Hamiltonian Cycle Reconfiguration with Answer Set Programming

Takahiro Hirate[1] Mutsunori Banbara[1] Katsumi Inoue[2] Xiao-Nan Lu[3] Hidetomo Nabeshima[4] Torsten Schaub[5] Takehide Soh[6] Naoyuki Tamura[6]

[1]Nagoya University [2]National Institute of Informatics [3]Gifu University [4]University of Ymanashi [5]Universität Potsdam [6]Kobe University

JELIA2023@Dresden

Combinatorial reconfiguration

Combinatorial reconfiguration is to study the structure and properties (e.g., reachability) of solution spaces of combinatorial problems.

- **Combinatorial Reconfiguration Problems** (CRPs) are defined as the task of deciding, for a given combinatorial problem and two of its feasible solutions, whether one is reachable from another via a sequence of adjacent feasible solutions.

Combinatorial reconfiguration

Combinatorial reconfiguration is to study the structure and properties (e.g., reachability) of solution spaces of combinatorial problems.

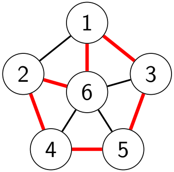
- **Combinatorial Reconfiguration Problems** (CRPs) are defined as the task of deciding, for a given combinatorial problem and two of its feasible solutions, whether one is reachable from another via a sequence of adjacent feasible solutions.
- A great effort has been made to investigate the theoretical aspects of CRPs over the last decade.
- For many NP-complete problems, their reconfigurations have been shown to be **PSPACE-complete**:
  - SAT reconfiguration [Gopalan+,'09]
  - Graph coloring reconfiguration [Bonsma+,'09]
  - **Hamiltonian cycle reconfiguration** [takaoka,'18], and many others.

However, little attention has been paid so far to its practical aspects.

Question

How many transitions we need under the transition constraint 3-opt, which enforces that exact 3 edges differ in each transition $X_t \Rightarrow X_{t+1}$?

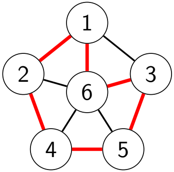
start state



X_0

$\Rightarrow \quad \cdots \qquad\qquad \cdots \quad \Rightarrow$

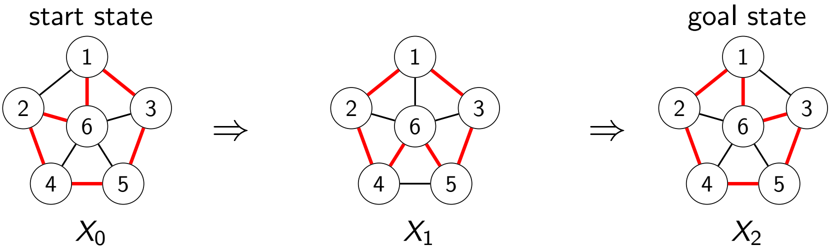
goal state



$X_?$

Question

How many transitions we need under the transition constraint 3-opt, which enforces that exact 3 edges differ in each transition $X_t \Rightarrow X_{t+1}$?



- The goal state is reached from the start state with 2 transitions.
- Each state X_i satisfies the constraints of HCP.
- Each transition ($\Rightarrow$) satisfies the k-opt constraint, in this case $k = 3$.
  - From X_0 to X_1, three edges 1–6, 2–6, and 4–5 are removed.

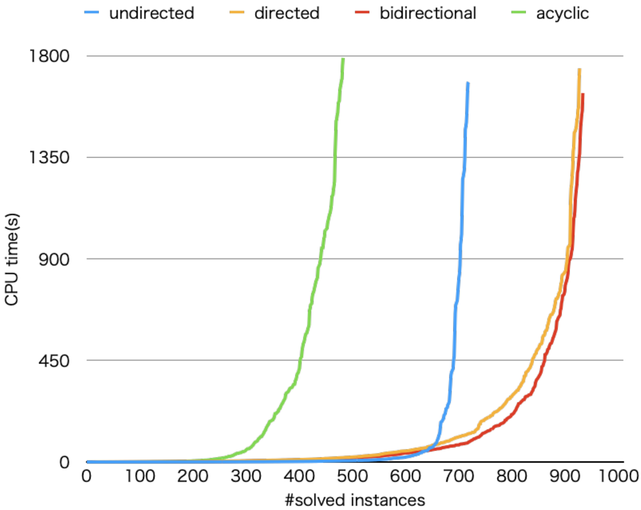
Part 1: Main contributions for HCP

We develop two ASP encodings for undirected HCP solving.

- **bidirectional encoding** and **acyclic encoding**
- They are based on the idea of a SAT encoding [Soh+,JELIA'14] that transforms undirected graph problems into directed ones by mapping each edge $u - v$ to one of its directional edges $u \to v$ and $v \to u$.

- Our empirical analysis considers all 1,001 HCP instances, which are publicly available from Flinders Hamiltonian Cycle Project (FHCP)
- The bidirectional encoding performs better than traditional encodings.
- We establish the competitiveness of our declarative approach by contrasting it to
  1. the award-winning solvers of the FHCP challenge,
  2. the 1st place solver of XCSP competition,
  3. a state-of-the-art SAT encoding for HCP solving [Heule,'21].

Cactus plot of HCP solving



- The **bidirectional** encoding solved the most, namely 934 instances.
- Followed by 928 of **directed**, 719 of **undirected**, and 483 of **acyclic**.

The award-winning solvers of the FHCP challenge

Rank	Team	#Solved	Method
1	INRIA, France	985	CPLEX
2	IBM, United Kingdom	614	SAT
3	King Saud University, Saudi Arabia	488	unknown
4	TU Darmstadt, Germany	464	unknown
5	Independent Researcher	385	unknown

[2] http://fhcp.edu.au/fhcpcs

The award-winning solvers of the FHCP challenge

Rank	Team	#Solved	Method
1	INRIA, France	985	CPLEX
	The bidirectional encoding (proposal)	**934**	**ASP**
2	IBM, United Kingdom	614	SAT
3	King Saud University, Saudi Arabia	488	unknown
4	TU Darmstadt, Germany	464	unknown
5	Independent Researcher	385	unknown

Our declarative approach can be highly competitive in performance.

[2] http://fhcp.edu.au/fhcpcs

Comparison with other approaches

CPU times(s) on all HCP instances of the XCSP 2019 competition

Instances	**ASP** (proposal)	*PicatSAT* (*xcsp_picat*)	SAT encoding [Heule,'21]
graph48	**0.752**	68.718	62.920
graph162	**7.500**	45.849	44.440
graph171	**10.383**	15.809	10.390
graph197	**0.342**	78.241	12.970
graph223	125.580	201.394	**22.600**
graph237	**0.306**	121.177	16.580
graph249	**0.956**	75.776	1.380
graph252	266.701	95.879	**9.950**
graph254	2.717	73.901	**2.660**
graph255	83.760	87.443	**6.110**
Average ratio	1.00	83.33	18.54

- Our bidirectional encoding is 83 times faster in average than *PicatSAT* and 18 times faster than the SAT encoding.

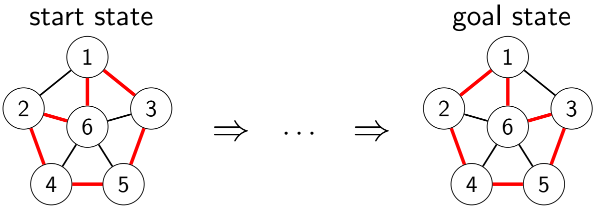
Part 2: Main contributions for HCRP

1. We extend our bidirectional encoding to solving HCRP, which is subsequently solved by an ASP-based CRP solver **recongo** [Yamada+,JELIA'23].
2. We develop three hint constraints to accelerate HCRP solving.
3. We create a new benchmark set of HCRP consisting of 948 HCRP instances, in which 431 are reachable and 517 are unreachable.

- The extended encoding for HCRP solving can manage to determine the reachability of 882 out of 948 instances.
- Furthermore, it is able to find shortest reconfiguration sequences of length 28 in about 200 seconds in average.

ASP fact format of HCRP instances

HCRP instances are represented as ASP facts in a standard way.



ASP fact format

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

Full encoding for HCRP solving (proposal)

```
#program base.
:- not 1 { in(X,Y,0) ; in(Y,X,0) } 1, start(X,Y).

#program step(t).
{ in(X,Y,t) ; in(Y,X,t) } 1 :- edge(X,Y).
:- not 1 { in(X,_,t) } 1, node(X).
:- not 1 { in(_,X,t) } 1, node(X).
reached(s,t).
reached(Y,t) :- reached(X,t), in(X,Y,t).
:- not reached(X,t), node(X).
:- not X < Y, in(s,X,t), in(Y,s,t).

removed(X,Y,t) :- in(X,Y,t-1), not in(X,Y,t), not in(Y,X,t), t>0.
:- not k { removed(_,_,t) } k, t>0.

#program check(t).
:- not 1 { in(X,Y,t) ; in(Y,X,t) } 1, goal(X,Y), query(t).
```

- The encoding consists of three parts: base, step(t), and check(t).

#program step(t)

```
{ in(X,Y,t) ; in(Y,X,t) } 1 :- edge(X,Y).
:- not 1 { in(X,_,t) } 1, node(X).
:- not 1 { in(_,X,t) } 1, node(X).

reached(s,t).
reached(Y,t) :- reached(X,t), in(X,Y,t).
:- not reached(X,t), node(X).

removed(X,Y,t) :- in(X,Y,t-1), not in(X,Y,t), not in(Y,X,t), t>0.
:- not k { removed(_,_,t) } k, t>0.
```

- The constant t is a parameter representing each step in a transition sequence.
- The auxiliary atom in(X,Y,t) is intended to represent that the directed edge X→Y is in a Hamiltonian cycle at step t.

#program step(t)

```
1  { in(X,Y,t) ; in(Y,X,t) } 1 :- edge(X,Y).
2  :- not 1 { in(X,_,t) } 1, node(X).
3  :- not 1 { in(_,X,t) } 1, node(X).
4
5  reached(s,t).
6  reached(Y,t) :- reached(X,t), in(X,Y,t).
7  :- not reached(X,t), node(X).
8
9  removed(X,Y,t) :- in(X,Y,t-1), not in(X,Y,t), not in(Y,X,t), t>0.
10 :- not k { removed(_,_,t) } k, t>0.
```

- **Key idea:** The rule in (1), for each `edge(X,Y)`, introduces two atoms `in(X,Y,t)` and `in(Y,X,t)` and enforces that **at most one** of them is included in the Hamiltonian cycle.
- Although the at-most-one constraints are implied constraints, they gain some performance improvement for HCP solving.

#program step(t)

```
1  { in(X,Y,t) ; in(Y,X,t) } 1 :- edge(X,Y).
2  :- not 1 { in(X,_,t) } 1, node(X).
3  :- not 1 { in(_,X,t) } 1, node(X).
4
5  reached(s,t).
6  reached(Y,t) :- reached(X,t), in(X,Y,t).
7  :- not reached(X,t), node(X).
8
9  removed(X,Y,t) :- in(X,Y,t-1), not in(X,Y,t), not in(Y,X,t), t>0.
10 :- not k { removed(_,_,t) } k, t>0.
```

- (2)–(3): The degree constraints
- (5)–(7): The connectivity constraints

#program step(t)

```
 1  { in(X,Y,t) ; in(Y,X,t) } 1 :- edge(X,Y).
 2  :- not 1 { in(X,_,t) } 1, node(X).
 3  :- not 1 { in(_,X,t) } 1, node(X).
 4
 5  reached(s,t).
 6  reached(Y,t) :- reached(X,t), in(X,Y,t).
 7  :- not reached(X,t), node(X).
 8
 9  removed(X,Y,t) :- in(X,Y,t-1), not in(X,Y,t), not in(Y,X,t), t>0.
10  :- not k { removed(_,_,t) } k, t>0.
```

- (9)–(10): The *k*-opt transition constraints
- (9): The auxiliary atom `removed(X,Y,t)` represents that the directed edge X→Y is removed from a Hamiltonian cycle from step `t-1` to `t`.
- (10): The rule enforces that exactly `k` edges in a Hamiltonian cycle are removed at each step `t`.

CPU time(s) of finding shortest transition sequences

Length	#Instance	CPU time(s) average	maximum	minimum
28	4	200.725	290.375	130.622
14	10	148.754	209.782	119.712
8	10	141.659	293.491	74.568
7	10	2.304	2.652	1.994
6	44	26.723	67.564	8.663
4	110	14.200	83.747	0.889
3	64	6.048	25.496	1.100
2	124	1.343	2.207	0.274
1	47	0.669	2.036	0.434

- Our encoding was able to find the solutions of length 28 in about 200 seconds in average.

The most relevant related fields

Combinatorial reconfiguration is

- to study the **solution spaces** of combinatorial problems,
- to decide whether there are sequences of feasible solutions that have special properties, such as **reachability**, **connectivity**, and **diameter**:

$$X_s = X_0 \Rightarrow X_1 \Rightarrow X_2 \Rightarrow \cdots \Rightarrow X_\ell = X_g$$

where X_s and X_g are optional.

- In contrast, **BMC** [Biere,'09] is to study properties (e.g., safety and liveness) of state transition systems and to decide whether there is no sequence for which X_s is a start state and X_g is an error state expressed by rich temporal logic.
- **Classical planning** [Kautz and Selman,'92] is to develop action plans for more practical applications and to decide whether there are sequences for which X_s is a start state and X_g is a goal state.

The relationship between those fields has not been well investigated.

Conclusion

We presented an ASP-based approach to solving the Hamiltonian cycle reconfiguration problem.

- All source code is available from:

  `https://github.com/banbaralab/hcr`.

Future work

- Solving the diameter problems of Hamiltonian cycle reconfiguration.
- Applying our declarative approach to a wide range of combinatorial reconfiguration problems.