Large Neighborhood Prioritized Search for Combinatorial Optimization with Answer Set Programming

Irumi Sugimori[1] Katsumi Inoue[2] Hidetomo Nabeshima[3]

Torsten Schaub[4] Takehide Soh[5] Naoyuki Tamura[5] Mutsunori Banbara[1]

[1]Nagoya University [2]National Institute of Informatics

[3]University of Yamanashi [4]Universität Potsdam [5]Kobe University

KR2024@Hanoi
November 7th, 2024

Outline

Large Neighborhood Search (LNS) is a hybrid between systematic and stochastic local search.

Main contributions

1. We propose **Large Neighborhood Prioritized Search (LNPS)**.
   - LNPS only fixes truth values heuristically, rather than rigidly.
   - We use heuristics to prioritize the search and to guide it.
2. We develop the *heulingo* solver which is an implementation of LNPS based on **Answer Set Programming (ASP)**.

- We succeeded in significantly enhancing the solving performance of *clingo* for ASP optimization.
- *heulingo* demonstrated that LNPS allows us to compete with ASP-based adaptive LNS by Eiter et al [KR'22,AIJ'24].

Background

Systematic search and Stochastic Local Search (SLS) are two major methods for solving Combinatorial Optimization Problems (COPs).

- Each method has strengths and weaknesses.
  - Systematic search can prove the optimality of solutions, but in general, it does not scale to large instances.
  - SLS can find near-optimal solutions within a reasonable amount of time, but it cannot guarantee the optimality of solutions.
- Therefore, there has been an increasing interest in the development of **hybrids** between systematic search and SLS [Hoos+,'15].

Large Neighborhood Search (LNS; [Shaw,'98]) is one of the most studied hybrids in recent years.

Large Neighborhood Search (LNS; [Shaw,'98])

LNS is an SLS-based metaheuristic that starts with an initial solution and then iteratively tries to find better solutions by alternately **destroying** and **repairing** a current solution.

☺ LNS has been so far successfully applied in the areas of routing and scheduling problems:

- multi-agent path finding [Li+,'21; Phan+,'24; Tan+,'24]
- timetabling [Kiefer+,'17; Demirovic+,'17]
- test laboratory scheduling [Geibinger+,'21], and many others

☺ Since the **repair** operators can be implemented with **systematic solvers**, LNS has been shown to be highly compatible with

- **ASP** [Eiter+,'22a,'22b,'24]
- MIP [Fischetti+,'03; Danna+,'05]
- CP [Shaw,'98; Dekker+,'18; Björdal+,'19,'20]

Motivation and Proposal

- However, LNS strongly depends on the **destroy** operators since the undestroyed part is fixed rigidly.
- In general, LNS cannot guarantee the optimality of solutions.

Challenge

It is still challenging to develop a universal algorithm for ASP optimization which has the advantages of both systematic search and SLS.

We propose Large Neighborhood Prioritized Search (LNPS).

- Since the undestroyed part is not fixed rigidly but heuristically (i.e., **variability**), LNPS allows for flexible search without strongly depending on the destroy operators.
- Moreover, LNPS can guarantee the **optimality** of solutions.

Large Neighborhood Prioritized Search (LNPS)

LNPS is an SLS-based metaheuristic that starts with an initial solution and then iteratively tries to find better solutions by alternately **destroying** a current solution and reconstructing it with **prioritized search**.

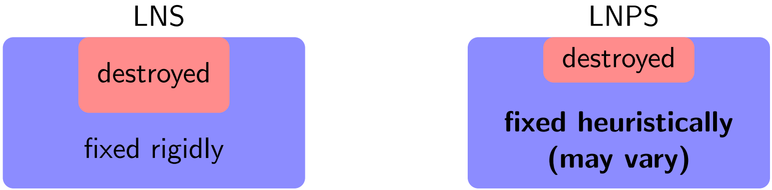
- **Prioritized search** allows us to guide the search by modifying its decision heuristic.
- Prioritized search can be easily implemented with heuristic-driven ASP solving (e.g., `#heuristic` statement).

Algorithm 1 LNPS

Input: a feasible solution x

$x^* \leftarrow x$
while stop criterion is not met **do**
    $x^t \leftarrow \mathit{prioritized\text{-}search}(\mathit{destroy}(x))$
    if $\mathit{accept}(x^t, x)$ **then**
        $x \leftarrow x^t$
    end if
    if $c(x^t) < c(x^*)$ **then**
        $x^* \leftarrow x^t$
    end if
end while
return x^*

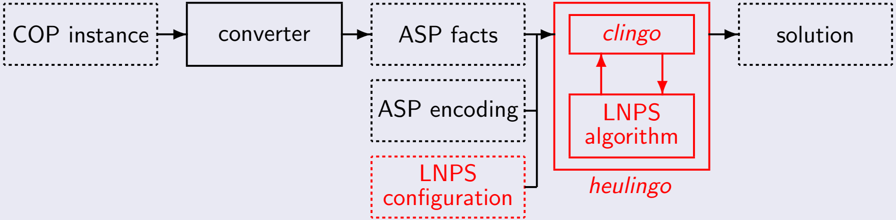
The main differences of LNPS from LNS



1. The undestroyed part is **fixed rigidly**.
2. Due to the strong dependency on destroy operators, the percentage of destruction should be **sufficiently large**.
3. The optimality cannot be guaranteed in general.

1. The undestroyed part is **fixed heuristically**.
2. Due to this **variability**, the percentage of destruction can be **smaller**.
3. The **optimality** can be guaranteed because of prioritized search.

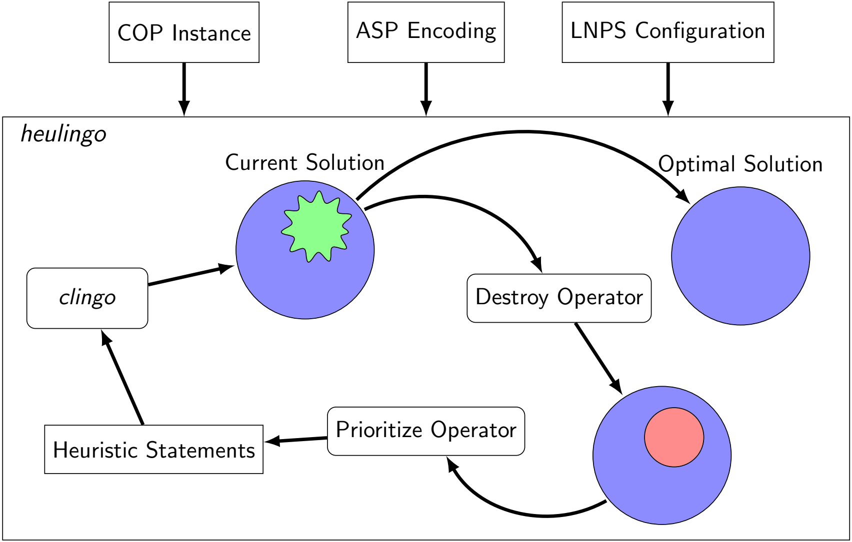
heulingo: an ASP-based implementation of LNPS



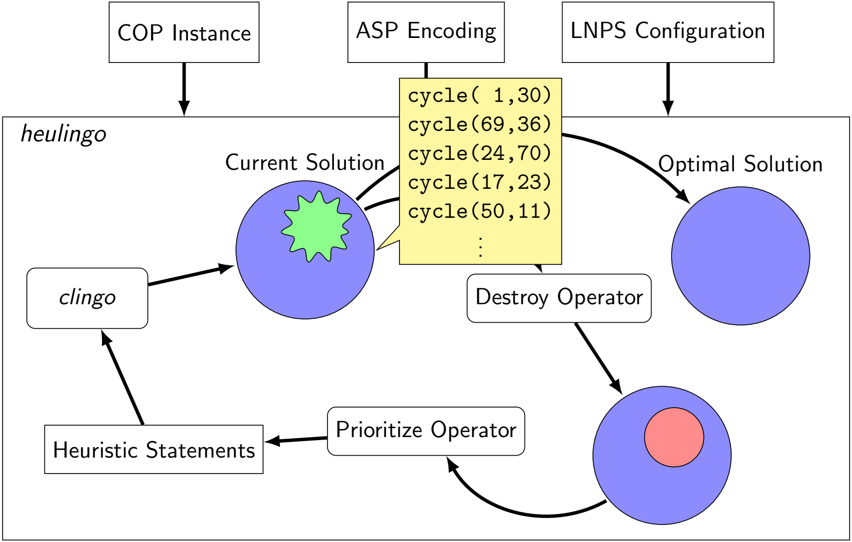
1. *heulingo* reads an instance of Combinatorial Optimization Problems (COPs) and an LNPS configuration in ASP fact format.
2. In turn, these facts are combined with an ASP encoding for COP solving, which are afterward solved by the LNPS algorithm powered by ASP solvers, in our case *clingo*.

- The LNPS algorithm can be compactly implemented by using *clingo*'s multi-shot ASP solving and `#heuristic` statements.

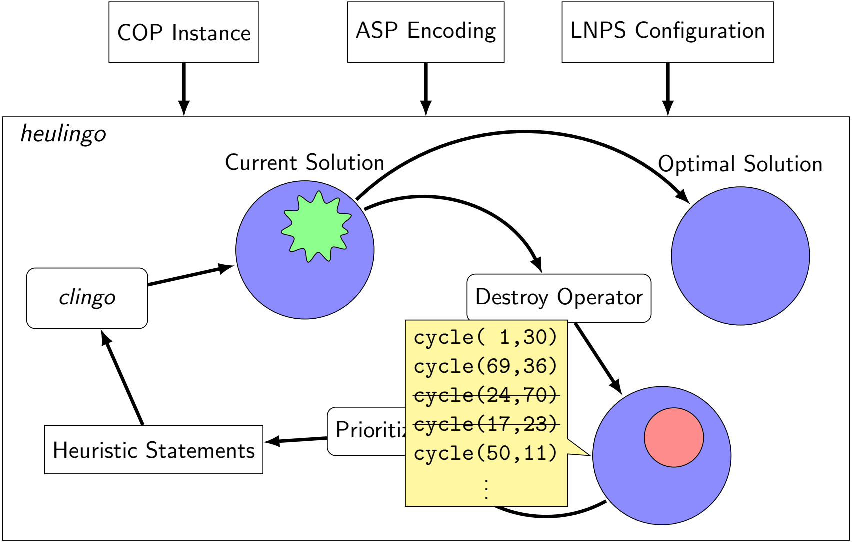
LNPS algorithm with clingo



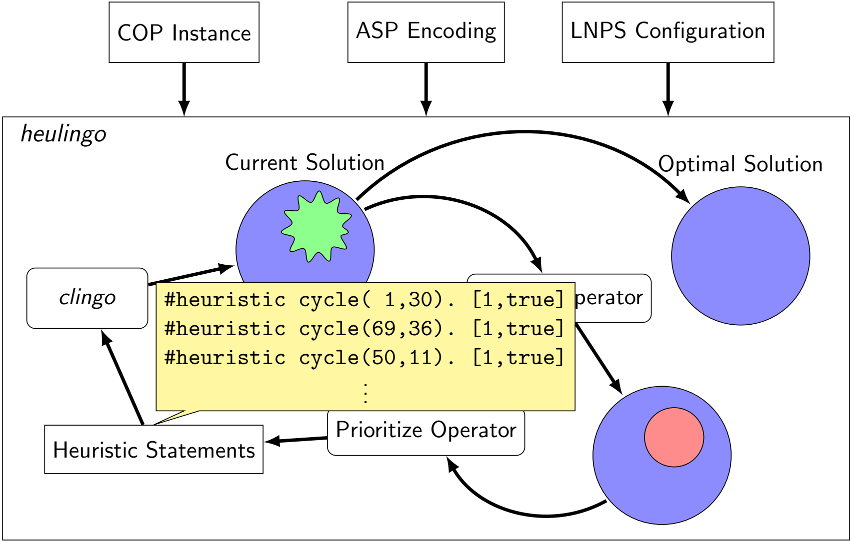
LNPS algorithm with clingo



LNPS algorithm with clingo



LNPS algorithm with clingo



The main features of heulingo

- **Variability and optimality**: Due to the variability of neighborhoods, *heulingo* allows for flexible search without strongly depending on the destroy operators, and can guarantee the optimality of solutions.
- **Expressiveness**: *heulingo* relies on ASP's expressive language that is well suited for modeling combinatorial optimization problems.
- **Domain heuristics**: *heulingo* allows for easy incorporation of domain heuristics in a declarative way.
- **Usability and Compatibility**: *heulingo* can deal with any ASP encoding for optimization without any modification. All we have to do is to add an LNPS configuration.

The main features of heulingo

- **Variability and optimality**: Due to the variability of neighborhoods, *heulingo* allows for flexible search without strongly depending on the destroy operators, and can guarantee the optimality of solutions.
- **Expressiveness**: *heulingo* relies on ASP's expressive language that is well suited for modeling combinatorial optimization problems.
- **Domain heuristics**: *heulingo* allows for easy incorporation of domain heuristics in a declarative way.
- **Usability and Compatibility**: *heulingo* can deal with any ASP encoding for optimization without any modification. All we have to do is to add an LNPS configuration.

For efficiency

The question is whether the *heulingo* approach can

1. enhance the performance of *clingo*,
2. match the performance of the (adaptive) LNS heuristic.

Experiments

We carry out experiments on a challenging benchmark set [Eiter+,KR'22].

- The benchmark set consists of 55 instances in total:
  1. Traveling Salesperson Problem
  2. Social Golfer Problem
  3. Sudoku Puzzle Generation
  4. Weighted Strategic Companies
  5. Shift Design
- We compare four solvers:
  1. *heulingo* (LNPS): an ASP-based implementation of LNPS
  2. *heulingo* (LNS): an ASP-based implementation of LNS
  3. *clingo*-5.6.2 [1]
  4. *ALASPO*: an ASP-based implementation of adaptive LNS [2]

[1] `https://potassco.org/clingo/`

[2] `http://www.kr.tuwien.ac.at/research/projects/bai/kr22.zip`

Benchmark results

Summary of results

heulingo (LNPS) was able to find the best bounds on average for **37** among all 55 instances (67% in a total).

- 🙂 *heulingo* (LNPS) succeeded in improving the bounds of *clingo* by
  35.1% for traveling salesperson problem,
  24.8% for social golfer problem,
  34.0% for sudoku puzzle generation, and
  7.7% for weighted strategic companies.
- 🙂 *heulingo* (LNPS) performed slightly better on average than *ALASPO*.
- 😵 On the other hand, on shift design, *heulingo* (LNPS) does not match the performance of *heulingo* (LNS) and *ALASPO*.

Results on Traveling Salesperson Problem (TSP)

Instance	*clingo*	*heulingo* (LNS)			*heulingo* (LNPS)			*ALASPO*		
		avg.	min.	max.	avg.	min.	max.	avg.	min.	max.
dom_rand_70_300_1155482584_3	591	438.7	427	454	**386.3**	383	390	424.3	397	444
rand_70_300_1155482584_0	552	371.7	351	393	**326.3**	320	333	367.7	349	384
rand_70_300_1155482584_11	606	447.0	436	454	**386.3**	381	392	447.0	433	466
rand_70_300_1155482584_12	540	386.3	364	406	**344.7**	341	349	380.7	371	386
rand_70_300_1155482584_14	567	393.7	388	404	**357.7**	355	359	397.0	382	409
rand_70_300_1155482584_3	575	444.7	428	458	**408.7**	398	419	450.0	445	459
rand_70_300_1155482584_4	649	476.7	464	483	**423.0**	419	428	475.7	470	479
rand_70_300_1155482584_5	601	420.0	397	449	**367.3**	361	374	396.3	393	401
rand_70_300_1155482584_7	604	435.0	429	446	**406.3**	405	407	442.0	428	462
rand_70_300_1155482584_8	553	441.7	426	461	**387.0**	385	389	427.0	412	441
rand_70_300_1155482584_9	546	414.3	391	427	**368.3**	365	372	403.3	402	405
rand_80_340_1159656267_0	714	464.7	446	492	**410.7**	410	411	479.0	476	484
rand_80_340_1159656267_10	654	494.0	480	503	**441.3**	438	445	499.7	495	507
rand_80_340_1159656267_11	731	528.7	509	539	**464.0**	458	475	520.7	497	534
rand_80_340_1159656267_13	686	467.7	437	487	**431.3**	426	440	471.3	466	477
rand_80_340_1159656267_15	720	492.7	484	499	**439.3**	435	446	478.0	471	488
rand_80_340_1159656267_16	667	546.7	525	559	**496.3**	492	499	558.7	551	571
rand_80_340_1159656267_17	737	501.3	492	509	**449.0**	443	457	472.3	461	479
rand_80_340_1159656267_18	674	484.7	466	510	**418.7**	417	420	488.0	477	506
rand_80_340_1159656267_4	590	471.7	442	511	**418.3**	413	421	462.3	460	466
Average rate	1.000	0.728			**0.649**			0.721		

- *heulingo* (LNPS) is able to find the best bounds on average for all 20 instances.
- *heulingo* (LNPS) succeeds in improving the bounds of *clingo* by 35.1% on average.

The details of experiments on TSP

We execute *heulingo* in 3 runs for each instance using the **random destruction**, which would be one of the most simple LNPS configurations.

- The **percentage** of the random destruction is set to
  - {1%, 3%, 5%} for *heulingo* (LNPS)
  - {28%, 30%, 32%} for *heulingo* (LNS)
- The **solve-limit** of *heulingo* is set to
  - 1,210,000 conflicts for finding an initial solution,
  - 800,000 conflicts for each iteration
- **Time-limit**: 300s for each instance
- **Environment**: Mac OS Apple M1 Ultra, 128GB memory
- **Note**: We execute *ALASPO* in 3 runs for each instance with the best portfolio presented in [Eiter+,KR'22].

ASP fact format of LNPS configuration

Random destruction heuristic

randomly destroys a current solution, and the undestroyed part is kept as much as possible in each iteration.

```
#program config.
_lnps_project(cycle,2).
_lnps_destroy(cycle,2,3,p(5)).
_lnps_prioritize(cycle,2,1,true).
```

- The atom `_lnps_project(cycle,2)` means that the atoms of `cycle/2` belonging to an answer set are subject to LNPS.

ASP fact format of LNPS configuration

Random destruction heuristic

randomly destroys a current solution, and the undestroyed part is kept as much as possible in each iteration.

```
#program config.
_lnps_project(cycle,2).
_lnps_destroy(cycle,2,3,p(5)).
_lnps_prioritize(cycle,2,1,true).
```

- The atom `_lnps_destroy(cycle,2,3,p(5))` means that 5% of a current solution characterized by `cycle/2` are destroyed.
- The 3rd argument $3 = (11)_2$ represents that all possible 2 arguments (`X`,`Y`) of `cycle(X,Y)` are subject to destruction.

ASP fact format of LNPS configuration

Random destruction heuristic

randomly destroys a current solution, and the undestroyed part is kept as much as possible in each iteration.

```
#program config.
_lnps_project(cycle,2).
_lnps_destroy(cycle,2,3,p(5)).
_lnps_prioritize(cycle,2,1,true).
```

- The atom `_lnps_prioritize(cycle,2,1,true)` means that the undestroyed part is kept as much as possible.
- Technically, this atom corresponds to *clingo*'s heuristic statement `#heuristic cycle(X,Y). [1,true]`.

Discussion

In general, it can be a hard and time-consuming task to find the best configuration for LNPS.

- The configurations were obtained in our preliminary experiments.
- We used less destruction than used for LNS in [Eiter+,AAAI'22].
- We roughly estimated the number of conflicts on which *clingo* is stuck, and then used it for the stop criterion of prioritized search.

Future Work

We plan to extend *heulingo* for **adaptive LNPS**, which selects in each iteration a potentially more effective destroy/prioritize operators.

Conclusion

We proposed Large Neighborhood Prioritized Search (LNPS) for solving COPs, and presented an ASP-based implementation of LNPS.

All source code is available from: