Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie
AGH University of Science
and Technology

Formal Representation and Automatic Synthesis of Local Search Neighborhoods

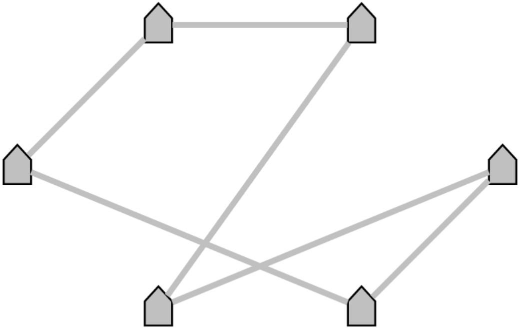
dr inż. Mateusz Ślażyński

AGH University of Krakow
Department of Applied Computer Science
al. A. Mickiewicza 30,
30-059 Krakow,
Poland
https://home.agh.edu.pl/~mslaz

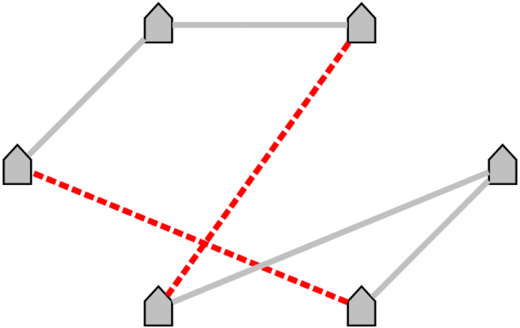
14.03.2024 r.

Example: Traveling Salesperson Problem

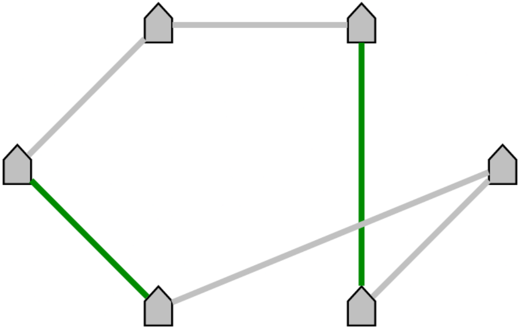
Example: Traveling Salesperson Problem



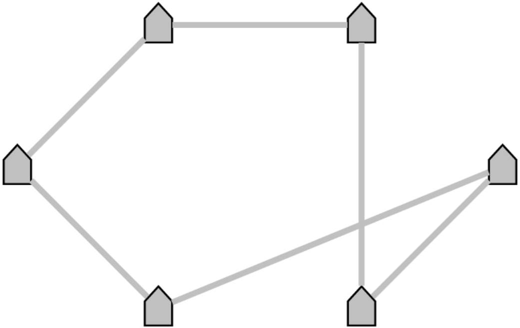
Example: Traveling Salesperson Problem



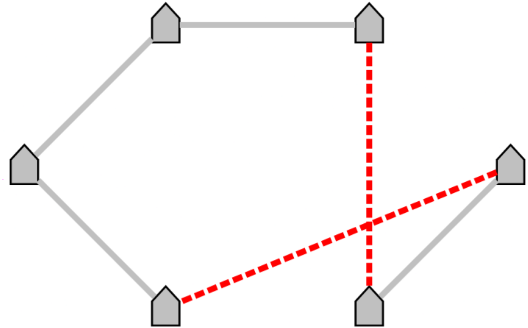
Example: Traveling Salesperson Problem



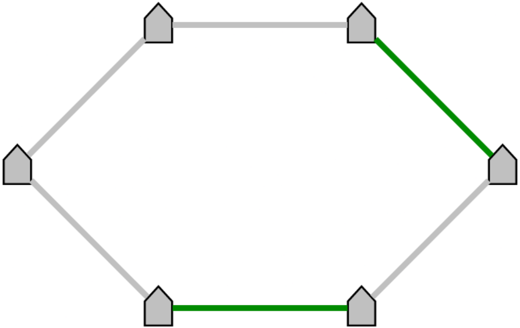
Example: Traveling Salesperson Problem



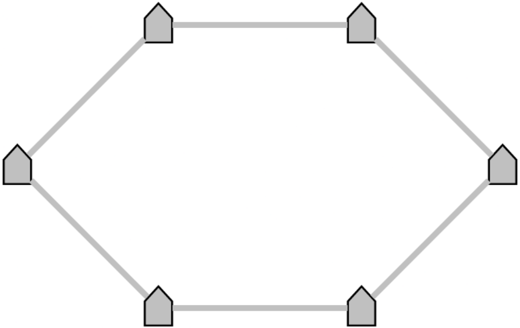
Example: Traveling Salesperson Problem



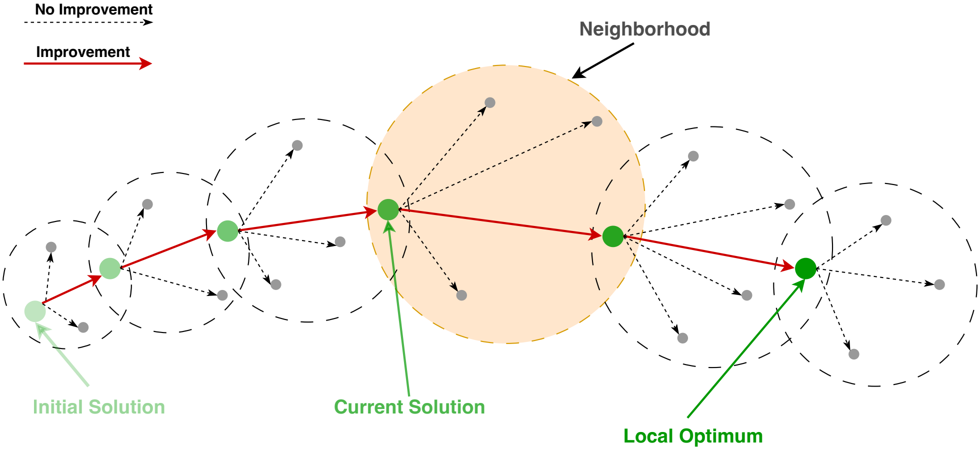
Example: Traveling Salesperson Problem



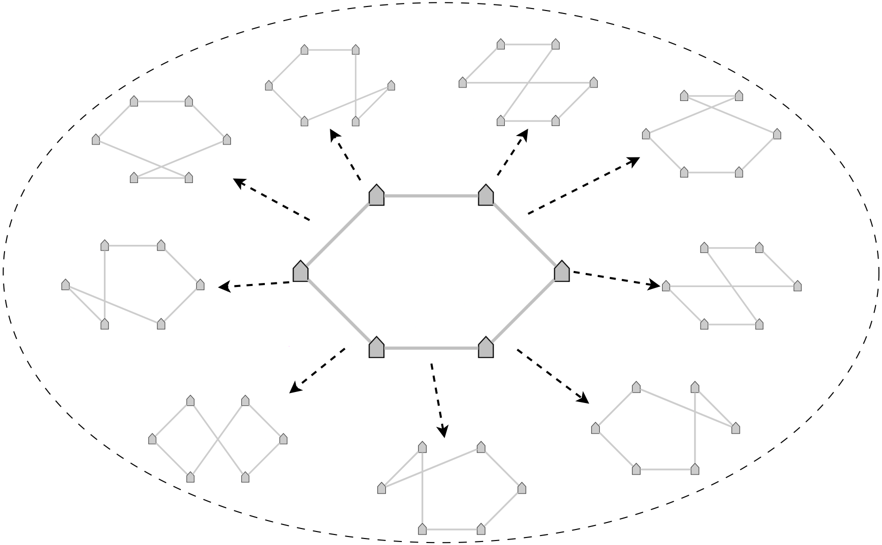
Example: Traveling Salesperson Problem



Local Search



2-opt Neighborhood



Open Problems

- Local minima

Example

Choosing in each move the best neighbor will (almost) always terminate in a local optimum. We need smarter strategies (called **meta-heuristics**) to achieve better results. Works by *Bezerra, López-Ibáñez and Stützle* discuss automatic design of such strategies.

Open Problems

- Local minima
- **Neighborhood**

Example

There are many possible neighborhoods for a given problem, how to choose the correct one?

Open Problems

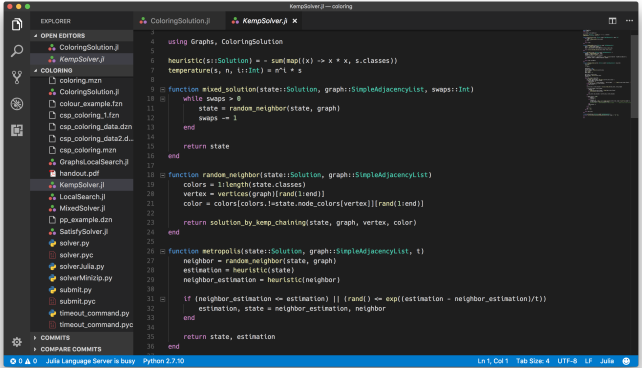
- Local minima
- **Neighborhood**
- Search order

Example

Often neighborhood is too big to be enumerated and we need to process it partially in an order defined by **heuristic**. Very useful in greedy algorithms, quickly leading to good-enough solutions.

Open Problems

- Local minima
- **Neighborhood**
- Search order
- **Model layer**



Domain

Discrete Optimization

Involves many practical problems:

- designing warehouse layout

Domain

Discrete Optimization

Involves many practical problems:

- designing warehouse layout
- designing circuit boards

Domain

Discrete Optimization

Involves many practical problems:

- designing warehouse layout
- designing circuit boards
- task assignment

Domain

Discrete Optimization

Involves many practical problems:

- designing warehouse layout
- designing circuit boards
- task assignment
- designing networks, e.g., streets, sewers

Domain

Discrete Optimization

Involves many practical problems:

- designing warehouse layout
- designing circuit boards
- task assignment
- designing networks, e.g., streets, sewers
- nurse scheduling

Domain

Discrete Optimization

Involves many practical problems:

- designing warehouse layout
- designing circuit boards
- task assignment
- designing networks, e.g., streets, sewers
- nurse scheduling
- **routing problems**

Research Problem

Goal

Automatic generation of **useful** neighborhood operators for given discrete optimization problems.

Research Problem

Goal

Automatic generation of **useful** neighborhood operators for given discrete optimization problems.

Proposed Solution

Defining the neighborhood relation based on a **declarative** problem model.

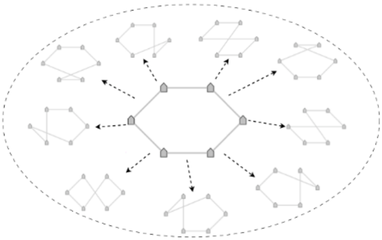
Research Problem

Goal

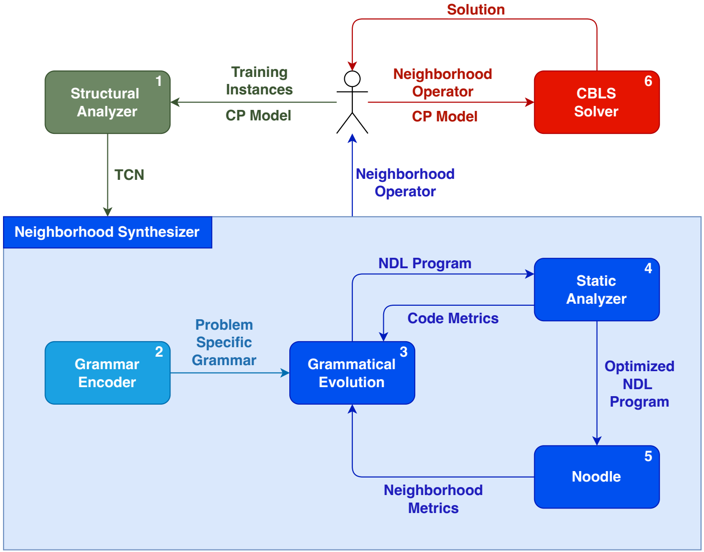
Automatic generation of **useful** neighborhood operators for given discrete optimization problems.

Proposed Solution

Defining the neighborhood relation based on a **declarative** problem model.

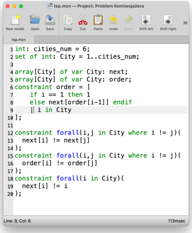


System Architecture



Representation

Model

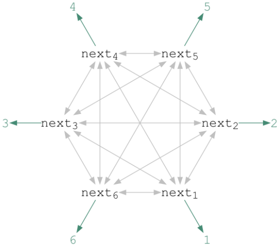


```
int: cities_num = 6;
set of int: City = 1..cities_num;

array[City] of var City: next;
array[City] of var City: order;
constraint order = [
    if i == 1 then 1
    else next[order[i-1]] endif
    | i in City
];

constraint forall(i,j in City where i != j)(
  next[i] != next[j]
);
constraint forall(i,j in City where i != j)(
  order[i] != order[j]
);
constraint forall(i in City)(
  next[i] != i
);
```

Typed Constraint Network



Neighborhood Operator

$$\mathit{GenCstrVars}(\mathit{diff}, \mathtt{n1}, \mathtt{n2}) \circ$$
$$\mathit{Map}(\mathtt{n1}, \mathtt{m1}, \mathtt{m2},$$
$$\quad \mathit{FilterVars}(\mathtt{n2}, \mathtt{m2}, \mathit{constrainted}_{\mathit{diff}}) \circ$$
$$\quad \mathit{SwapVals}(\mathtt{n2}, \mathtt{n1}) \circ$$
$$\quad \mathit{SwapVals}(\mathtt{m2}, \mathtt{n1}))$$

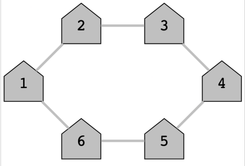
- **Mateusz Ślażyński**, Salvador Abreu, Grzegorz J. Nalepa
  Towards a Formal Specification of Local Search Neighborhoods From a Constraint Satisfaction Problem Structure
  GECCO, 2019

Representation: TSP Example

```
include "globals.mzn";

int: cities_num = 6;
set of int: City = 1..cities_num;

array[City] of var City: next;
constraint circuit(next);
```



next = [2,3,4,5,6,1]

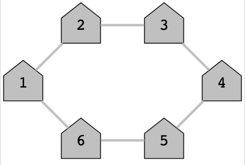
- there are 6 cities: `cities_num`
- an array of `next` variables, where `next[i]` is a city visited after visiting the `i`'th city
- variables should form a Hamiltonian cycle (*circuit*)

Representation: TSP Example

```
int: cities_num = 6;
set of int: City = 1..cities_num;

array[City] of var City: next;
array[City] of var City: order::auxiliary = [
  if i == 1 then 1
  else next[order[i-1]] endif
  | i in City
];

constraint forall(i,j in City where i != j)(
  next[i] != next[j]
);
constraint forall(i,j in City where i != j)(
  order[i] != order[j]
);
constraint forall(i in City)(
  next[i] != i
);
```



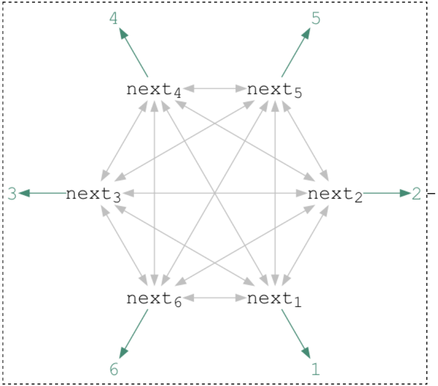

```
next  = [2,3,4,5,6,1]
order = [1,2,3,4,5,6]
```

Typed Constraint Network

$next_i \neq next_j$

$next_i \neq i$

$order_i \neq order_j$

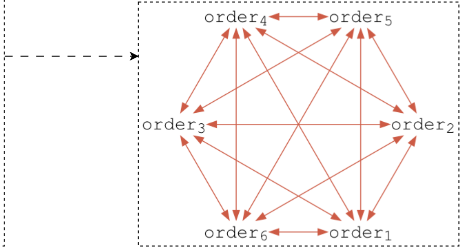


$D_{next} = \{1,2,3,4,5,6\}$

$D_{order} = \{1,2,3,4,5,6\}$

$I_{next} = \{1,2,3,4,5,6\}$

$I_{order} = \{1,2,3,4,5,6\}$



Idea

Declarative programming language designed to define neighborhood operators.

Neighborhood Definition Language

Idea

Declarative programming language designed to define neighborhood operators.

Features

- operates on the data stored in solution and the *Typed Constraint Network*
- non-deterministic — program returns just a single neighbor, but called many times will explore the whole neighborhood, returning different neighbor each time
- Tuing incomplete — predictable run-time, always terminating

Neighborhood Definition Language

- Selectors

Example

Select a single edge fro the TCN, e.g., corresponding to a constraint $next_i \neq next_j$.

Neighborhood Definition Language

- Selectors
- Filters

Example

Check whether the two variables do not represent two consecutive cities: $next_i \neq j$ and $next_j \neq i$.

Neighborhood Definition Language

- Selectors
- Filters
- Modifiers

Example

Assign the current value of $next_i$ to variable $next_j$.

Neighborhood Definition Language

- Selectors
- Filters
- Modifiers
- Higher-Order Operators

Example

For every constraint that is violated in the current solution, select its corresponding edge and perform a smaller NDL program.

$$SwapVals\colon \frac{\begin{array}{l}\exists x_1, x_2 \in X\colon \{b_1 \leftarrow x_1, b_2 \leftarrow x_2\} \subseteq \beta \\ \qquad \wedge\, \sigma(x_1) \in \mathit{dom}(x_2) \wedge \sigma(x_2) \in \mathit{dom}(x_1) \\ \qquad \wedge\, \Sigma_T(x_1) \notin A_T \wedge \Sigma_T(x_2) \notin A_T\end{array}}{\sigma \mid \beta \circ \mathit{SwapVals}(b_1, b_2) \longrightarrow [\sigma(x_2) \mapsto x_1, \sigma(x_1) \mapsto x_2]\sigma \mid \beta}$$

Figure: Formal definition of a *SwapVals* operator.

$$\begin{array}{l}\mathit{GenCstrVars}(\mathit{diff}, \mathtt{n1}, \mathtt{n2}) \circ \\ \mathit{Map}(\mathtt{n1}, \mathtt{m1}, \mathtt{m2}, \\ \qquad \mathit{FilterVars}(\mathtt{n2}, \mathtt{m2}, \mathit{constrained}_{\mathit{diff}}) \circ \\ \qquad \mathit{SwapVals}(\mathtt{n2}, \mathtt{n1}) \circ \\ \qquad \mathit{SwapVals}(\mathtt{m2}, \mathtt{n1}))\end{array}$$

Figure: Formal example of an NDL program.

NDL — Formal Specification vs Implementations

```
rule <k>S:Map | swap_vals(BVAR1:BindName, BVAR2:BindName) =>
        S | get_val(BVAR1, $x) | get_val(BVAR2, $y)
          | set_val(BVAR1, $y) | set_val(BVAR2, $x)
          | restore_env(ENV) ...</k>
     <env>ENV</env>
```

Figure: Modifier *SwapVals* defined in the K language.

```
  gen_cst_vars(neq, first_variable, second_variable)
| map(first_variable, b1, b2,
        filter_vars(second_variable, b2, !=)
        | swap_vals(first_variable, second_variable)
        | swap_vals(first_variable, b2)
    )
```

Figure: NDL program as implemented in the K language.

NDL — Formal Specification vs Implementations

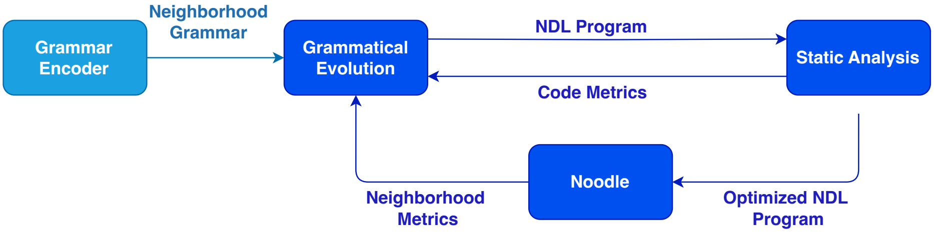
```
ndl_swap_values(OldSolution, Var1, Var2, NewSolution) :-
    ndl_get_value(OldSolution, Var1, Val1),
    ndl_get_value(OldSolution, Var2, Val2),
    ndl_set_value(OldSolution, Var1, Val2, Solution),
    ndl_set_value(Solution, Var2, Val1, NewSolution).
```

Figure: The *SwapVals* operator implemented in Prolog.

```
1.   constraint(all_diff_next, T0, T1) ∧
2.   iterate(T3 - T4, T0, (
2.1.   constraint(all_diff_next, T4, T1) ∧
2.2.   swap_values(T1, T0) ∧
2.3.   swap_values(T4, T0)))
```

Figure: An NDL program implemented in Prolog.

Operator Synthesis



- **Mateusz Ślażyński**, Salvador Abreu, Grzegorz J. Nalepa
  Generating Local Search Neighborhood with Synthesized Logic Programs
  International Conference on Logic Programming, 2019

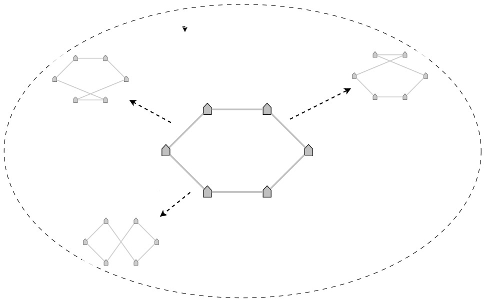
NDL Grammar

$$
\begin{aligned}
\langle \text{program} \rangle &\models \langle \text{query} \rangle \circ \langle \text{selection} \rangle \circ \langle \text{filtering} \rangle \circ \langle \text{body-update} \rangle \\
\langle \text{query} \rangle &\models \langle \text{generator} \rangle \circ \langle \text{query} \rangle \mid \langle \text{generator} \rangle \\
\langle \text{selection} \rangle &\models \langle \text{getter} \rangle \circ \langle \text{selection} \rangle \mid \epsilon \\
\langle \text{filtering} \rangle &\models \langle \text{filter} \rangle \circ \langle \text{filtering} \rangle \mid \epsilon \\
\langle \text{body-update} \rangle &\models \langle \text{modifier} \rangle \circ \langle \text{body-update} \rangle \mid \langle \text{combinator} \rangle \circ \langle \text{body-update} \rangle \\
&\quad \mid \langle \text{modifier} \rangle \mid \langle \text{combinator} \rangle \\
\langle \text{move} \rangle &\models \langle \text{selection} \rangle \circ \langle \text{filtering} \rangle \circ \langle \text{move-update} \rangle \\
\langle \text{move-update} \rangle &\models \langle \text{modifier} \rangle \circ \langle \text{move-update} \rangle \mid \langle \text{combinator} \rangle \circ \langle \text{move-update} \rangle \\
&\quad \mid \langle \text{modifier} \rangle \mid \langle \text{combinator} \rangle
\end{aligned}
$$

Figure: A basic skeleton of a formal grammar defining an NDL program. The red symbols are to be defined per problem.

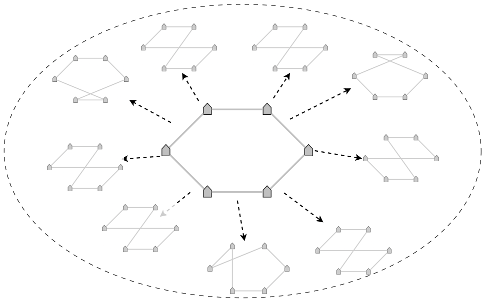
Fitness Criteria

1. The neighborhood should not be too small.



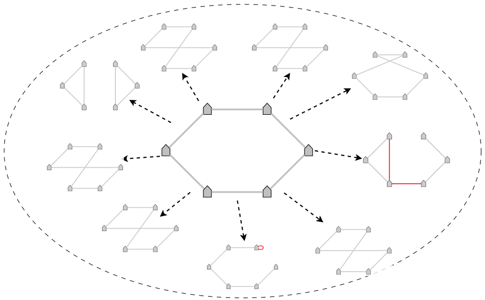
Fitness Criteria

1. The neighborhood should not be too small.
2. The duplicate neighbors are unwelcome.



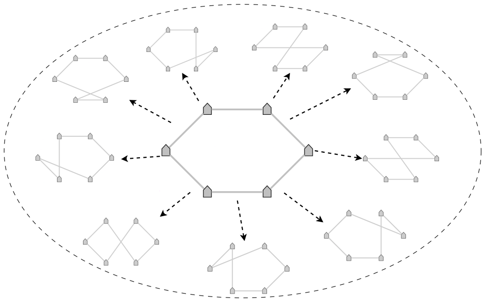
Fitness Criteria

1. The neighborhood should not be too small.
2. The duplicate neighbors are unwelcome.
3. **If solution satisfies a constraint, its neighbors also should satisfy the same constraint.**



Fitness Criteria

1. The neighborhood should not be too small.
2. The duplicate neighbors are unwelcome.
3. **If solution satisfies a constraint, its neighbors also should satisfy the same constraint.**
4. **Preferably** the neighborhood operator should modify a varying number of variables.



Example: TSP Neighborhood Synthesis

Training Input

- two small instances with 6 and 7 cities.
- random distances.
- for each instance, three random **feasible** initial solutions.

Example: TSP Neighborhood Synthesis

Training Input

- two small instances with 6 and 7 cities.
- random distances.
- for each instance, three random **feasible** initial solutions.

Process

- parameters: 50 generations, 500 programs each
- hardware: 64 CPU Cores and 64GB RAM
- duration: 30 minutes

Example: TSP Neighborhood Synthesis

Training Input

- two small instances with 6 and 7 cities.
- random distances.
- for each instance, three random **feasible** initial solutions.

Process

- parameters: 50 generations, 500 programs each
- hardware: 64 CPU Cores and 64GB RAM
- duration: 30 minutes

Results

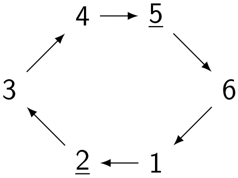
- four different neighborhoods matching four fitness functions
- including the 2-opt operator, when all the fitness criteria were considered

Example: 2-opt Operator

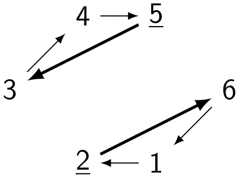
```
1.   constraint(all_diff_next, T0, T1) ∧
2.   iterate(T3 - T4, T0, (
2.1.   constraint(all_diff_next, T4, T1) ∧
2.2.   swap_values(T1, T0) ∧
2.3.   swap_values(T4, T0)))
```

Figure: The synthesized 2-opt neighborhood.

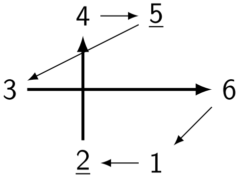
Example: 2-opt Operator



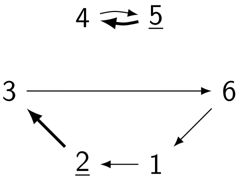
(a) Line `1.`
`T0 = 2`, `T1 = 5`



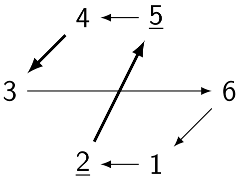
(b) Line `2.2.`
`T4 = 3`.



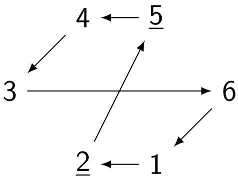
(c) Line `2.3.`
`T4 = 3`.



(d) Line `2.2.`
`T4 = 4`.

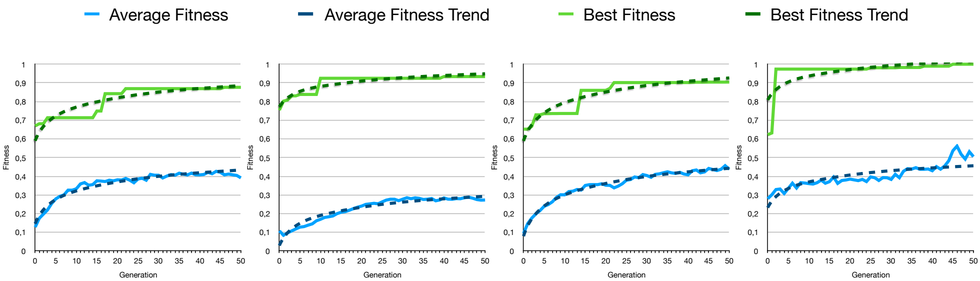


(e) Line `2.3.`
`T4 = 4`



(f) Line `2.1.`
`T4 = 5`.

Examplet: Fitness



(a) *2-opt* (b) *Basic 3-opt* (c) *3-swap* (d) *Even swap*

Figure: Fitness function improvement in four different experiment runs.

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.
- Using Automated Algorithm Design methods bridges the gap between users and advanced AI systems.

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.
- Using Automated Algorithm Design methods bridges the gap between users and advanced AI systems.
- I have presented a prototype AAD system to find useful neighborhoods given a declarative model of the considered problem.

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.
- Using Automated Algorithm Design methods bridges the gap between users and advanced AI systems.
- I have presented a prototype AAD system to find useful neighborhoods given a declarative model of the considered problem.
- Future research:

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.
- Using Automated Algorithm Design methods bridges the gap between users and advanced AI systems.
- I have presented a prototype AAD system to find useful neighborhoods given a declarative model of the considered problem.
- Future research:
    - experiments on other synthesis algorithms;

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.
- Using Automated Algorithm Design methods bridges the gap between users and advanced AI systems.
- I have presented a prototype AAD system to find useful neighborhoods given a declarative model of the considered problem.
- Future research:
  - experiments on other synthesis algorithms;
  - an efficient (low-level) implementation of the system;

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.
- Using Automated Algorithm Design methods bridges the gap between users and advanced AI systems.
- I have presented a prototype AAD system to find useful neighborhoods given a declarative model of the considered problem.
- Future research:
  - experiments on other synthesis algorithms;
  - an efficient (low-level) implementation of the system;
  - including meta-heuristics and heuristics in the process;

Summary

- Choosing correct Local-Search neighborhood is important to solve difficult optimization problems.
- Using Automated Algorithm Design methods bridges the gap between users and advanced AI systems.
- I have presented a prototype AAD system to find useful neighborhoods given a declarative model of the considered problem.
- Future research:
  - experiments on other synthesis algorithms;
  - an efficient (low-level) implementation of the system;
  - including meta-heuristics and heuristics in the process;
  - implementing more effective over-fitting countermeasures.

Related Papers

- **Mateusz Ślażyński**, Salvador Abreu, Grzegorz J. Nalepa
  Towards a Formal Specification of Local Search Neighborhoods From a Constraint Satisfaction Problem Structure
  The Genetic and Evolutionary Computation Conference, 2019
  140 p.
- **Mateusz Ślażyński**,
  Research Report on Automatic Synthesis of Local Search Neighborhood Operators
  International Conference on Logic Programming, 2019
  140 p.
- **Mateusz Ślażyński**, Salvador Abreu, Grzegorz J. Nalepa
  Generating Local Search Neighborhood with Synthesized Logic Programs
  International Conference on Logic Programming, 2019
  140 p.

Thanks

Thank you for your attention!