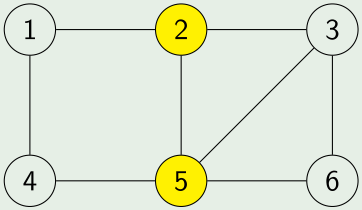
Dominating Set Reconfiguration Using Answer Set Programming

Masato Kato[1], Torsten Schaub[2], Takehide Soh[3], Naoyuki Tamura[3], Mutsunori Banbara[1]

[1] Nagoya University
[2] Universität Potsdam
[3] Kobe University

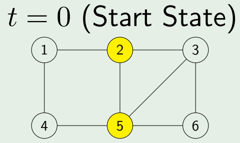
The 2nd NII Collaborative Research Meeting
Nagoya University

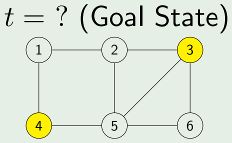
Revisiting the Dominating Set



- For a given graph $G = (V, E)$,
  a subset of nodes $S \subseteq V$ is called a **dominating set** of G
  if every node in V is either in S or adjacent to a node in S.
- The **Dominating Set Problem** (DSP) is the problem of finding a dominating set of a given size k and graph.
- The **Minimum Dominating Set Problem** (MDSP) is the problem of finding the smallest dominating set.

What is Dominating Set Reconfiguration Problem (DSRP)?

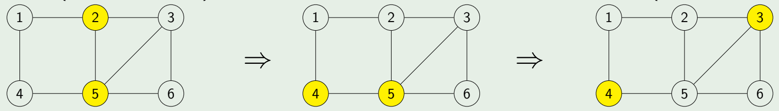




Input: DSP and its two solutions (i.e., dominating sets)
Question: Can we transform the start state into the goal state by passing through only dominating sets, changing only one node at a time?

What is Dominating Set Reconfiguration Problem (DSRP)?

$t = 0$ (Start State) $\Rightarrow$ $t = 1$ $\Rightarrow$ $t = 2$ (Goal State)



Input: DSP and its two solutions (i.e., dominating sets)
Question: Can we transform the start state into the goal state by passing through only dominating sets, changing only one node at a time?
Answer: Yes (or given instance is reachable).

- At each step t, the constraints of the dominating set are satisfied.
- In each transition, exactly one node (token) changes.
  - For example, in the transition from $t = 0$ to $t = 1$, a token moves from node 2 to node 4, and the other tokens remain unchanged.
- It is possible to reach the goal state from the start state in two steps.

Definition and Background of DSRP

DSRP is the problem of determining, given a DSP and two of its feasible solutions, whether there exists a sequence of *adjacent solutions* that transforms one solution into the other.

- *adjacent solutions*: are two feasible solutions that differ by only one node. (a.k.a. Token Jumping)

Definition and Background of DSRP

DSRP is the problem of determining, given a DSP and two of its feasible solutions, whether there exists a sequence of *adjacent solutions* that transforms one solution into the other.

- *adjacent solutions*: are two feasible solutions that differ by only one node. (a.k.a. Token Jumping)

Background

- DSRP is one of the representative **combinatorial reconfiguration problems**, alongside the Independent Set Reconfiguration [Ito+,'11].
- It is generally known to be **PSPACE-complete** [Haddadan+,'16].
- Well studied in theory [Haas+ '14, Bonamy+ '21, Suzuki+ '16].
- DSRP has applications in sensor networks and social networks.

Definition and Background of DSRP

DSRP is the problem of determining, given a DSP and two of its feasible solutions, whether there exists a sequence of *adjacent solutions* that transforms one solution into the other.

- *adjacent solutions*: are two feasible solutions that differ by only one node. (a.k.a. Token Jumping)

Background

- DSRP is one of the representative **combinatorial reconfiguration problems**, alongside the Independent Set Reconfiguration [Ito+,'11].
- It is generally known to be **PSPACE-complete** [Haddadan+,'16].
- Well studied in theory [Haas+ '14, Bonamy+ '21, Suzuki+ '16].
- DSRP has applications in sensor networks and social networks.

Despite its importance, research on practical aspects is lacking. In particular, there is no established implementation for solving DSRP.

Research Goal

Our goal is to realize a system that efficiently solves DSRP .

Research Goal

Our goal is to realize a system that efficiently solves DSRP using ASP.

Advantages of Using ASP for DSRP

- **The high expressiveness of the ASP language** allows for concise descriptions of the underlying combinatorial problems.
  - Extensions to combinatorial reconfiguration problems are also easy.
- **The combinatorial reconfiguration solver** *recongo* [Yamada+, '23], based on ASP, is available.
  - It won multiple categories in the International Combinatorial Reconfiguration Competition (CoRe Challenge) 2023.

Contribution Summary

1. **Comparison of existing ASP encodings for DSP**
   - Potassco Enc. [ASPCOMP, '09] vs. Huynh Enc. [Huynh, '20]
   - We used 167 graph instances in the MDSP setting.
   - Potassco Enc. found optimal solutions for 124 instances, confirming its superiority.

Contribution Summary

1. **Comparison of existing ASP encodings for DSP**
    - Potassco Enc. [ASPCOMP, '09] vs. Huynh Enc. [Huynh, '20]
    - We used 167 graph instances in the MDSP setting.
    - Potassco Enc. found optimal solutions for 124 instances, confirming its superiority.
2. **Development of ASP encodings/hint constraints for DSRP**

Contribution Summary

1. **Comparison of existing ASP encodings for DSP**
   - Potassco Enc. [ASPCOMP, '09] vs. Huynh Enc. [Huynh, '20]
   - We used 167 graph instances in the MDSP setting.
   - Potassco Enc. found optimal solutions for 124 instances, confirming its superiority.
2. **Development of ASP encodings/hint constraints for DSRP**
3. **Development of a benchmark set (442 in total) for DSRP**
   - This is the first public benchmark set for DSRP including 310 reachable and 132 unreachable instances.
   - It consists of graphs with 11–1000 nodes and 200k–450k edges, with reconfiguration lengths ranging from 1–23.

Contribution Summary

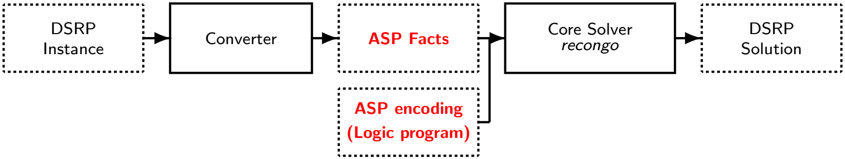
1. **Comparison of existing ASP encodings for DSP**
   - Potassco Enc. [ASPCOMP, '09] vs. Huynh Enc. [Huynh, '20]
   - We used 167 graph instances in the MDSP setting.
   - Potassco Enc. found optimal solutions for 124 instances, confirming its superiority.
2. **Development of ASP encodings/hint constraints for DSRP**
3. **Development of a benchmark set (442 in total) for DSRP**
   - This is the first public benchmark set for DSRP including 310 reachable and 132 unreachable instances.
   - It consists of graphs with 11–1000 nodes and 200k–450k edges, with reconfiguration lengths ranging from 1–23.
4. **Evaluation of the developed ASP encoding for DSRP**
   - effectiveness of the proposed hint constraints
   - comparison with a ZDD-based DSRP solver

Contribution Summary

1. **Comparison of existing ASP encodings for DSP**
   - Potassco Enc. [ASPCOMP, '09] vs. Huynh Enc. [Huynh, '20]
   - We used 167 graph instances in the MDSP setting.
   - Potassco Enc. found optimal solutions for 124 instances, confirming its superiority.
2. **Development of ASP encodings/hint constraints for DSRP**
3. **Development of a benchmark set (442 in total) for DSRP**
   - This is the first public benchmark set for DSRP including 310 reachable and 132 unreachable instances.
   - It consists of graphs with 11–1000 nodes and 200k–450k edges, with reconfiguration lengths ranging from 1–23.
4. **Evaluation of the developed ASP encoding for DSRP**
   - effectiveness of the proposed hint constraints
   - comparison with a ZDD-based DSRP solver

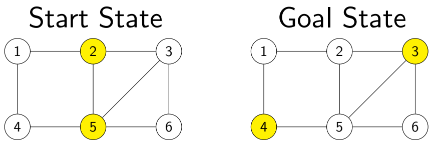
In the following, we explore the contributions 2 and 4.

ASP-based Framework for Solving DSRP



1. Convert DSRP instance into ASP facts.
2. Combine the obtained ASP facts with the logic program (ASP encoding) to represent DSRP.
3. Use the ASP-based combinatorial reconfiguration solver *recongo* to solve DSRP.

ASP Facts for DSRP



```
node(1). node(2). node(3). node(4). node(5). node(6).

edge(1,2). edge(1,4). edge(2,3). edge(2,5).
edge(3,5). edge(3,6). edge(4,5). edge(5,6).


k(2).
start(2). start(5).
goal(3).  goal(4).
```

- node(X) represents a node X, and edge(X,Y) represents an edge X−Y.
- k(2) indicates that the size of the dominating set is exactly 2.
- start(X) represents the start state, and goal(X) represents the goal state.

ASP Encoding for DSRP

```
#program base.
% Constraints of the start state
:- not in(X,0), start(X).

#program step(t).
% Constraints of the dominating set problem
K { in(X,t) : node(X) } K :- k(K).
dominated(X,t) :- in(X,t).
dominated(Y,t) :- in(X,t), edge(X,Y).
dominated(Y,t) :- in(X,t), edge(Y,X).
:- not dominated(X,t), node(X).

% Constraints of the token jumping
token_removed(X,t) :- in(X,t-1), not in(X,t), t > 0.
:- not 1 { token_removed(X,t) } 1, t > 0.

#program check(t).
% Constraints of the goal state
:- not in(X,t), goal(X), query(t).
```

base

step(t)

check(t)

- Consists of three subprograms: `base`, `step(t)`, and `check(t)`.
- The argument `t` in the program represents the step number.
- `in(X,t)` indicates that node `X` is included in the dominating set at step t.

ASP Encoding for DSRP

```
 1  #program base.
 2  % Constraints of the start state
 3  :- not in(X,0), start(X).
 4
 5  #program step(t).
 6  % Constraints of the dominating set problem
 7  K { in(X,t) : node(X) } K :- k(K).
 8  dominated(X,t) :- in(X,t).
 9  dominated(Y,t) :- in(X,t), edge(X,Y).
10  dominated(Y,t) :- in(X,t), edge(Y,X).
11  :- not dominated(X,t), node(X).
12
13  % Constraints of the token jumping
14  token_removed(X,t) :- in(X,t-1), not in(X,t), t > 0.
15  :- not 1 { token_removed(X,t) } 1, t > 0.
16
17  #program check(t).
18  % Constraints of the goal state
19  :- not in(X,t), goal(X), query(t).
```

base

- The base part describes the constraints for the start state.
- Line 3 enforces that the state at step 0 matches the start state.

ASP Encoding for DSRP

```
1  #program base.
2  % Constraints of the start state
3  :- not in(X,0), start(X).
4
5  #program step(t).
6  % Constraints of the dominating set problem
7  K { in(X,t) : node(X) } K :- k(K).
8  dominated(X,t) :- in(X,t).
9  dominated(Y,t) :- in(X,t), edge(X,Y).
10 dominated(Y,t) :- in(X,t), edge(Y,X).
11 :- not dominated(X,t), node(X).
12
13 % Constraints of the token jumping
14 token_removed(X,t) :- in(X,t-1), not in(X,t), t > 0.
15 :- not 1 { token_removed(X,t) } 1, t > 0.
16
17 #program check(t).
18 % Constraints of the goal state
19 :- not in(X,t), goal(X), query(t).
```

step(t)

- The `step(t)` part describes the constraints of the DSP and adjacency relation.
- Line 7 specifies that the number of nodes `X` for which `in(X,t)` is true must be exactly `K`.

ASP Encoding for DSRP

```
1  #program base.
2  % Constraints of the start state
3  :- not in(X,0), start(X).
4
5  #program step(t).
6  % Constraints of the dominating set problem
7  K { in(X,t) : node(X) } K :- k(K).
8  dominated(X,t) :- in(X,t).
9  dominated(Y,t) :- in(X,t), edge(X,Y).
10 dominated(Y,t) :- in(X,t), edge(Y,X).
11 :- not dominated(X,t), node(X).
12
13 % Constraints of the token jumping
14 token_removed(X,t) :- in(X,t-1), not in(X,t), t > 0.
15 :- not 1 { token_removed(X,t) } 1, t > 0.
16
17 #program check(t).
18 % Constraints of the goal state
19 :- not in(X,t), goal(X), query(t).
```

step(t)

- Lines 8-10 introduce an auxiliary atom `dominated(X,t)` to represent that node `X` is included in the dominating set or in the set of its neighboring nodes.
- Line 11 enforces that all nodes `X` satisfy `dominated(X,t)`.

ASP Encoding for DSRP

```
 1  #program base.
 2  % Constraints of the start state
 3  :- not in(X,0), start(X).
 4
 5  #program step(t).
 6  % Constraints of the dominating set problem
 7  K { in(X,t) : node(X) } K :- k(K).
 8  dominated(X,t) :- in(X,t).
 9  dominated(Y,t) :- in(X,t), edge(X,Y).
10  dominated(Y,t) :- in(X,t), edge(Y,X).
11  :- not dominated(X,t), node(X).
12
13  % Constraints of the token jumping
14  token_removed(X,t) :- in(X,t-1), not in(X,t), t > 0.
15  :- not 1 { token_removed(X,t) } 1, t > 0.
16
17  #program check(t).
18  % Constraints of the goal state
19  :- not in(X,t), goal(X), query(t).
```

step(t)

- Line 14 introduces an auxiliary atom `token_removed(X,t)` to indicate that a token was removed from node `X` at step `t`.
- Line 15 enforces that the number of `token_removed(X,t)` is exactly 1.

ASP Encoding for DSRP

```
1  #program base.
2  % Constraints of the start state
3  :- not in(X,0), start(X).
4
5  #program step(t).
6  % Constraints of the dominating set problem
7  K { in(X,t) : node(X) } K :- k(K).
8  dominated(X,t) :- in(X,t).
9  dominated(Y,t) :- in(X,t), edge(X,Y).
10 dominated(Y,t) :- in(X,t), edge(Y,X).
11 :- not dominated(X,t), node(X).
12
13 % Constraints of the token jumping
14 token_removed(X,t) :- in(X,t-1), not in(X,t), t > 0.
15 :- not 1 { token_removed(X,t) } 1, t > 0.
16
17 #program check(t).
18 % Constraints of the goal state
19 :- not in(X,t), goal(X), query(t).
```

check(t)

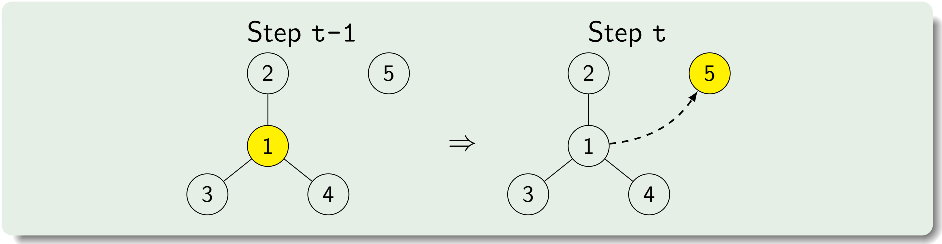
- The `check(t)` part describes the constraints for the goal state.
- Line 19 enforces that `in(X,t)` matches the goal state at the current step `t`.

Proposed Hint Constraints

Two hint constraints were devised to improve solving performance.

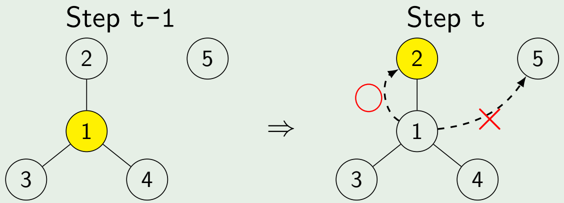
1. [Existing] Constraint on the lower bound of required transition steps (d1 and d2)
   - d1: Enforces that the number of nodes included in the dominating set at the start but not at step t is at most t.
2. [Existing] Constraint prohibiting redundant token moves (t1 and t2)
   - t1: Prohibits a token from being added back at step t to a node from which it was removed at step t−1.
3. **[Proposed]** Constraint on token movement destination (t3)
   - If a token is removed at step t from a node X that has no neighboring tokens at step t−1, the token's destination must be one of X's neighboring nodes.
4. **[Proposed]** Heuristics for minimal dominating sets (heu)
   - At each step, variable selection and value assignment are performed to ensure the dominating set remains as minimal as possible.

Example of Transitions Prohibited by Constraint t3



- If the token on node 1 moves to node 5, which is not an adjacent node, the dominating set constraint will not be satisfied.
  - There will be no tokens on node 1 or its neighboring nodes.

Example of Transitions Prohibited by Constraint `t3`



- If the token on node 1 moves to node 5, which is not an adjacent node, the dominating set constraint will not be satisfied.
    - There will be no tokens on node 1 or its neighboring nodes.

Hint

When a token moves from node 1, which has no neighboring tokens, the destination must be one of node 1's neighboring nodes.

Evaluation Experiment (existent category)

Purpose

Evaluation of the effectiveness of the proposed method.

Experiment 1: hint constraint evaluaion

- Among six hint constraints, only three—`t1`, `t2`, and `t3`—showed good performance in a preliminary experiment. We tested their 2^3 combinations.

Experiment 2: comparison with state-of-the-art

- **ddreconf** is a ZDD-based combinatorial reconfiguration solver awarded in an international competition CoRe Challenge 2023.

The setting is as follows:

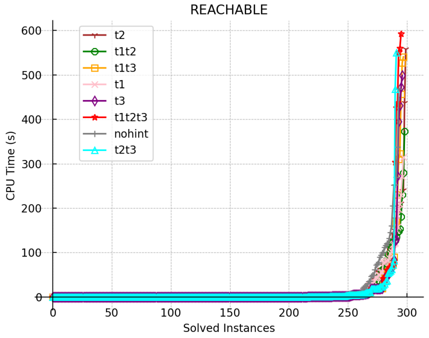
Benchmark created: 442 problems (310 reachable, 132 unreachable)

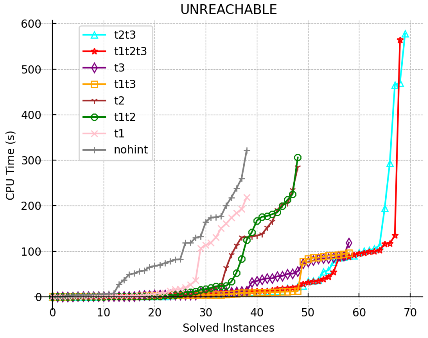
Solver used: *recongo*-0.3.0 + *clingo*-5.6.2 (*trendy*)

Environment: Mac OS, Intel Core i7 3.2GHz, 64GB RAM

Time limit: 10 minutes per problem

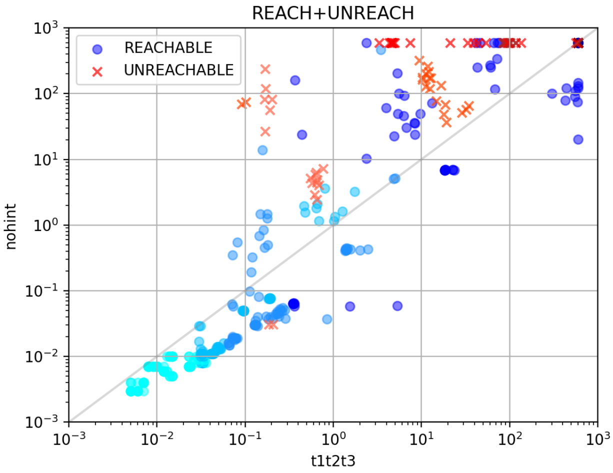
Evaluation of Hint Constraint Combinations





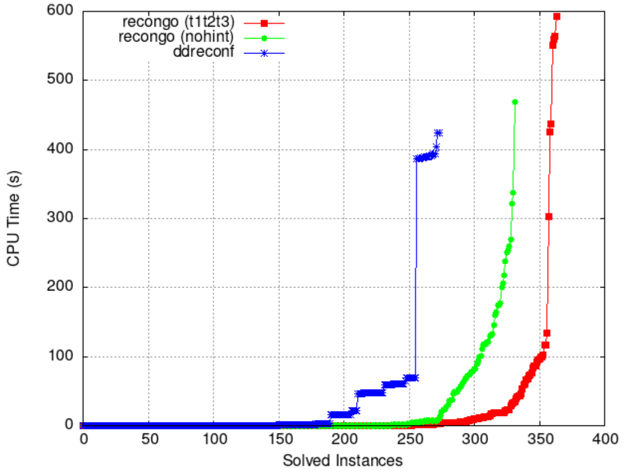
- For the reachable problems, `t2` solved the most **299** instances.
- For the unreachable problems, `t2t3` solved the most **69** instances.
- The hint constraint `t3` was confirmed to be effective in unreachable instances.
- In total, `t1t2t3` solved the most **363** instances.

Analysis on the hint `t1t2t3`: CPU Scatter Plot



- A darker color indicates larger instances.
- For the unreachable instances, `t1t2t3` was able to quickly determine unreachability in larger-scale instances.

Comparison with other approaches



- The proposed method, **without any hints**, solved 58 more instances than the ZDD-based solver **ddreconf**.
- **With hint constraints**, the method solved 90 more instances than **ddreconf**.

Summary

This presentation discussed a method for solving DSRP using ASP

1. Comparative evaluation of existing ASP encodings for solving DSP.
2. Development of ASP encodings for solving DSRP
3. Development of a benchmark set for DSRP (Total 442 instances)
4. Evaluation of the Developed ASP Encoding for DSRP

Future Work

1. Creation of benchmark problems for other adjacent relations
2. Development of encodings for other adjacent relations
3. Application to Secure Dominating Set
4. Application to Eternal Dominating Set