents the number of instances that each solver can find the shortest sequence among all solvers.

Single-engine solvers shortest metric (CPU)

benchmark family	#instances	max length	*PARIS*	*recongo*	*sano*	*ddreconf*
`color04`	202	112	198	**200**	154	76
`grid`	49	8	**2**	**2**	**2**	**2**
`handcraft`	6	69	**5**	**5**	**5**	**5**
`power`	17	442,175	4	0	2	**11**
`queen`	48	94	**46**	25	9	8
`sp`	30	360,437	9	2	6	**15**
`square`	17	1,722	6	0	3	**17**
`exp_instance`	20	4,239	9	5	7	**17**
`power_wide`	17	221,003	0	0	0	**10**
`power_wider`	17	110,423	0	0	0	**9**
`square_wide`	17	1,722	0	0	0	**17**
`square_wider`	17	1,722	0	0	0	**17**
`ph-isr`	36	31	**1**	0	0	**1**
`random_instance`	200	115	140	**194**	17	36
total	693		420	**433**	205	241

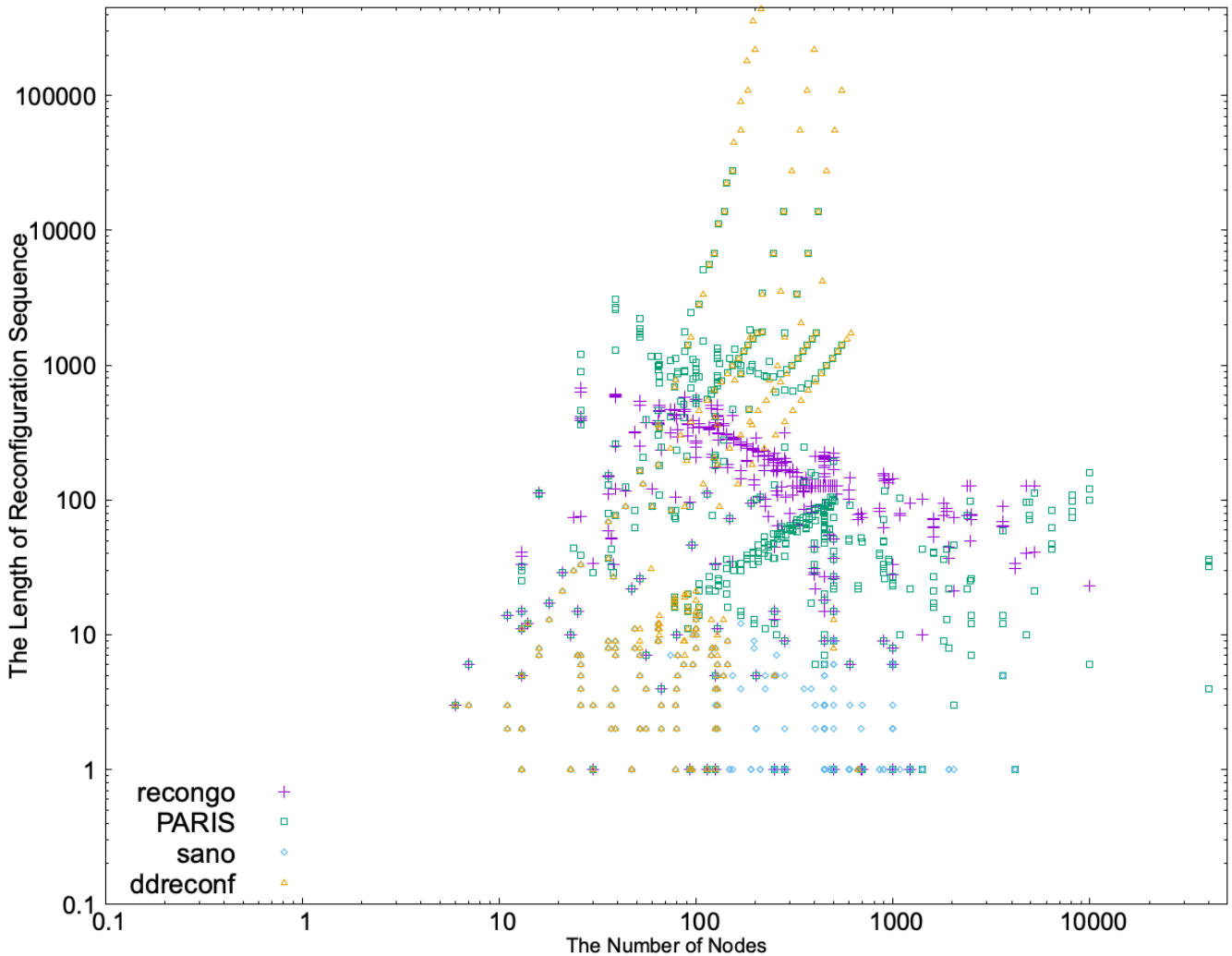
- *recongo* showed good performance for the `color04` and `random_instance` families.

Scatter diagram: longest reconfiguration sequences



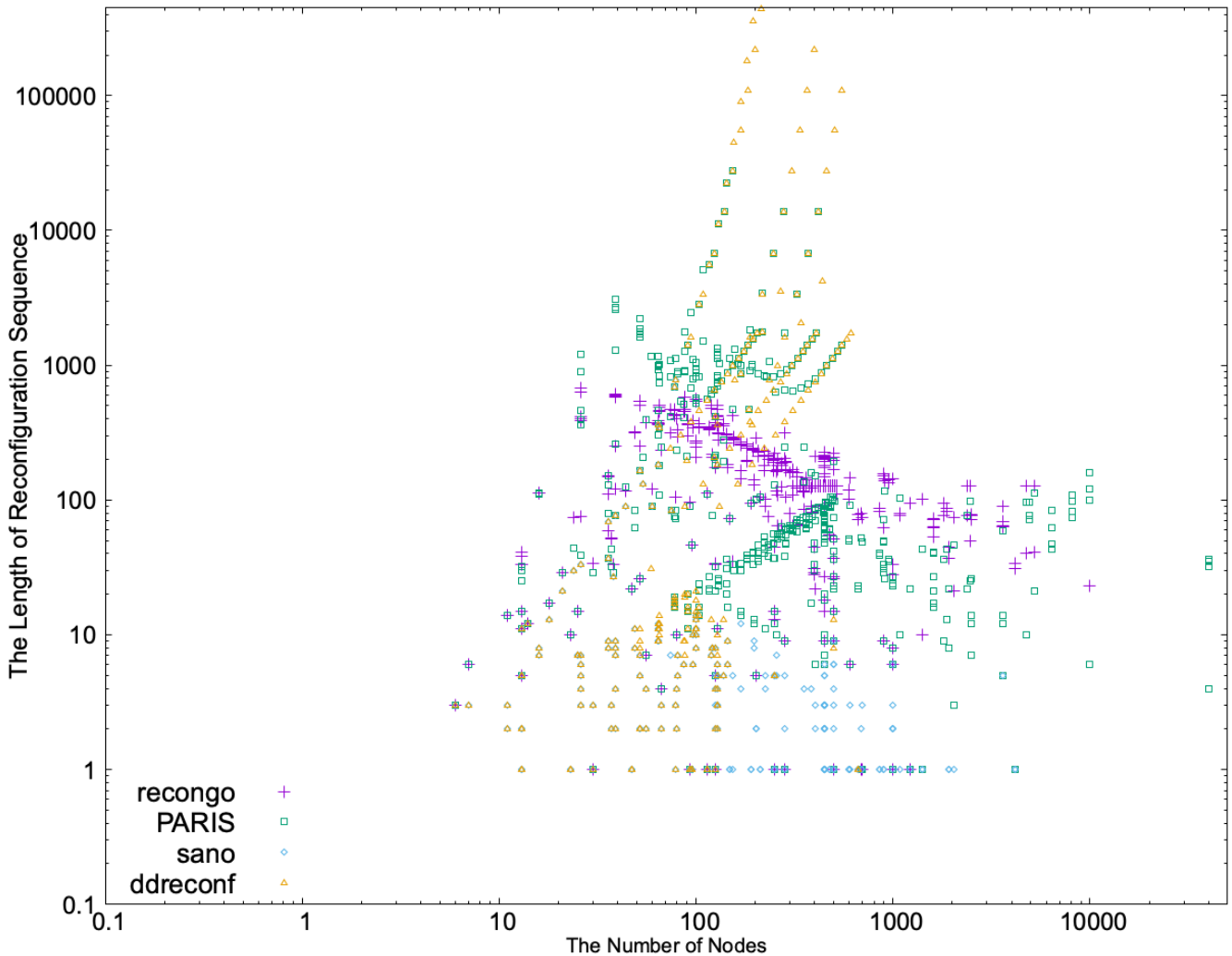
- The horizontal axis indicates the number of nodes, and the vertical axis indicates the length of found reconfiguration sequence.

Scatter diagram: longest reconfiguration sequences



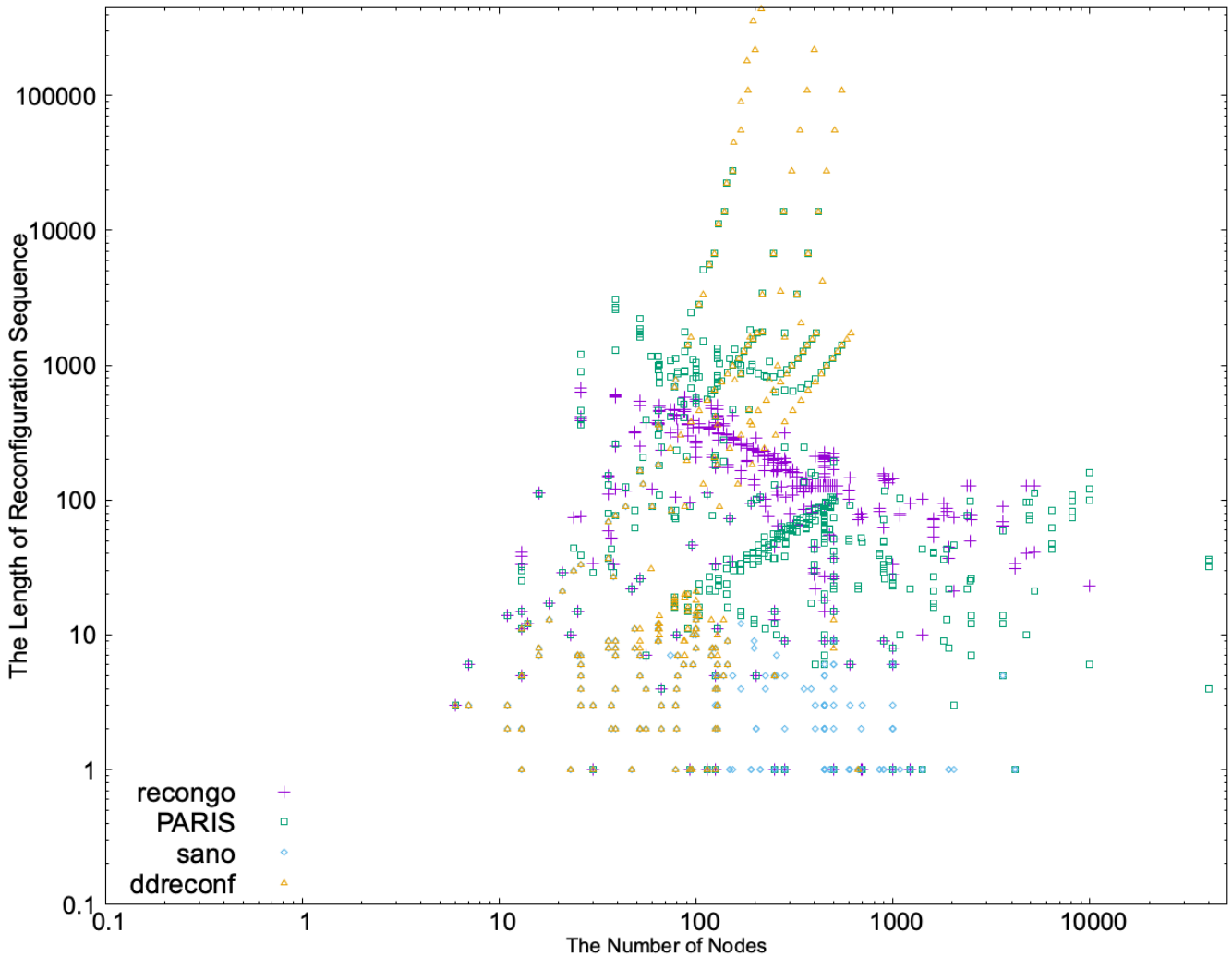
- Each dot represents that a solver found a reconfiguration sequence with length y for an instance in which its number of nodes is x.

Scatter diagram: longest reconfiguration sequences



- Each color corresponds to each solver.
- *recongo* is violet one.

Scatter diagram: longest reconfiguration sequences



- *recongo* was able to find almost reconfiguration sequences $x < 1,000$ and $y < 1,000$.

Conclusion

We presented an approach for solving CRPs based on ASP.

- *recongo* is available from https://github.com/banbaralab/recongo.

Future work

1. **Extending the BCR** to other problems of combinatorial reconfiguration
   - The connectivity problems
   - The diameter problems
2. **Deciding unreachability** without giving the upper bounds of length
   - At first, we have investigated a **counter abstraction** used in the *PARIS* solver [Christen+,'23].
3. **Extending recongo** to a cost-optimal combinatorial reconfiguration