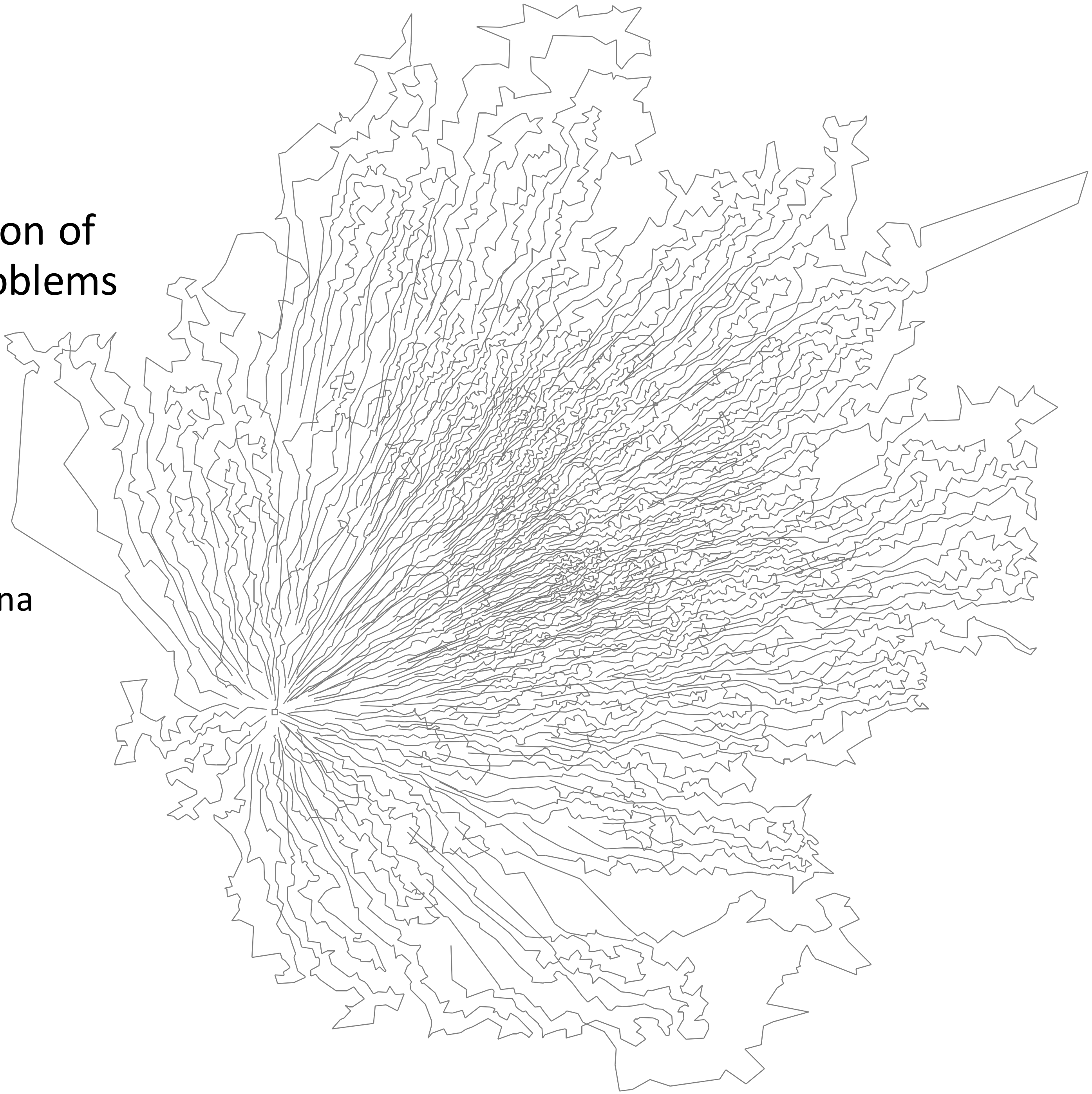
FILO

A Fast and Scalable Heuristic for the Solution of Large-Scale Capacitated Vehicle Routing Problems

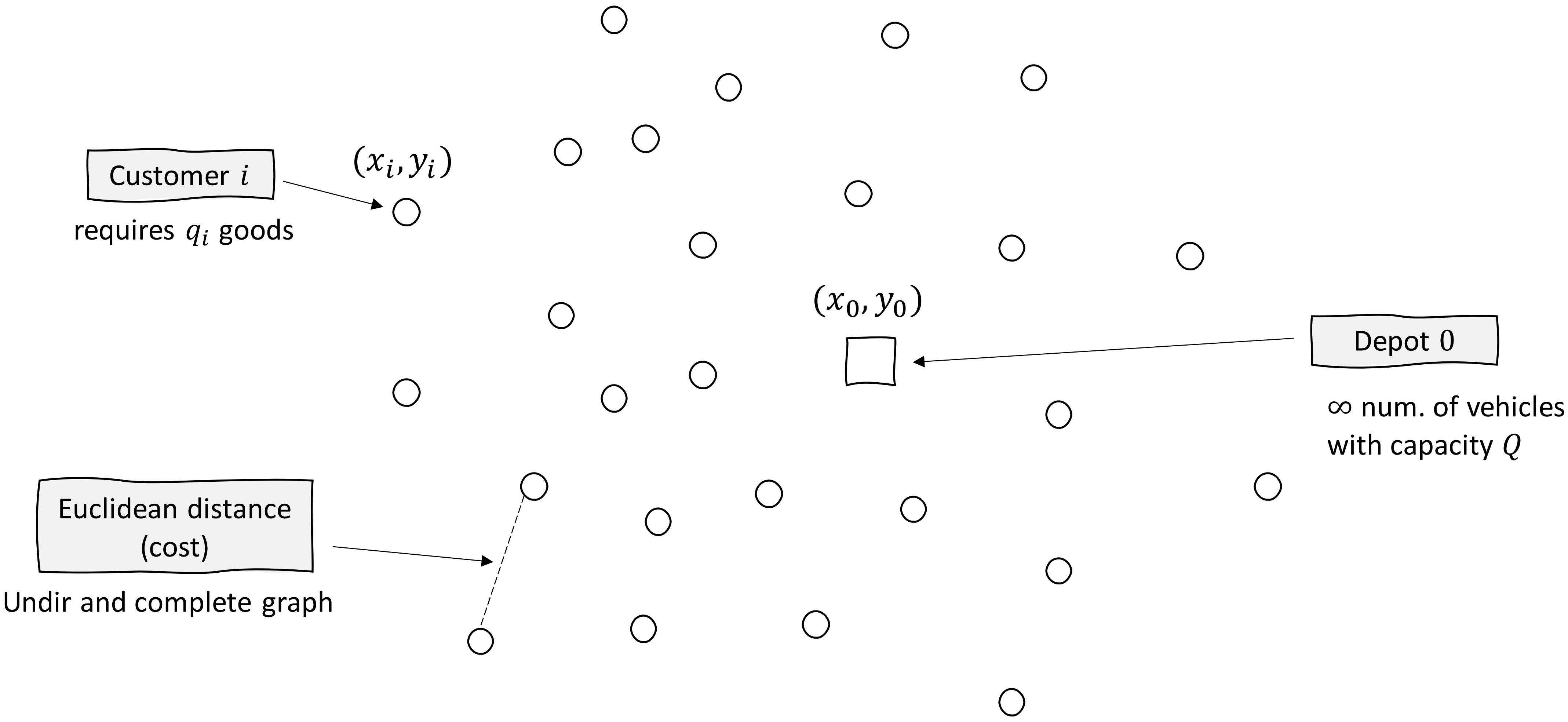


Luca Accorsi[1] and Daniele Vigo[1,2]

[1] DEI «Guglielmo Marconi», University of Bologna
[2] CIRI ICT, University of Bologna

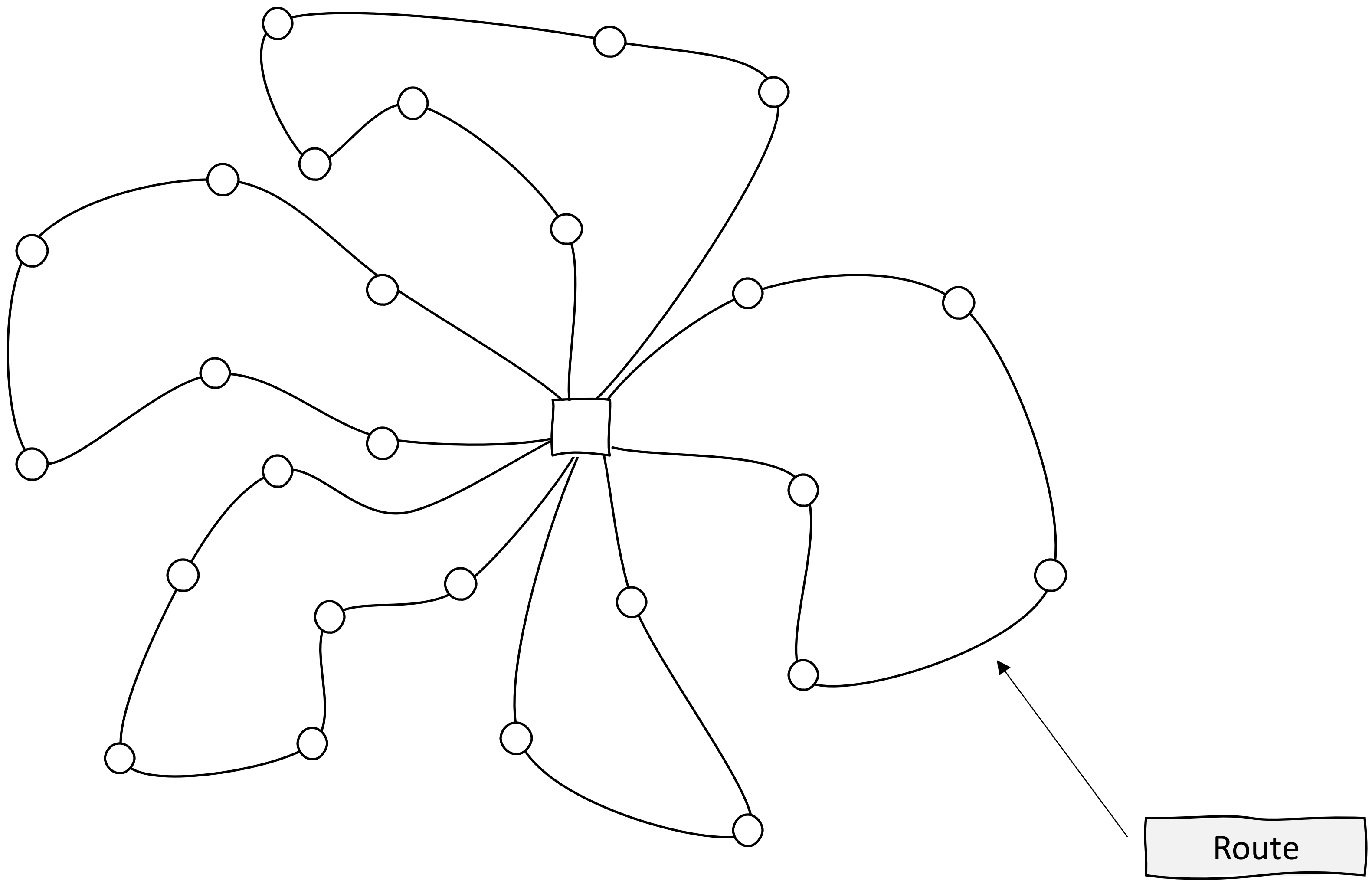
CAPACITATED VEHICLE ROUTING PROBLEM (CVRP)

INSTANCE

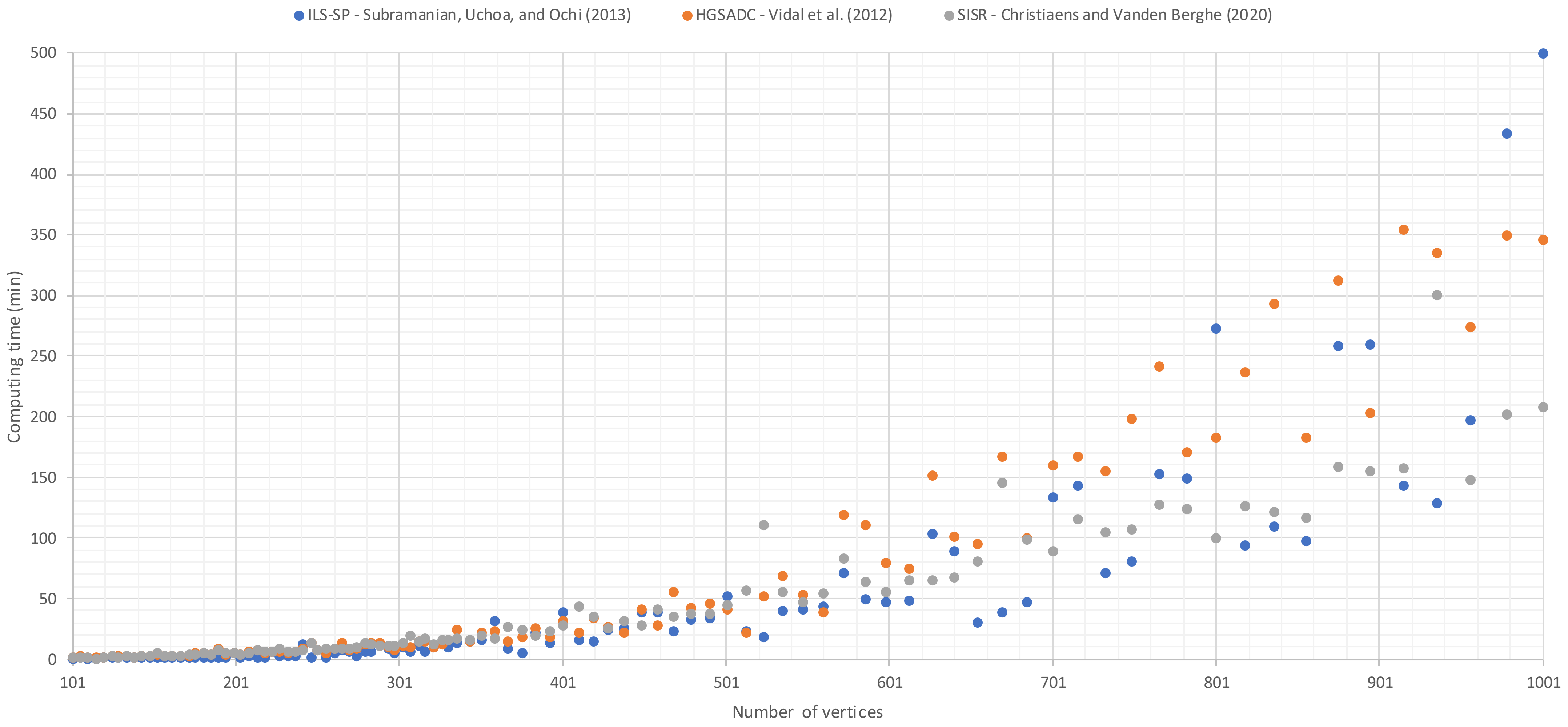


Capacitated Vehicle Routing Problem (CVRP)

SOLUTION

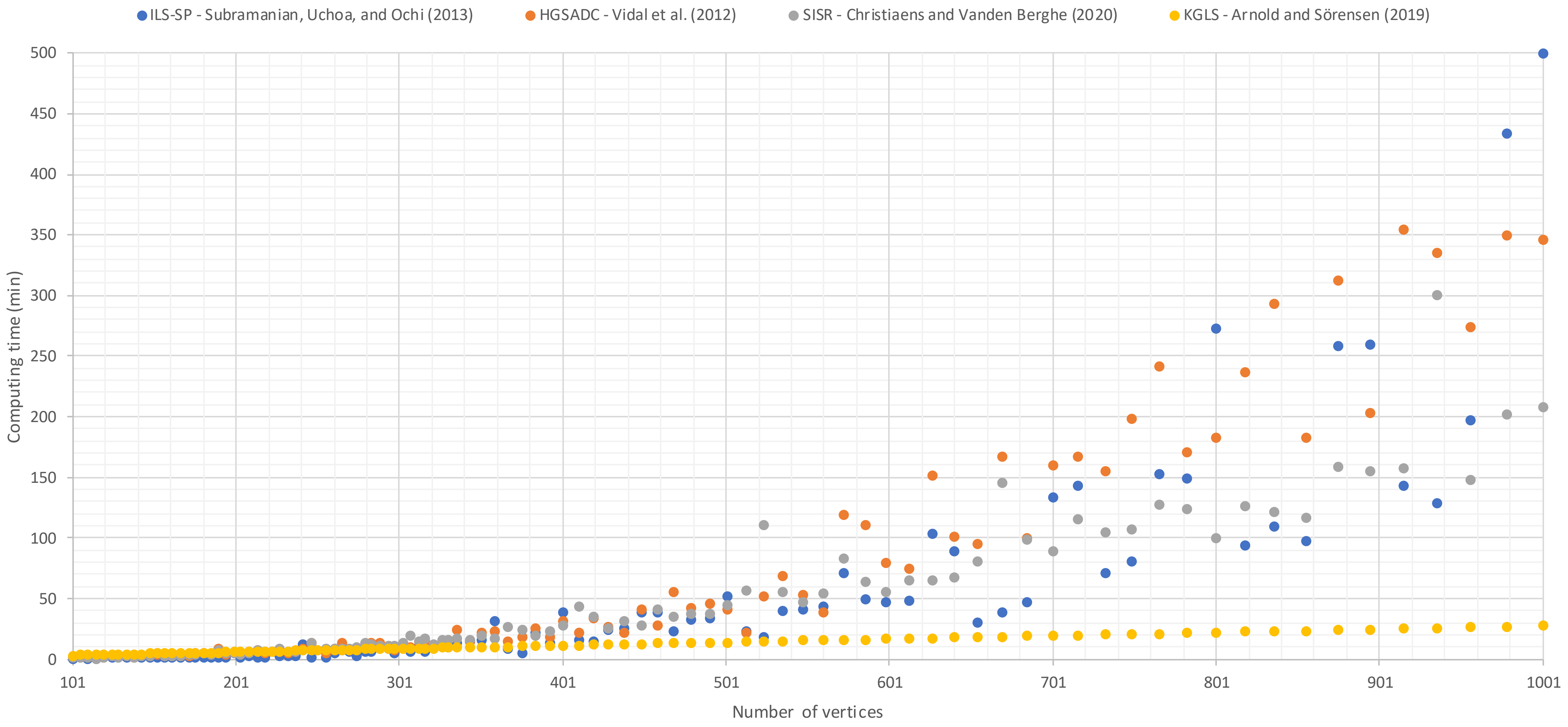


Motivation



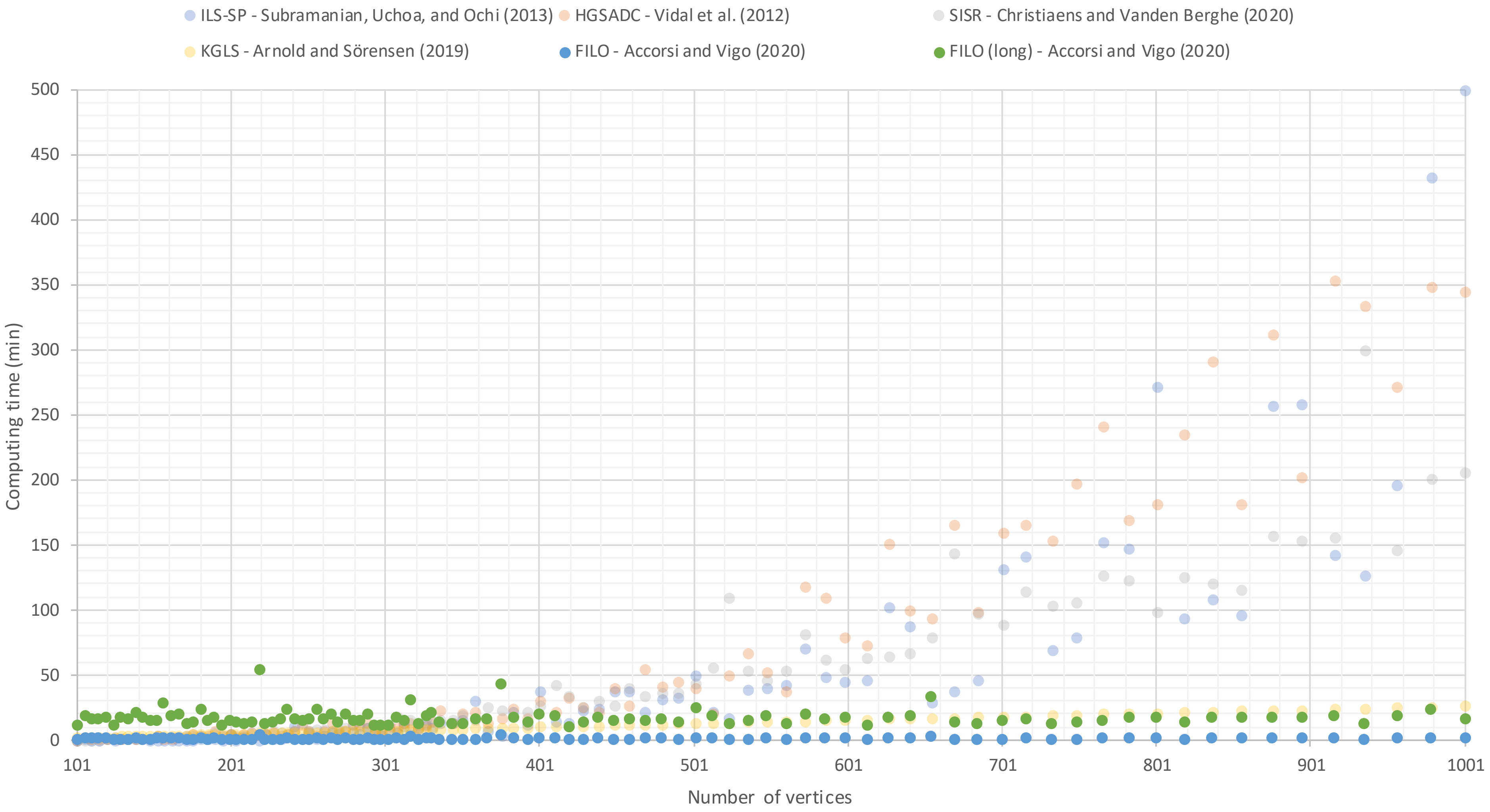
State-of-the-art (heuristic) CVRP algorithms often exhibit a **quadratic** growth

MOTIVATION



Others achieve a **linear** growth by fixing a maximum computing time

GOAL

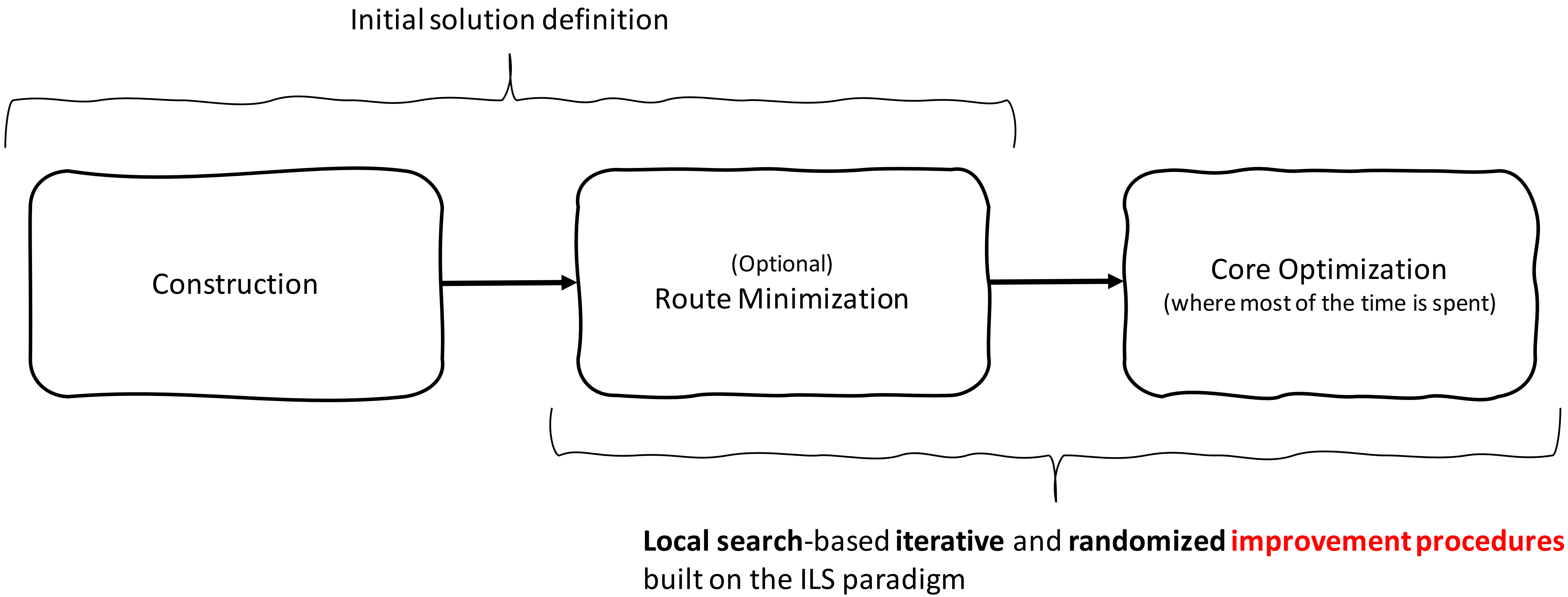


Designing a **fast**, naturally **scalable** and **effective** heuristic approach

OUR RECIPE

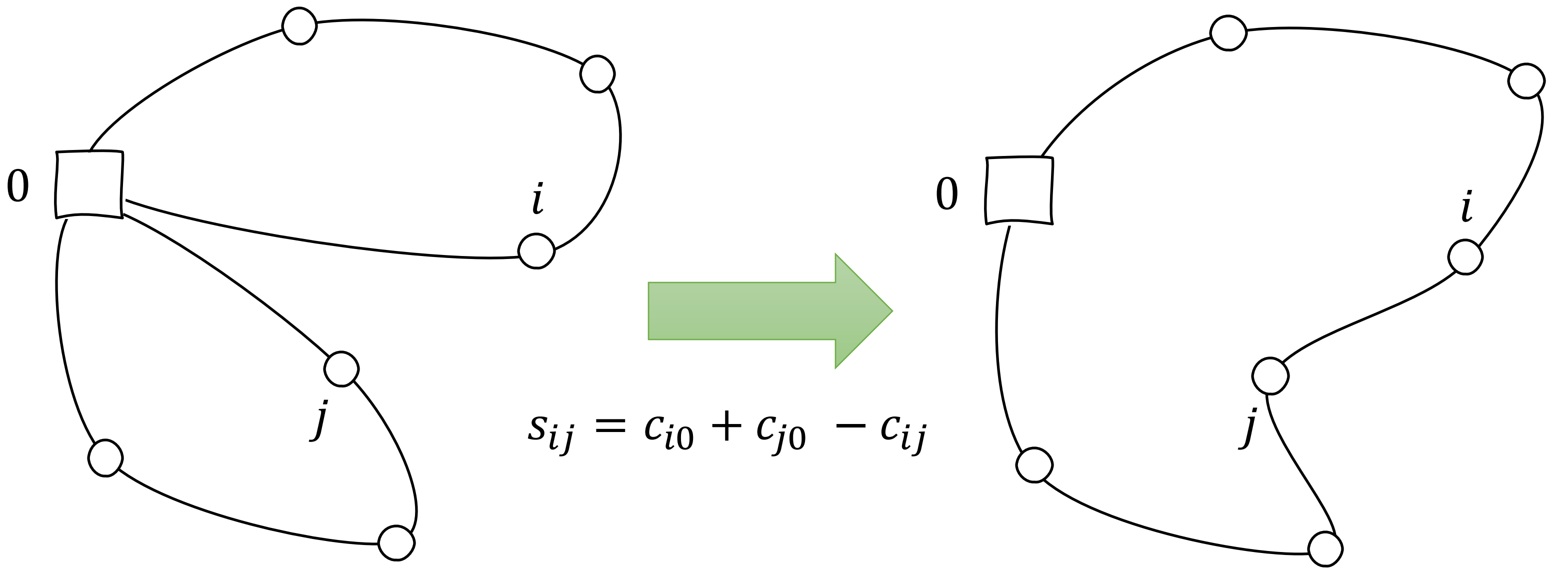
- Local Search **Acceleration** Techniques
  - Static Move Descriptors
- **Pruning** Techniques
  - Granular Neighborhoods and Selective Vertex Caching
- Careful **Design**
- Careful **Implementation**
- Careful **Parameters Tuning**

Fast ILS Localized Optimization (FILO)



CONSTRUCTION

A variation of the **Savings algorithm** by Clarke and Wright (1964)



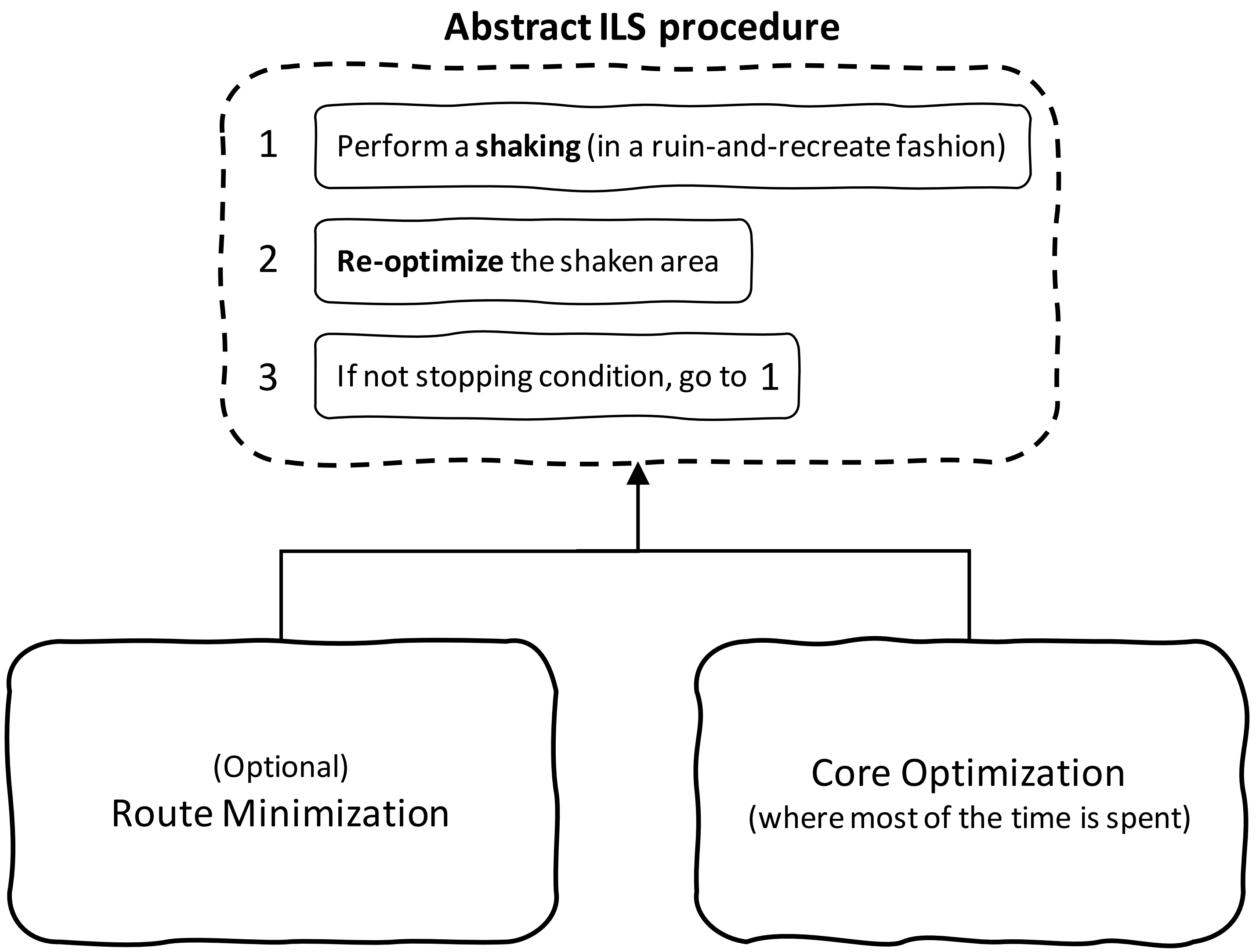
$$s_{ij} = c_{i0} + c_{j0} - c_{ij}$$

Adaptation proposed by Arnold, Gendrau, and Sörensen (2019)

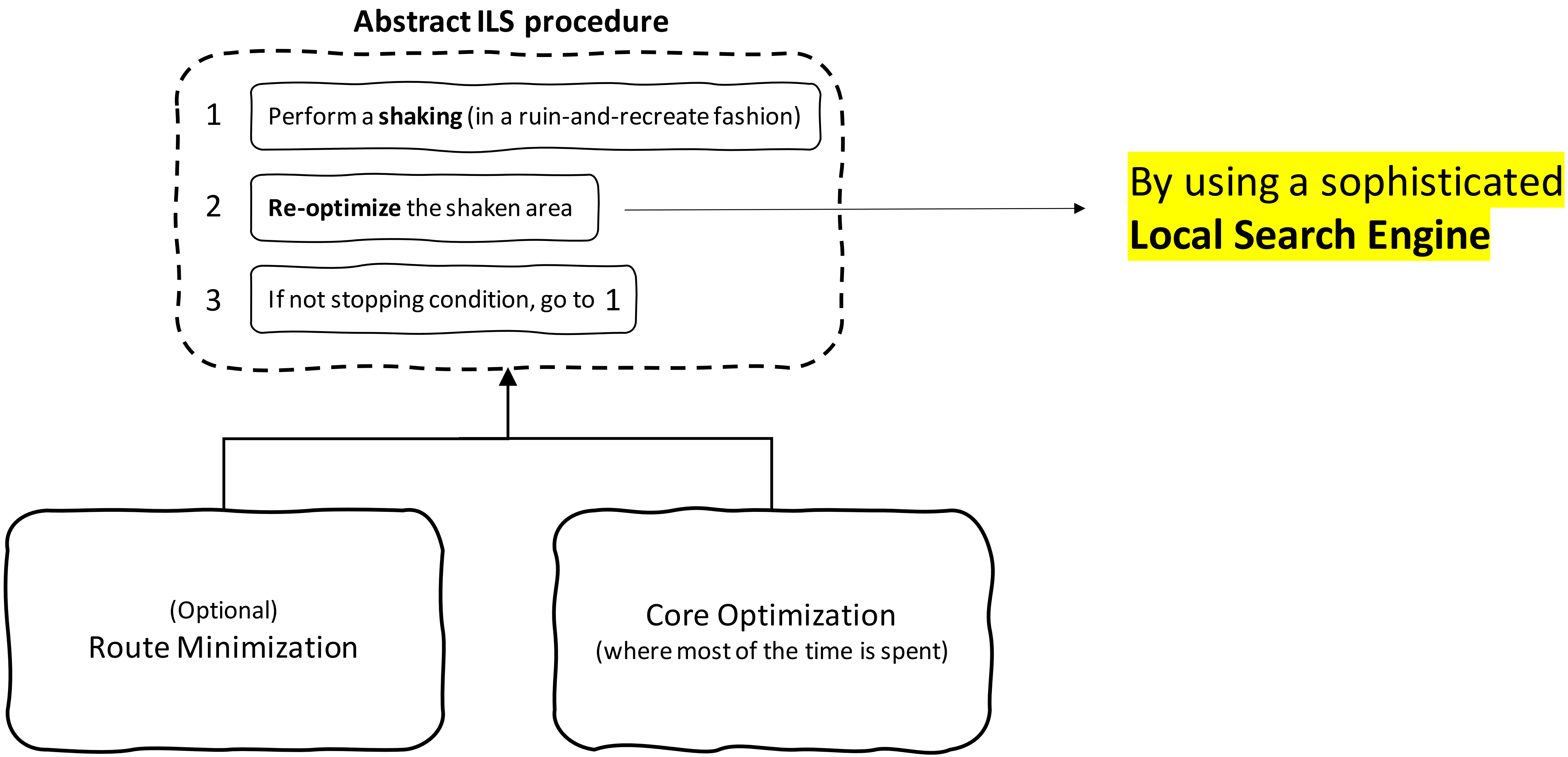
- For each i, compute s_{ij} only for $j \in Neighbors(i, 100)$ and $i < j$

$$O(n^2) \rightarrow O(n)$$

IMPROVEMENT PROCEDURES



IMPROVEMENT PROCEDURES

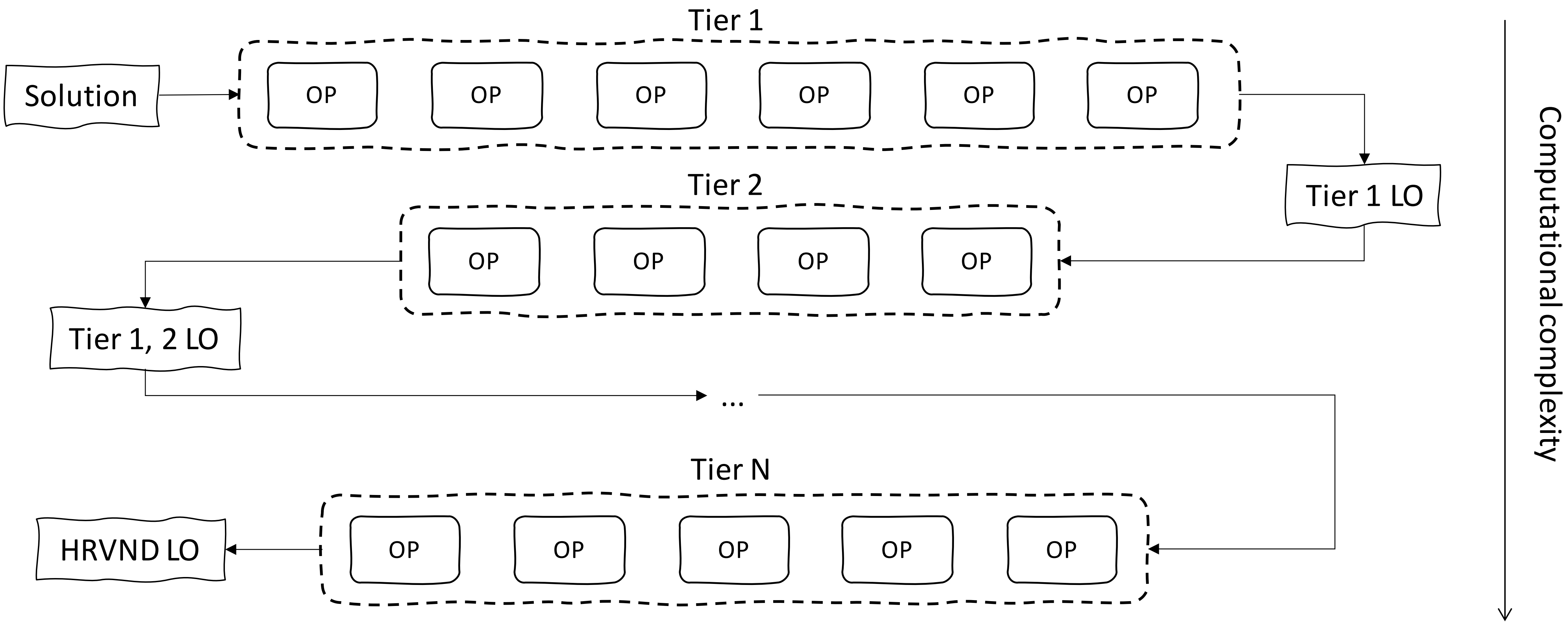


LOCAL SEARCH ENGINE

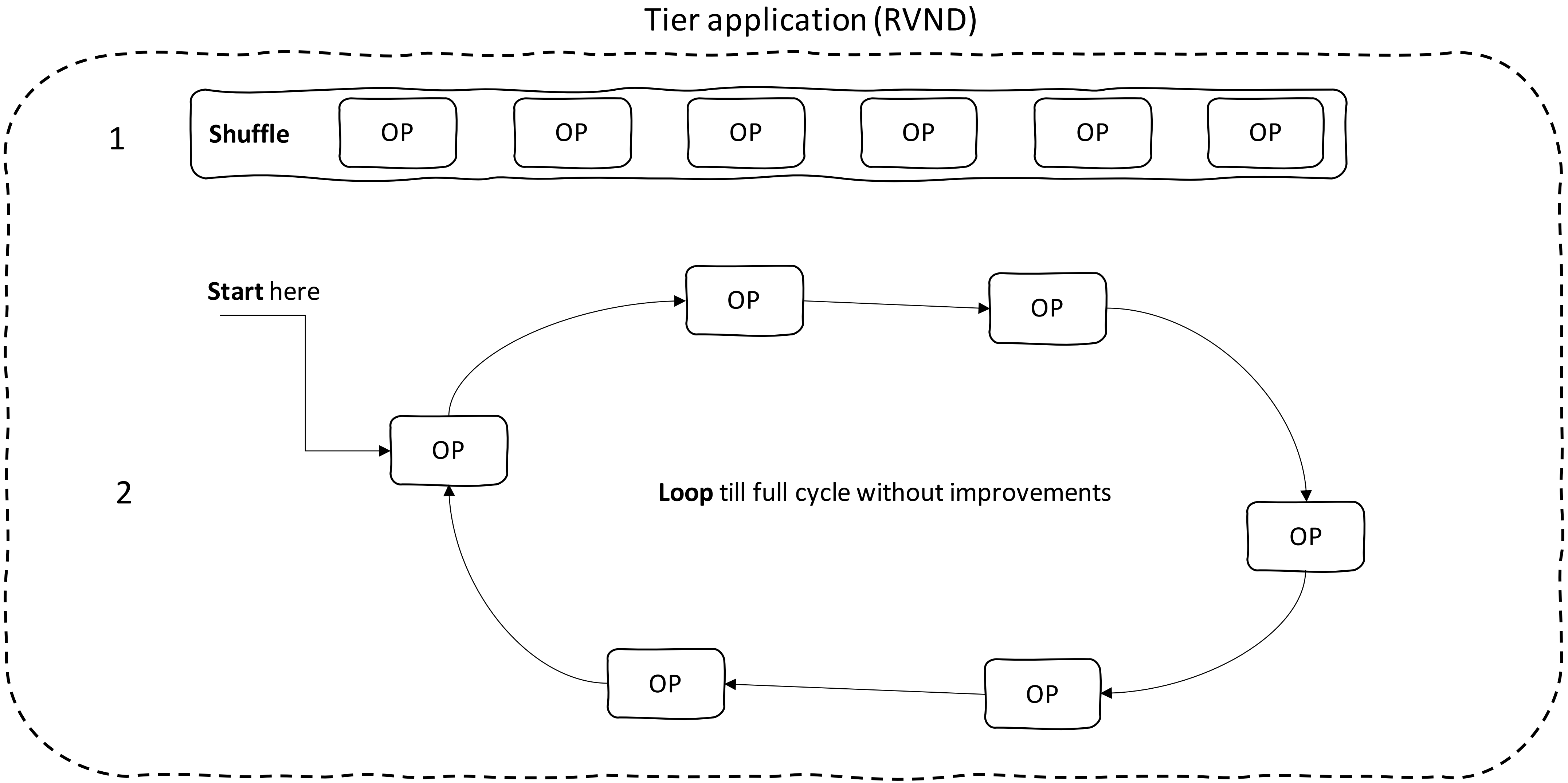
- Several operators explored in a VND fashion
  - **Hierarchical Randomized Variable Neighborhood Descent**
- Acceleration techniques for neighborhood exploration
  - **Static Move Descriptors**
- Pruning techniques
  - **Granular Neighborhoods** and **Selective Vertex Caching**

Hierarchical Randomized Variable Neighborhood Descent (HRVND)

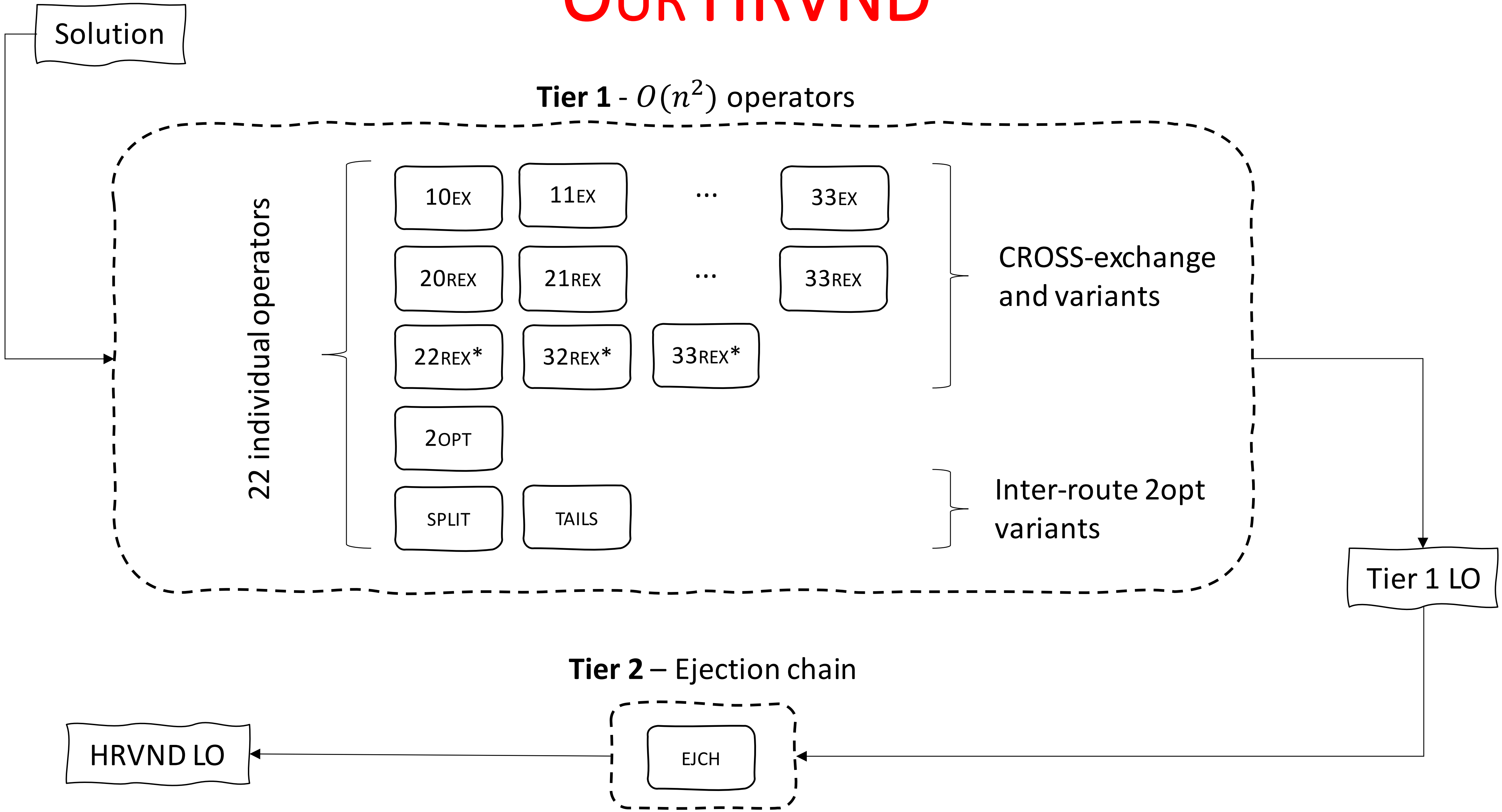
An effective organization of several local search operators



HRVND


Tier application (RVND)
1
Shuffle
OP
OP
OP
OP
OP
OP
Start here
OP
OP
OP
2
Loop till full cycle without improvements
OP
OP
OP
OP

Our HRVND


Solution
Tier 1 - $O(n^2)$ operators
22 individual operators
10EX
11EX
...
33EX
20REX
21REX
...
33REX
22REX*
32REX*
33REX*
2OPT
SPLIT
TAILS
CROSS-exchange and variants
Inter-route 2opt variants
Tier 1 LO
Tier 2 – Ejection chain
EJCH
HRVND LO

HRVND MOTIVATION

Combining the good parts of VND and RVND

- From **RVND**
  - do not fix a possibly not ideal neighborhood exploration order within tiers

- From **VND**
  - more complex operators are executed after simpler ones in subsequent tiers
    - to further polish solutions and escape from local optima

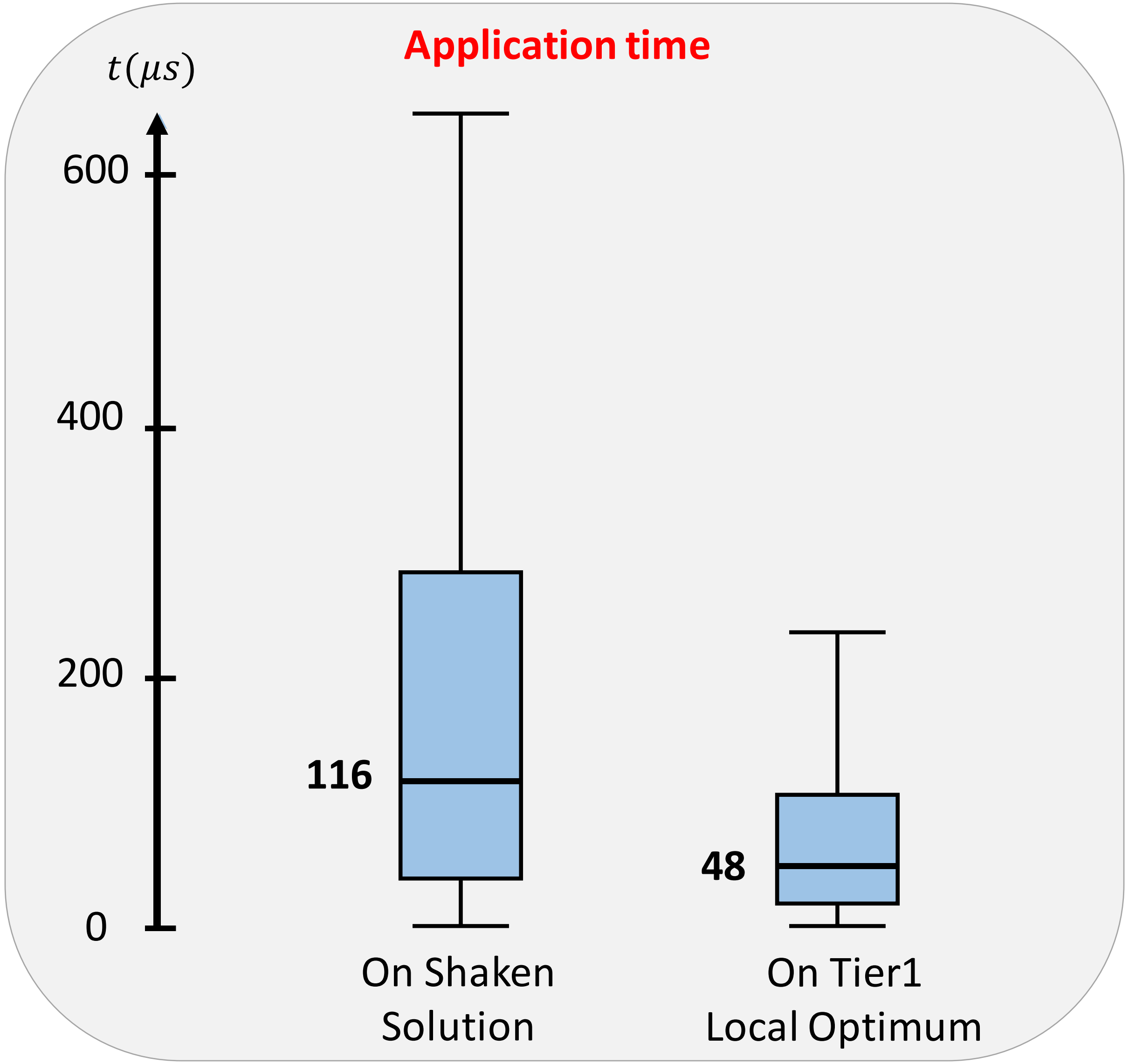
Complex operators expected application time (as well as their improvement) is reduced because they are applied on already high-quality solutions

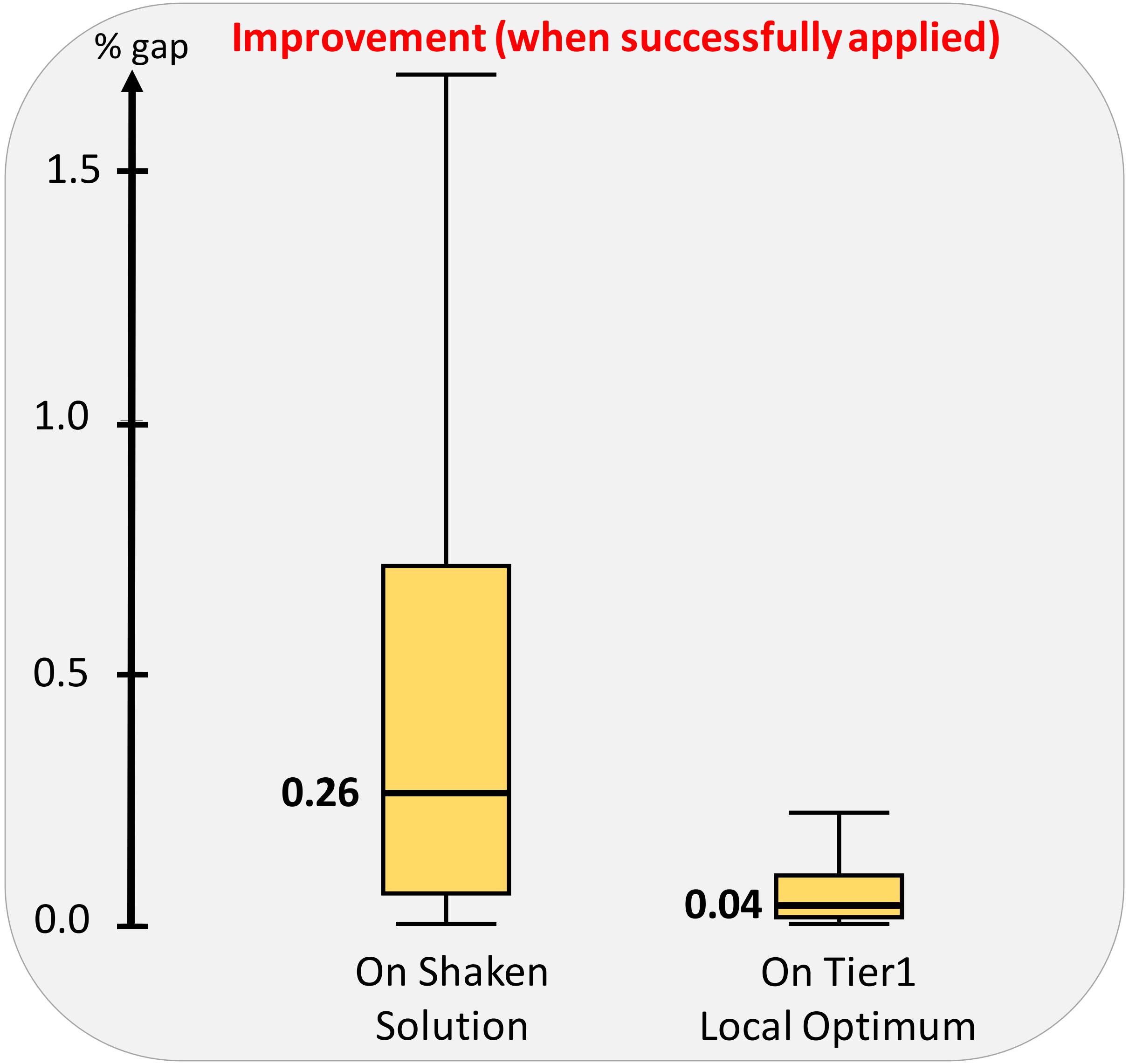
HRVND MOTIVATION

EJECTION CHAIN

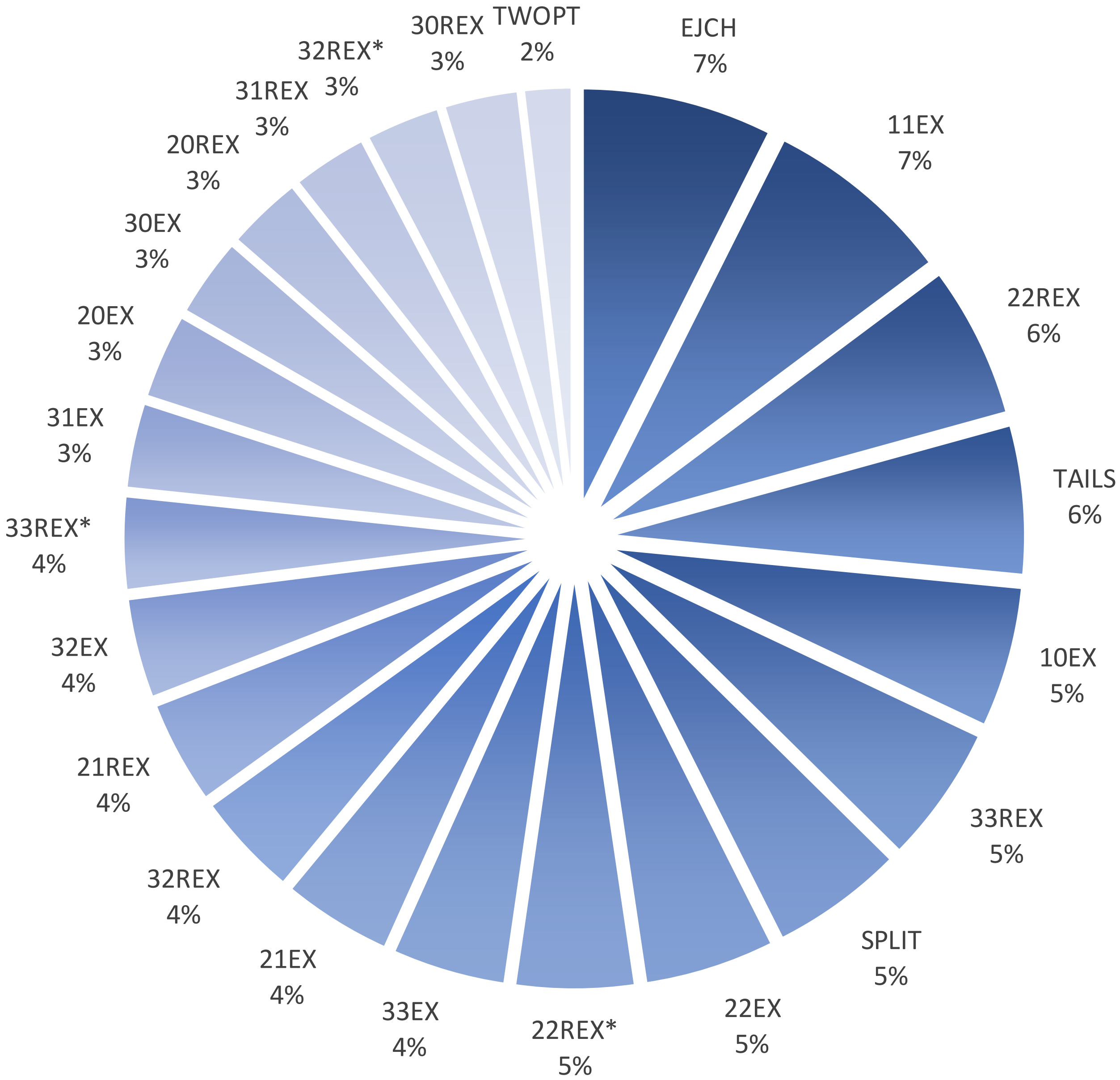
Success ratio

On Shaken Solution	**78.71 %**
On Tier1 LO	**30.70 %**





HRVND MOTIVATION



$$O = \text{ set of available LS operators}$$

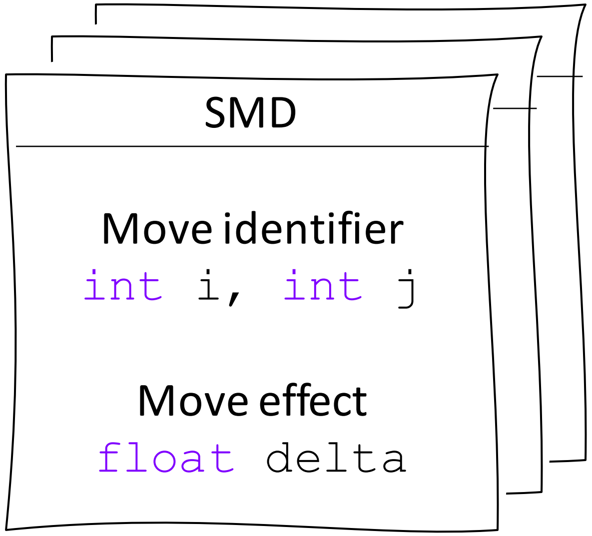
$$RNI(o, O) = 100 \frac{R(o)}{\sum_{o' \in O} R(o')}$$

$$R(o) = \frac{\text{tot improvement of } o}{\text{successful application of } o}$$

STATIC MOVE DESCRIPTORS (SMDs)

A data-oriented approach to local search

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    eval/apply(i, j)
  }
}
```

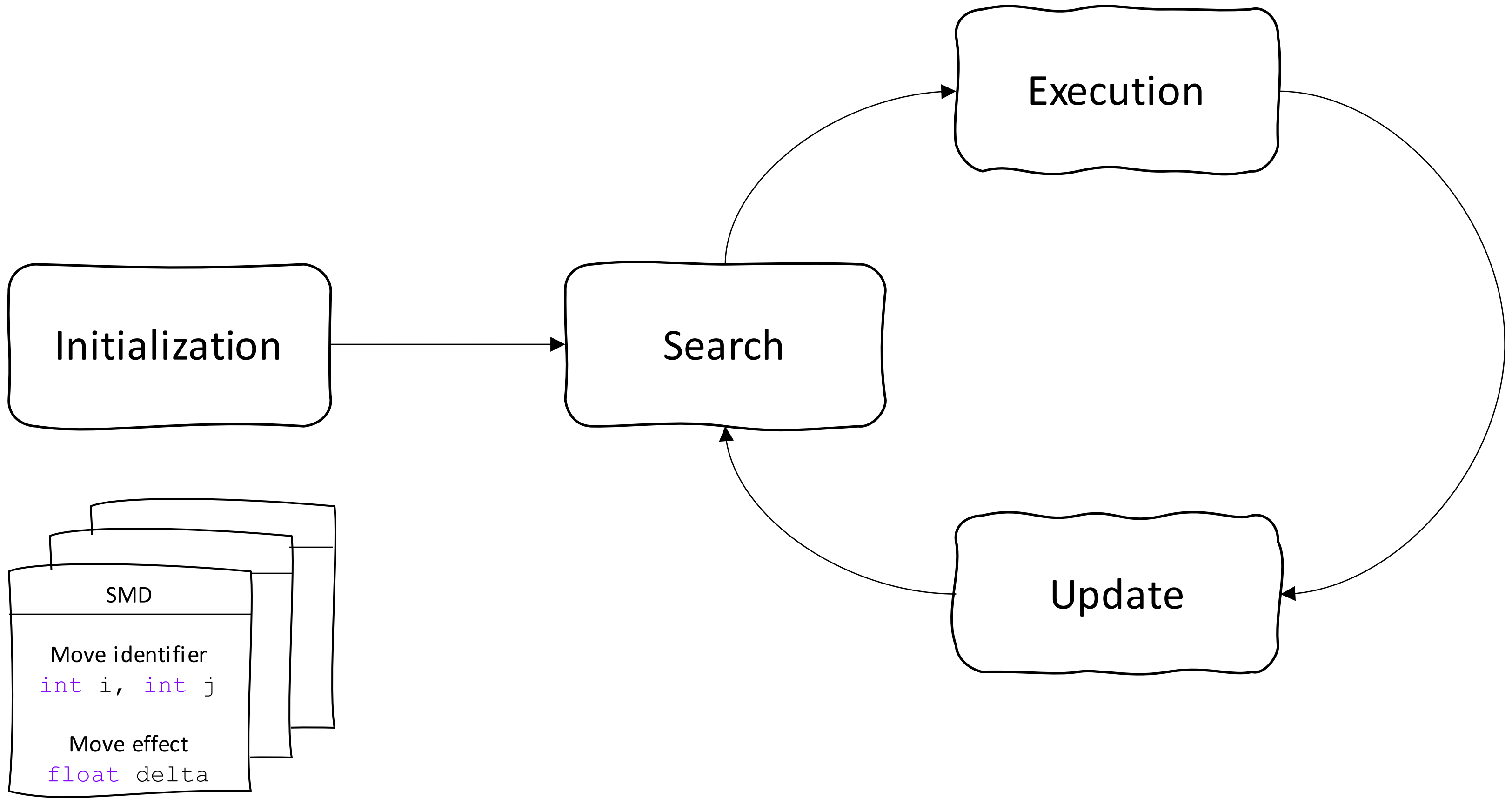


BIBLIOGRAPHY FOR SMDs

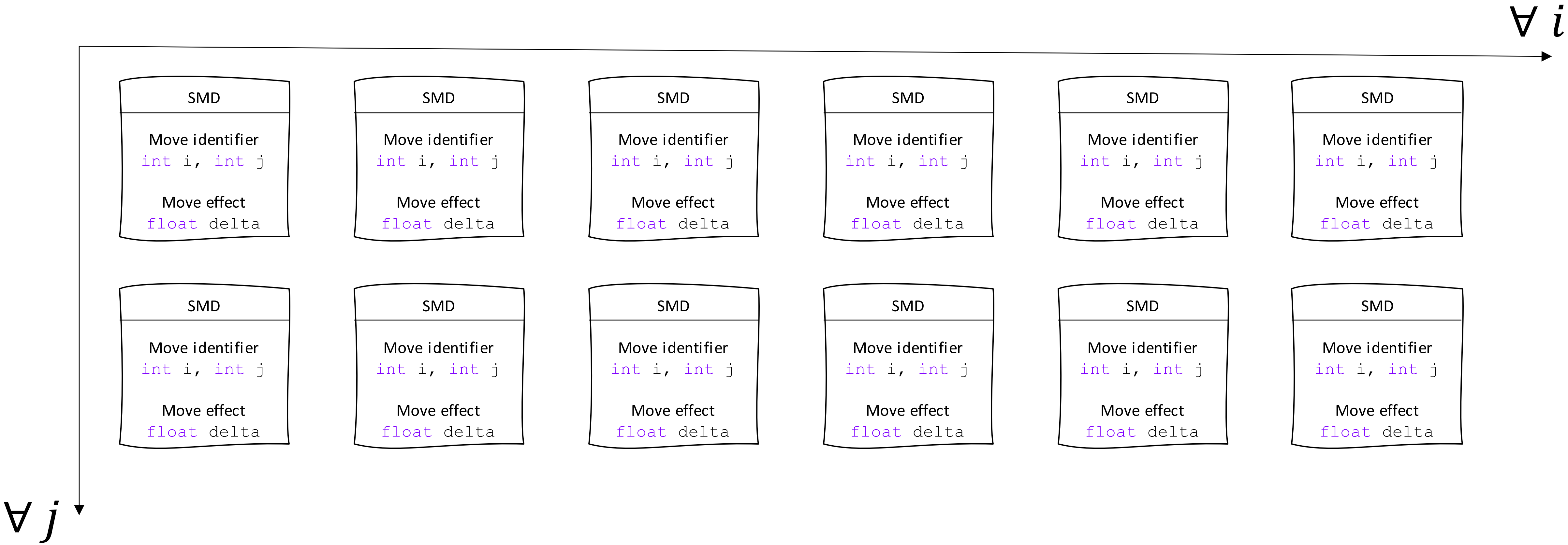
- Emmanouil E. Zachariadis, Chris T. Kiranoudis, A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem, Computers & Operations Research, Volume 37, Issue 12, 2010, Pages 2089-2105
- Onne Beek, Birger Raa, Wout Dullaert, Daniele Vigo, An Efficient Implementation of a Static Move Descriptor-based Local Search Heuristic, Computers & Operations Research, Volume 94, 2018, Pages 1-10

SMD PROCEDURES

Replace the "for-loop" neighborhood exploration with a more structured inspection of moves

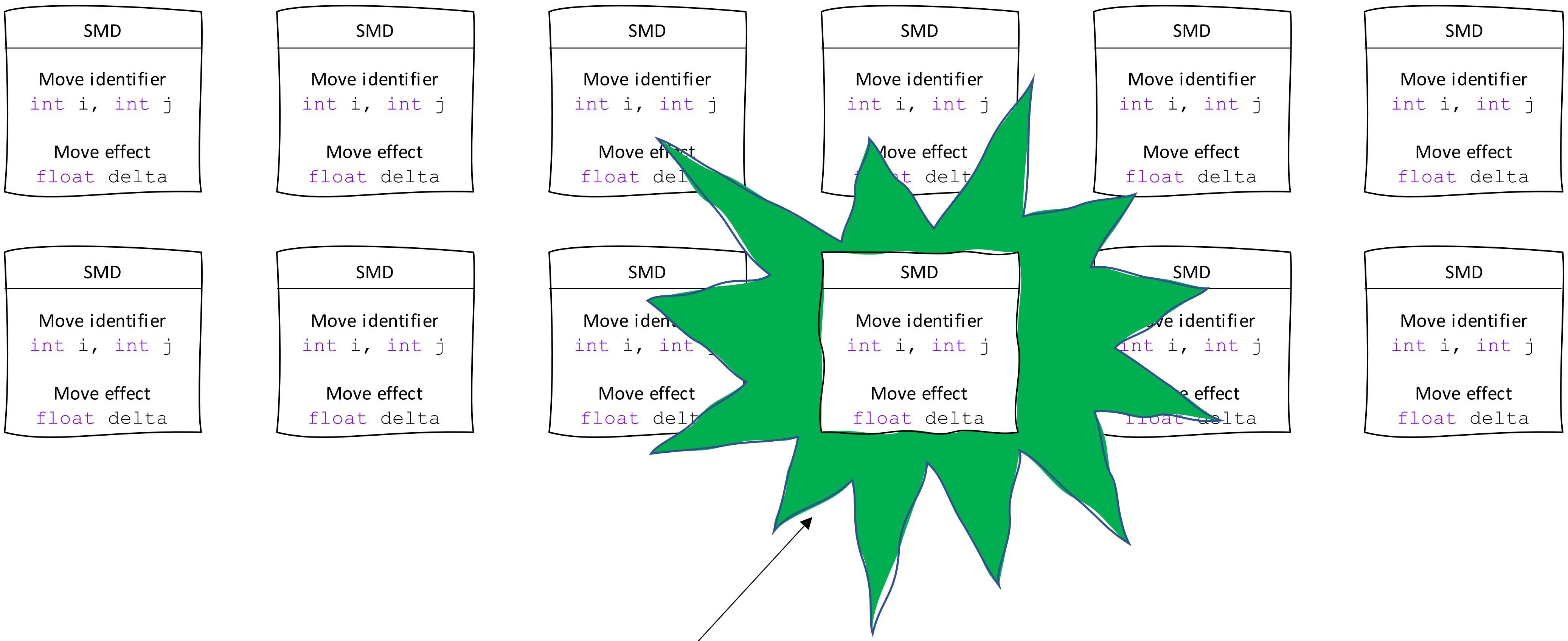


SMD Initialization



$$O(single\ loop - based\ exploration)$$

SMD Search



Feasible and **best** (e.g. most improving) SMD

SMD SEARCH

Zachariadis and Kiranoudis (2010) suggest to store SMDs into a **heap**

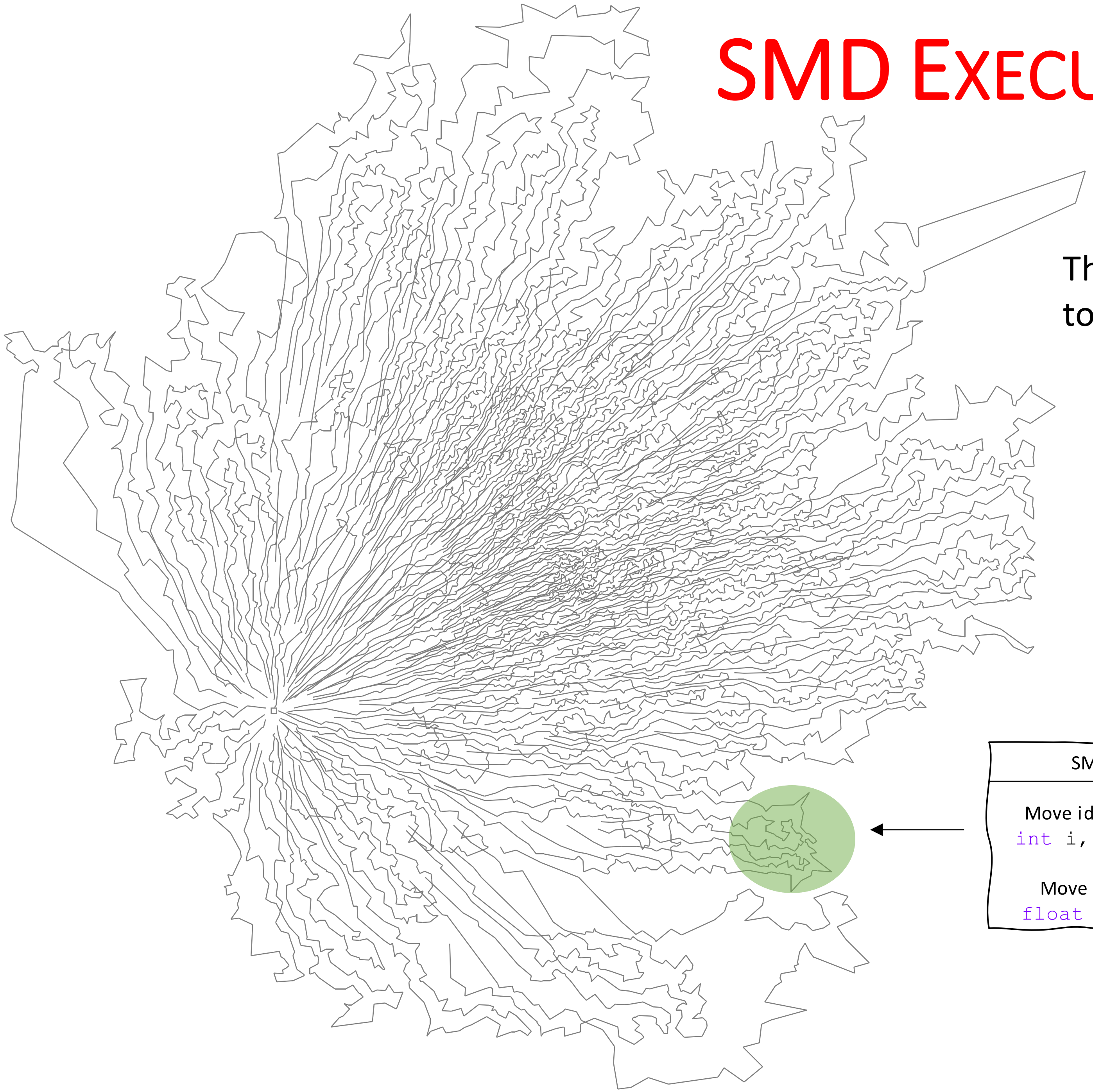
- Retrieve in $O(1)$, remove and restore heap property in $O(\log n)$
- If not feasible, store and reinsert later $O(\log n)$

OUR CHOICE

Beek et al. (2018) suggest to linearly scan the heap to avoid removal and reinsertion for each SMD not feasible

- No more guarantees of retrieving the best SMD
- The heap entries are roughly sorted

SMD Execution

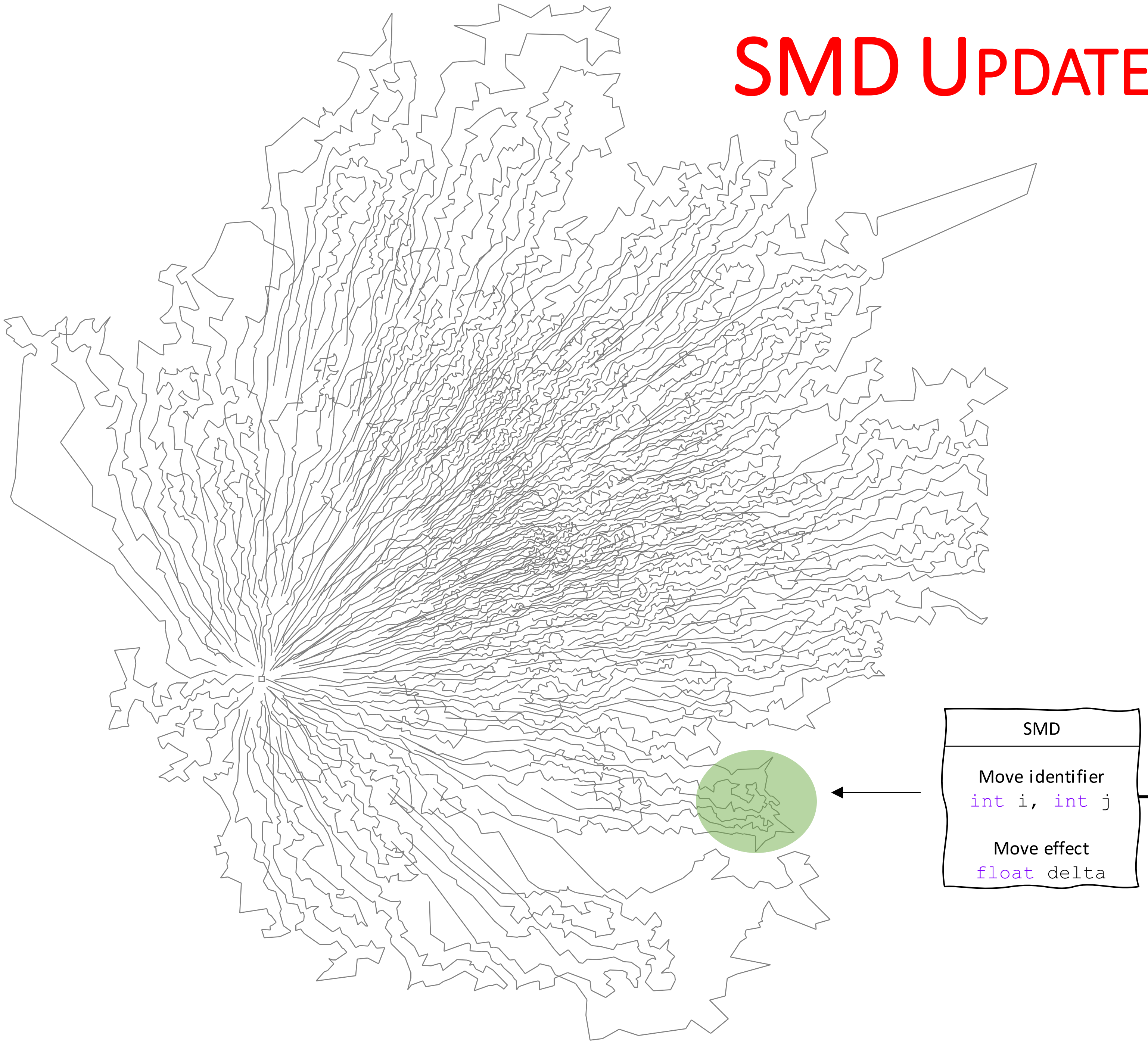


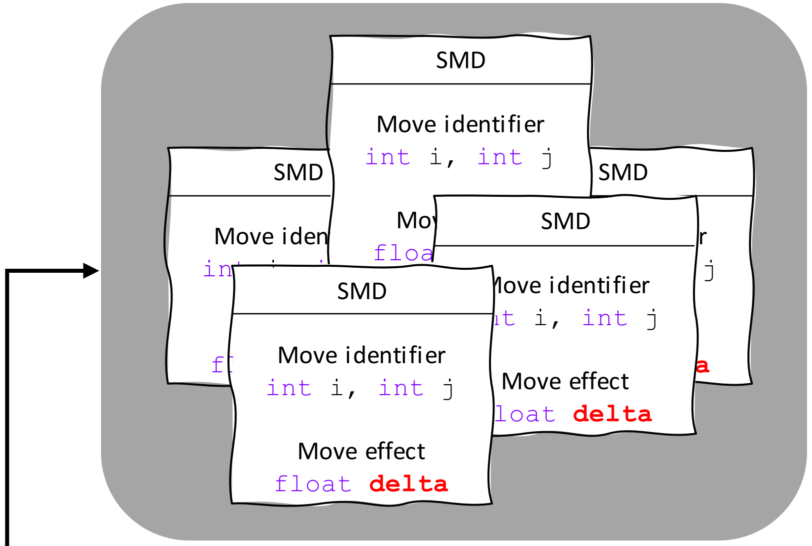
The move associated with the selected SMD is applied to the current solution

Local search operators perform local changes thus **most** of the SMDs will still hold a correct `delta` value

SMD
Move identifier `int i, int j`
Move effect `float delta`

SMD UPDATE

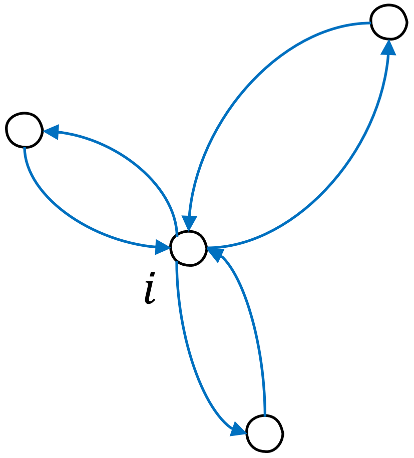




A move `(i,j)` of operator `XYZ` requires the update of the `delta` value of fixed set of SMDs

Granular Neighborhoods (GNs)

Restricting local search move evaluations to promising ones only



Sparsification rule

For each vertex i consider only the moves (SMDs) generated by arcs (i, j) and (j, i) such that j ∈ $\mathrm{Neighbors}(i, 25)$

$$T = \cup_i \ \{(i,j),(j,i) : j \in Neighbors(i, 25)\}$$

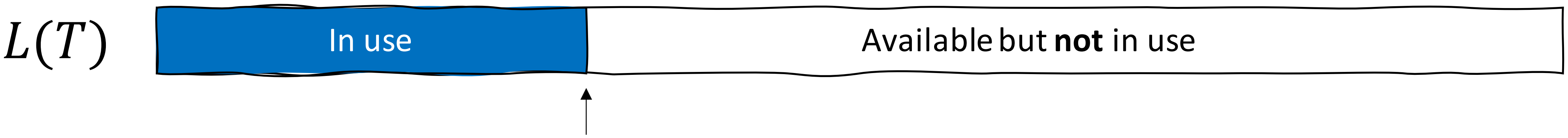
Set of **move generators**

BIBLIOGRAPHY FOR GNs

- Paolo Toth and Daniele Vigo, The Granular Tabu Search and Its Application to the Vehicle-Routing Problem, INFORMS Journal on Computing 2003 15:4, 333-346
- Michael Schneider, Fabian Schwahn, Daniele Vigo, Designing granular solution methods for routing problems with time windows, European Journal of Operational Research, Volume 263, Issue 2, 2017, Pages 493-509

Dynamic GNs

Ordered list of **move generators**

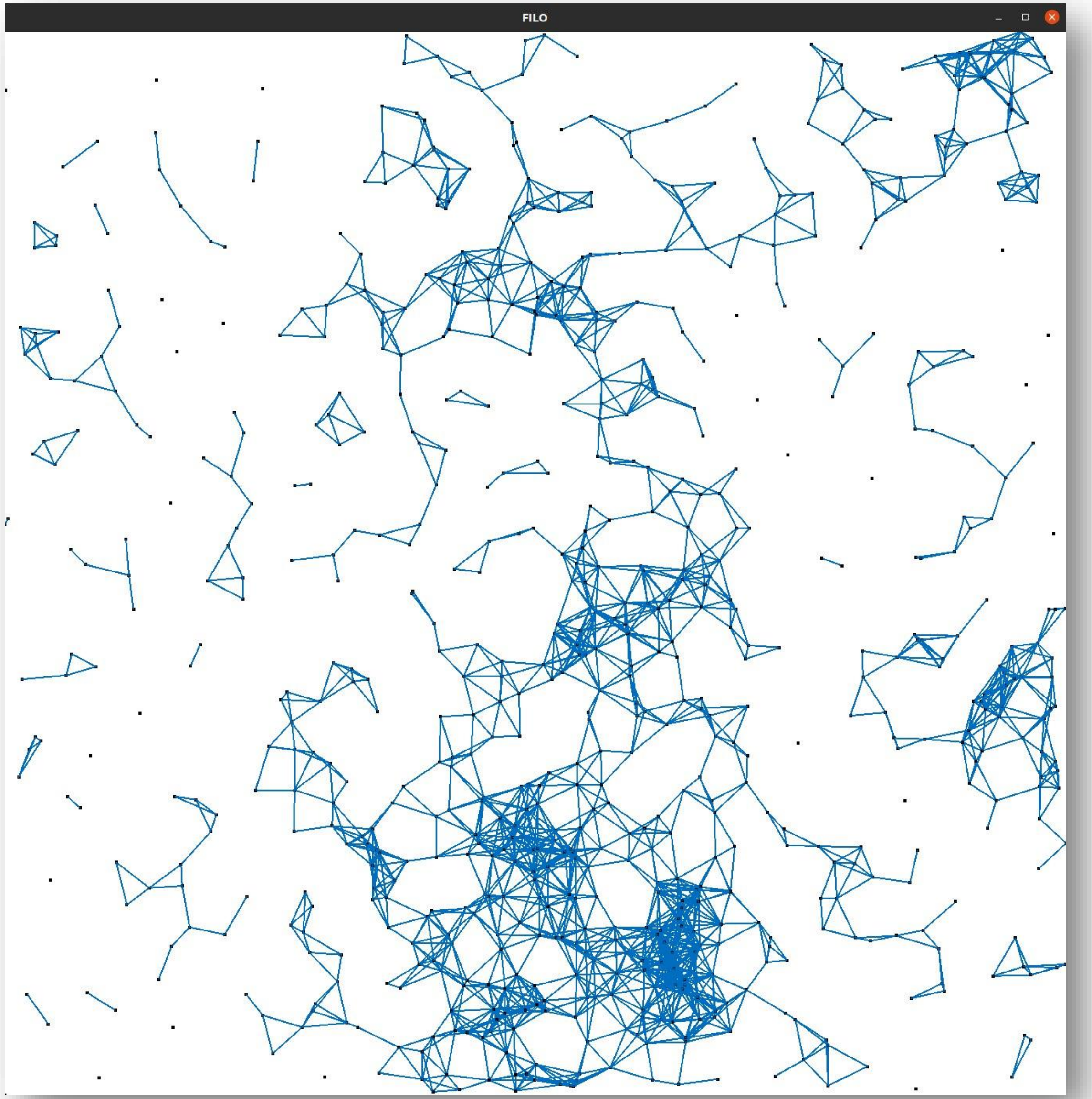


γ **sparsification factor**
(percentage $\gamma \in [0,1]$ or cost threshold $\gamma \in \mathbb{R}$)

Update rule
$$\begin{cases} \text{set } \gamma = \min\{2\gamma, 1\} & \text{if several non improving iterations} \\ \text{set } \gamma = \gamma_{base} & \text{if new BKS is found} \end{cases}$$

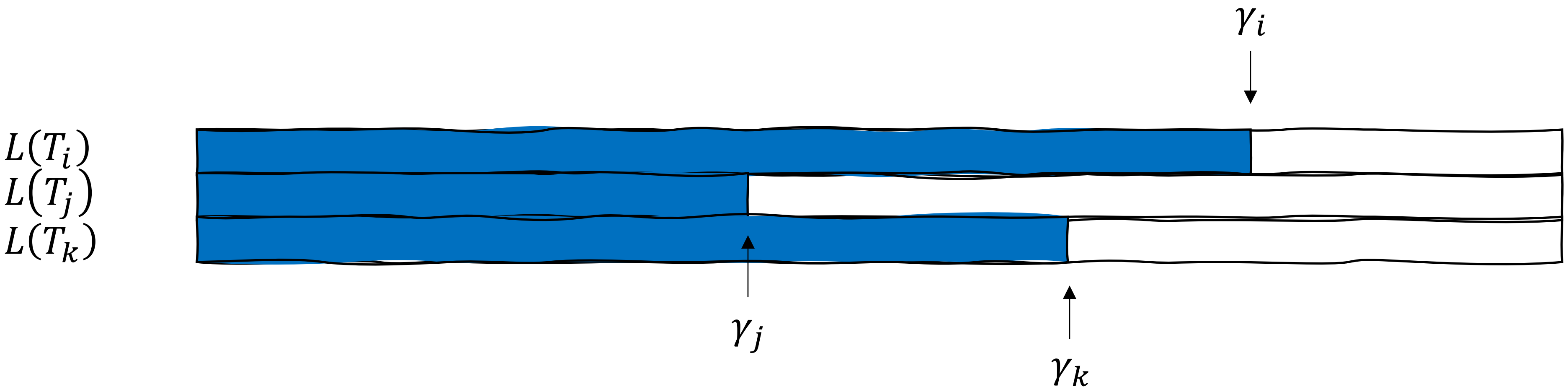
Dynamic GNs

May not capture scenarios with different densities of customers (when γ is low)



VERTEX-WISE DYNAMIC GNs

Let each vertex manage its own move generators



γ_i **sparsification factor**
(percentage $\gamma_i \in [0, 1]$ for each vertex i)

Update rule

- set $\gamma_i = \min\{2\gamma_i, 1\}$ if several non improving iterations involving i
- set $\gamma_i = \gamma_{base}$ if new BKS is found by optimizing a solution area containing i

VERTEX-WISE DYNAMIC GNS

PRO

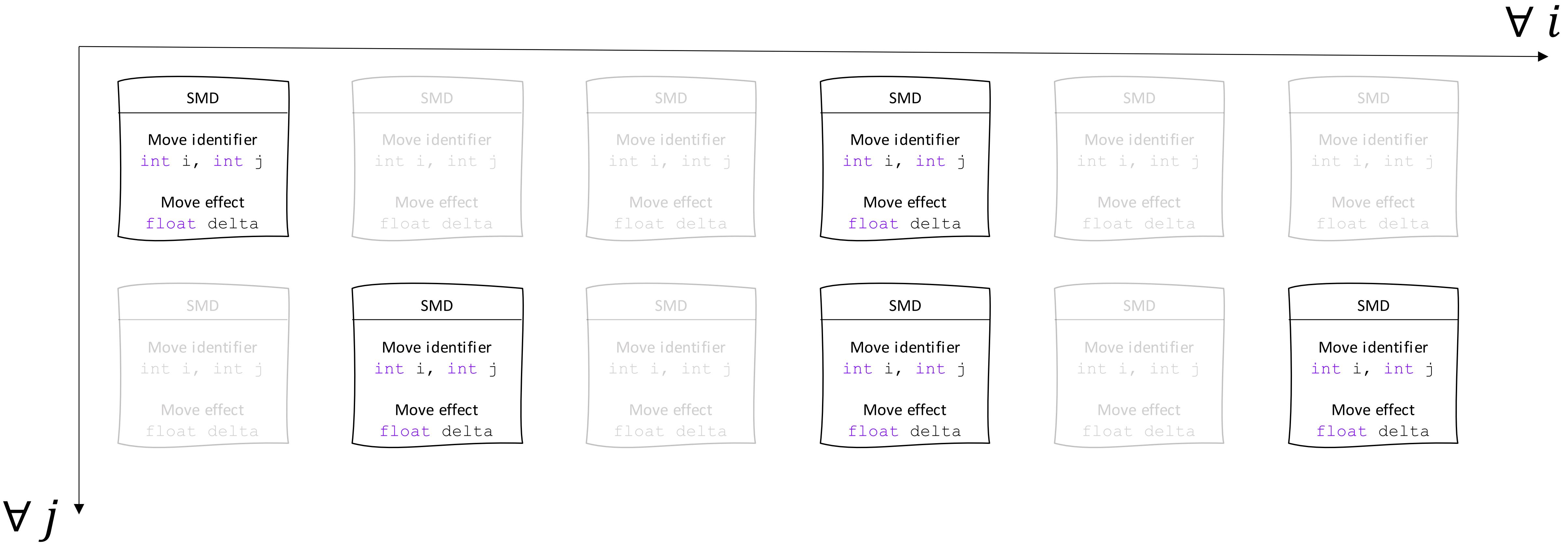
- A minimum number of move generators is guaranteed per vertex
- Tailored intensification: move generators are increased only for areas that more likely require a stronger intensification
- Intensification is globally increased at a slower rate
  - faster local search for more optimization iterations

CONS

- Management of a γ_i for each vertex i
- Intensification is globally increased at a slower rate:
  - more iterations are required for a globally stronger local search

Granular SMD Neighborhoods

Only consider **SMDs** associated with **active move generators**



SELECTIVE VERTEX CACHING (SVC)

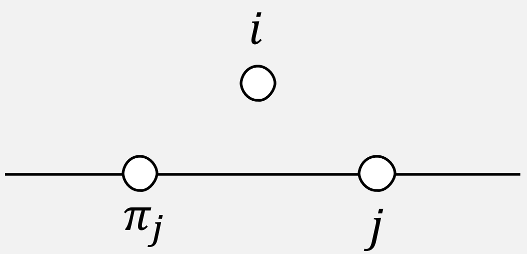
A granular neighborhoods counterpart for vertices

Keep track of a set of **interesting vertices** $\overline{V_S}$ associated with solution S

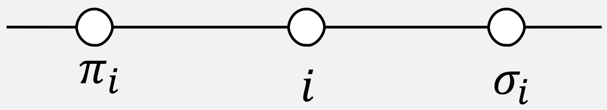
INTERESTING

Vertices belonging to solution areas that **recently** underwent some **change**

Insertion of i before j: π_j, j, i



Removal of i: π_i, i, σ_i



RECENTLY

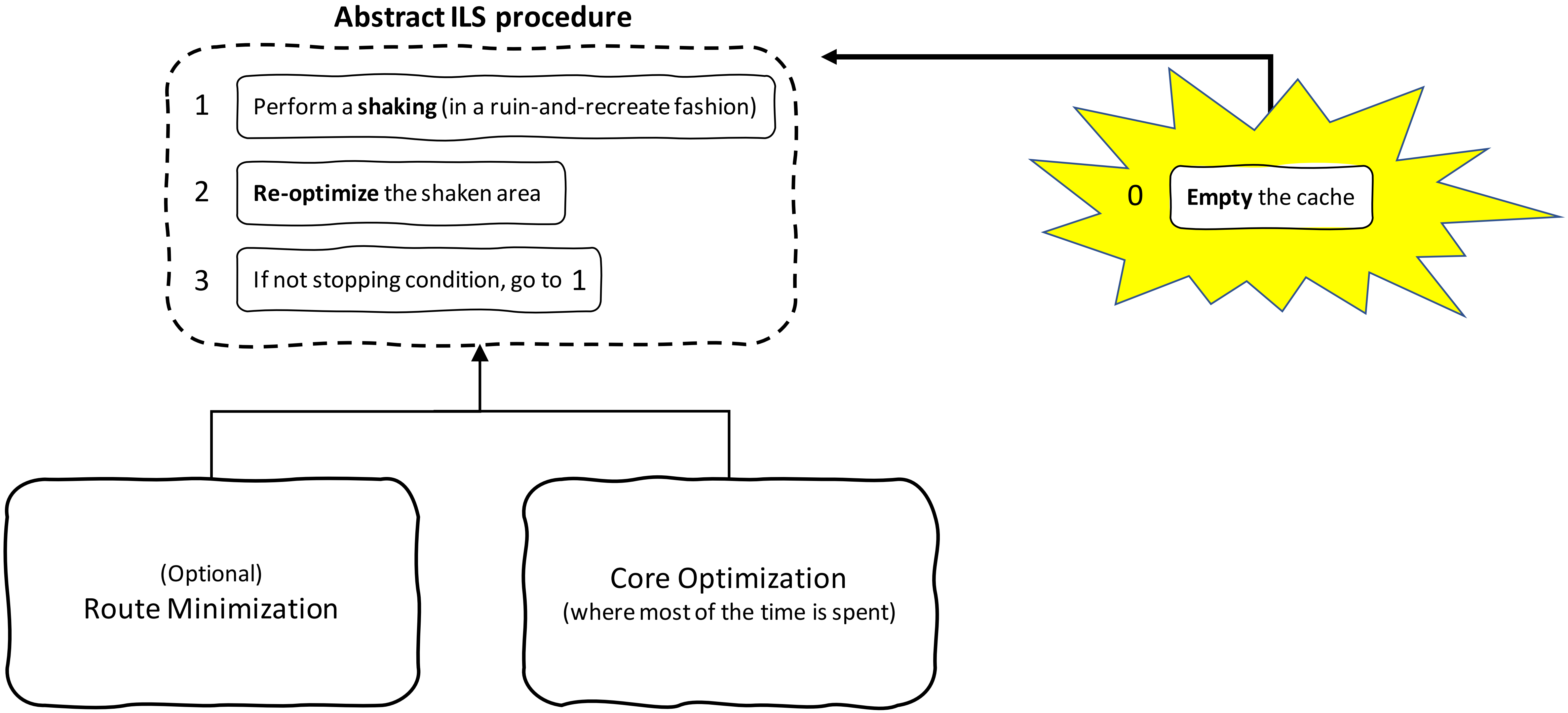
$|\overline{V_S}| < C$ constant + LRU update policy

SVC TO RESTRICTED SMD INITIALIZATION

Initialize only SMDs associated with active move generators such that at least one of the endpoints belongs to the cache $\overline{V_S}$

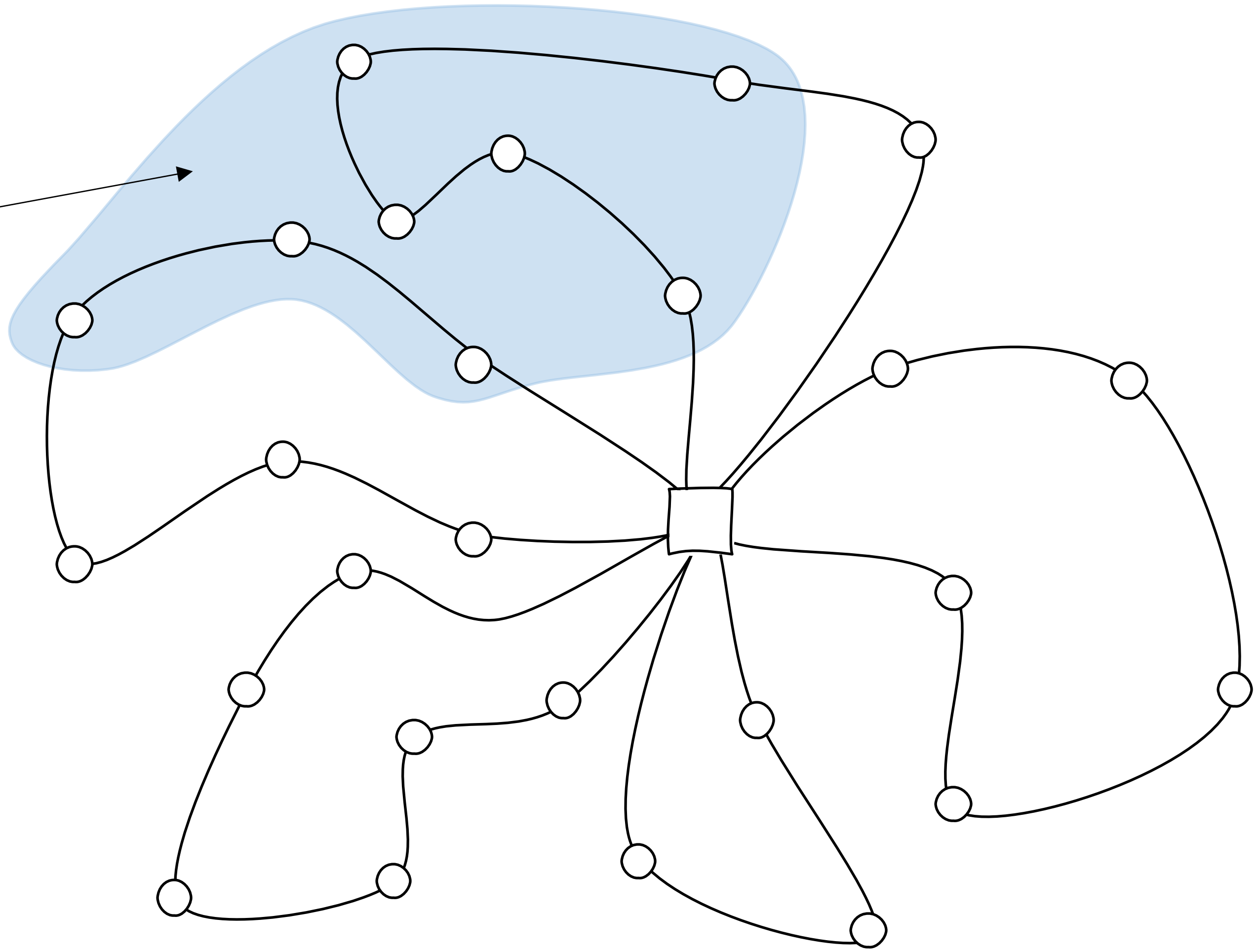
Subsequent SMD Updates may incrementally include additional SMDs

SVC to Focus Local Search Applications



SVC to Update Vertex-wise Move Generators

Cached vertices
after HRVND execution



Update rule

$$\begin{cases} \text{set } \gamma_i = \min\{2\gamma_i, 1\} & \text{if several non improving iterations involving } i \\ \text{set } \gamma_i = \gamma_{base} & \text{if new BKS is found by optimizing a solution area containing } i \end{cases}$$

Route Minimization

- Empirical correlation between number of routes and solution cost
- Optional polishing of the initial solution if it appears to be using **more routes than necessary**
    - Greedy estimate k from solution of Bin Packing Problem
- Contrarily to standard route minimization procedures, it is still a **quality-oriented** procedure

Route Minimization

Loop

1 Destroy a pair of close routes | Place the removed customers into a list L | Sort L

2 **For each** customer i in L

Find best insertion position of i in **existing routes**

- **If** a position is found → Insert i
- None of the routes can accomodate i
  - **If** $|S| < k$ **or** $U(0,1) > p$ → Create a route with i
  - Otherwise → Place i into a list $\bar{L}$

3 Set $L = \bar{L}$

4 Optimize the solution using the LS engine

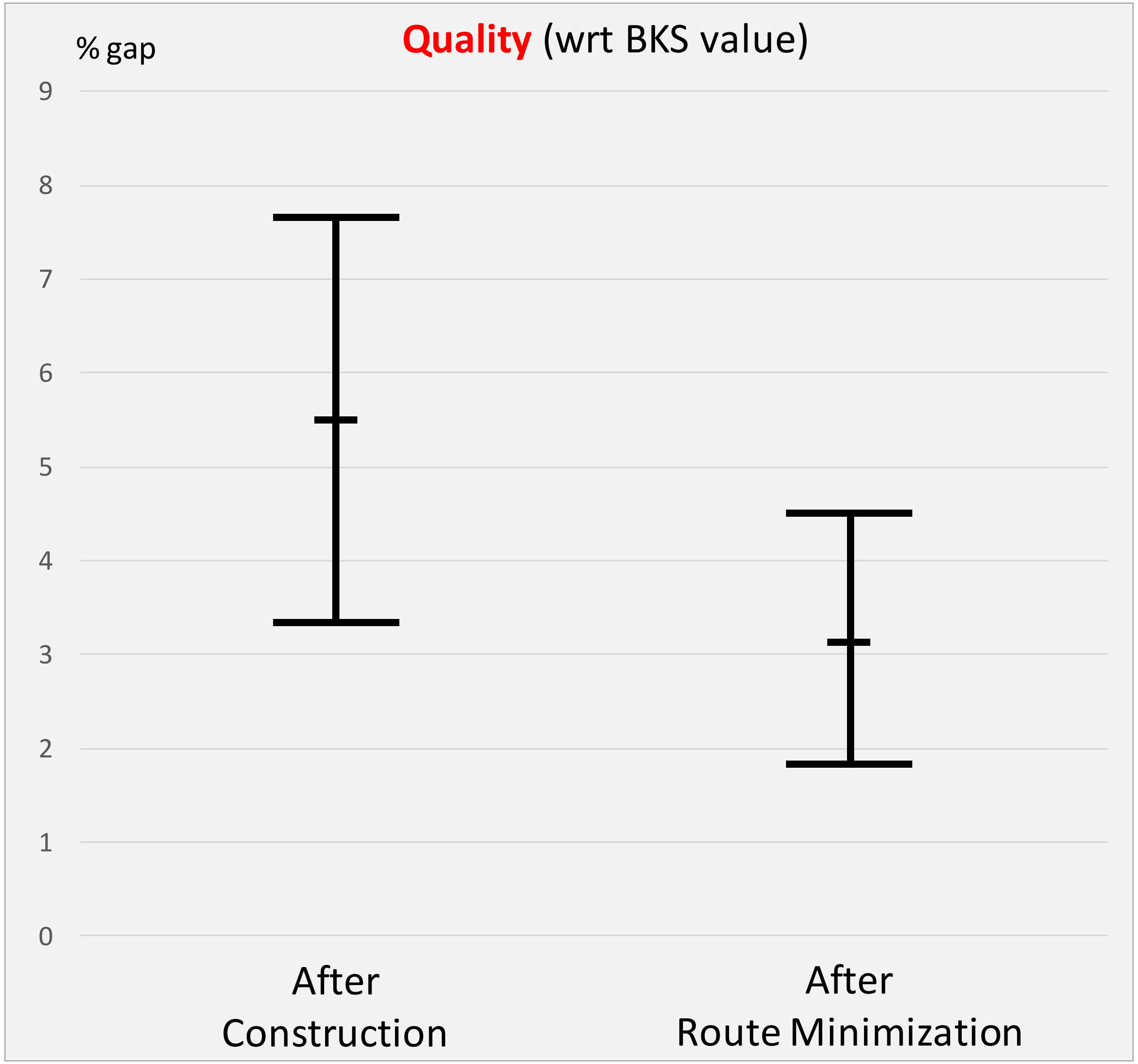
5 **If** $|L| = 0$ **and** $(cost(S) < cost(S^*)$ **or** $cost(S) = cost(S^*)$ **and** $|S| < |S^*|)$ → Set $S^* = S$ → **Early stopping condition** **If** $|S^*| \leq k$ → **Stop**

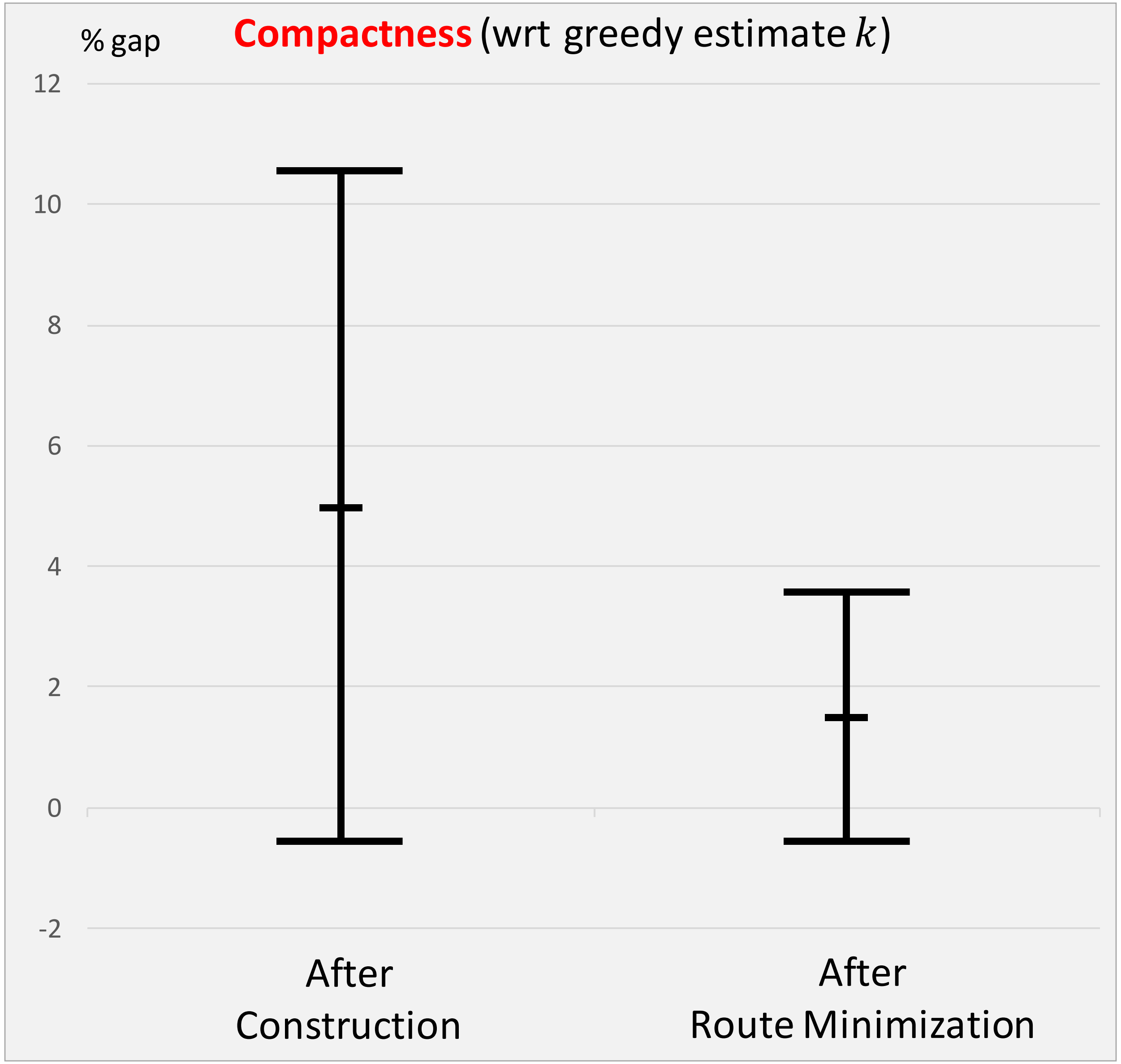
6 $p = z \cdot p$

7 **If** $cost(S) > cost(S^*)$ → Set $S = S^*$

ROUTE MINIMIZATION

IN ABOUT 3 SECONDS





Core Optimization

1 Initialize shaking parameters $\overline{\omega}$ | Initialize sparsification vector $\bar{\gamma}$ | $S^* = S$

Loop

2 Perform a **random walk** ruin-and-recreate application on S to obtain S'

3 Optimize the S' using the LS engine

- **If** $cost(S') < cost(S^*)$ | $S^* = S'$ | Reset $\bar{\gamma}$
- Otherwise | Update $\bar{\gamma}$

4 Update $\overline{\omega}$

4 **If** $accept(S', t)$ | $S = S'$

4 $t = c \cdot t$

Random Walk Ruin-and-Recreate

1 Select a seed customer s — $i = s$

Loop ω_s times

2 Select the next customer j

- Move within same route
  - Forward
  - Backward
- Jump to a neighbor route
  - Any
  - Not yet visited one — **If** not available — **Stop**

3 Remove i — $i = j$

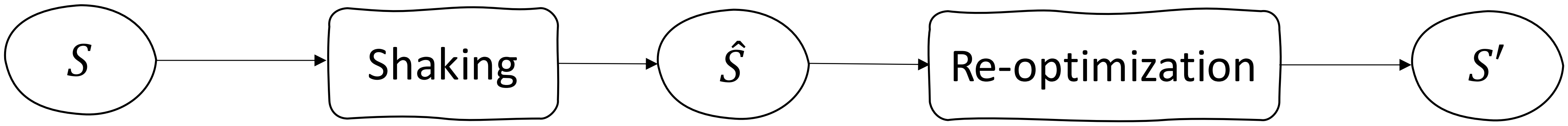
For each removed customer i

4 Find best insertion position of i in **existing routes**

- **If** a position is found — Insert i
- None of the routes can accomodate i — Create a route with i

A DECLARATIVE SELECTION OF SHAKING PARAMETERS $\overline{\boldsymbol{\omega}}$

A structure-aware and quality-oriented shaking meta-strategy

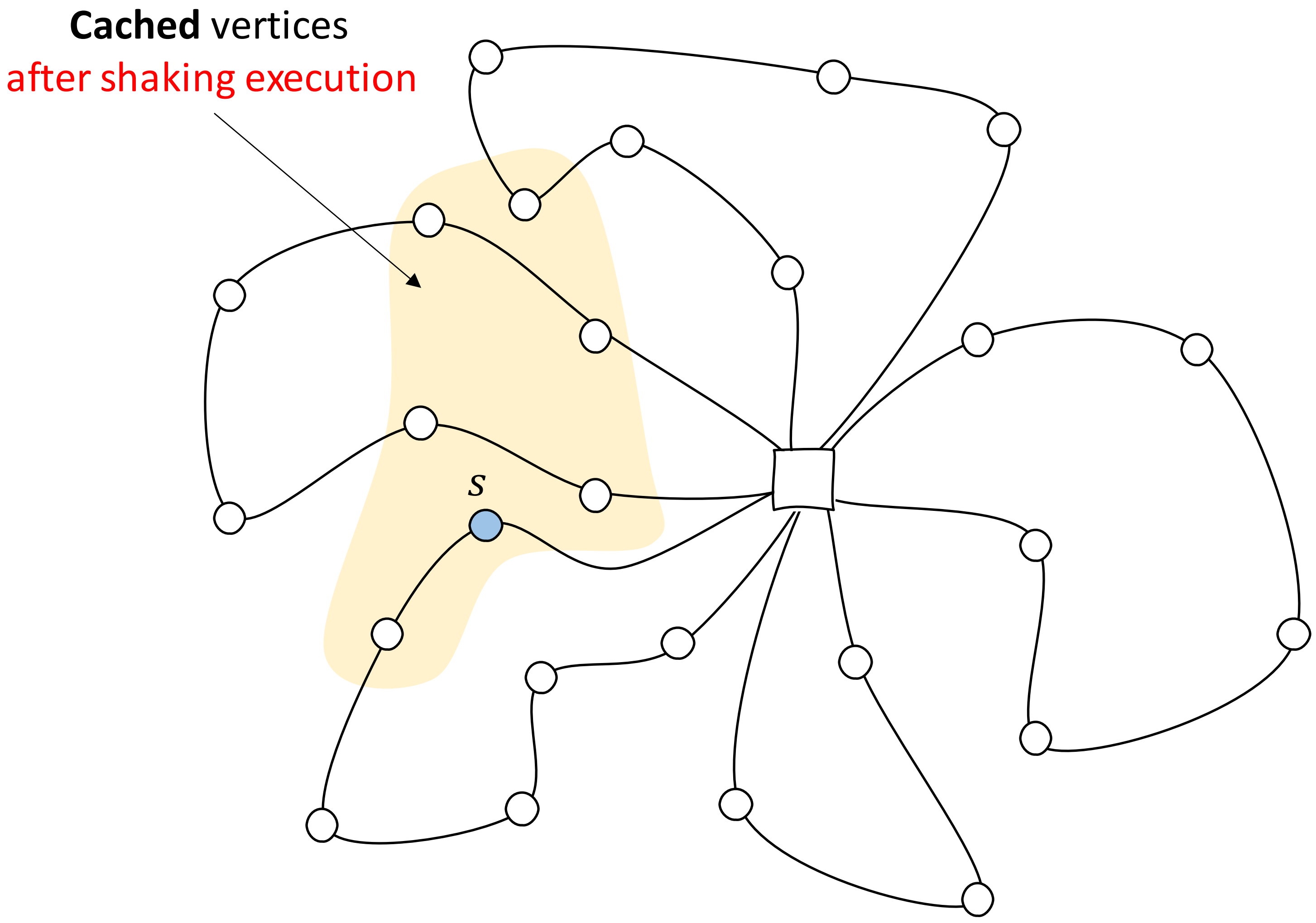


Random walk of length ω_s
from a seed customer s

Compare S with S’ and introduce a feedback to adjust the shaking intensity

A DECLARATIVE SELECTION OF SHAKING PARAMETERS $\overline{\omega}$

SVC TO UPDATE SHAKING PARAMETERS

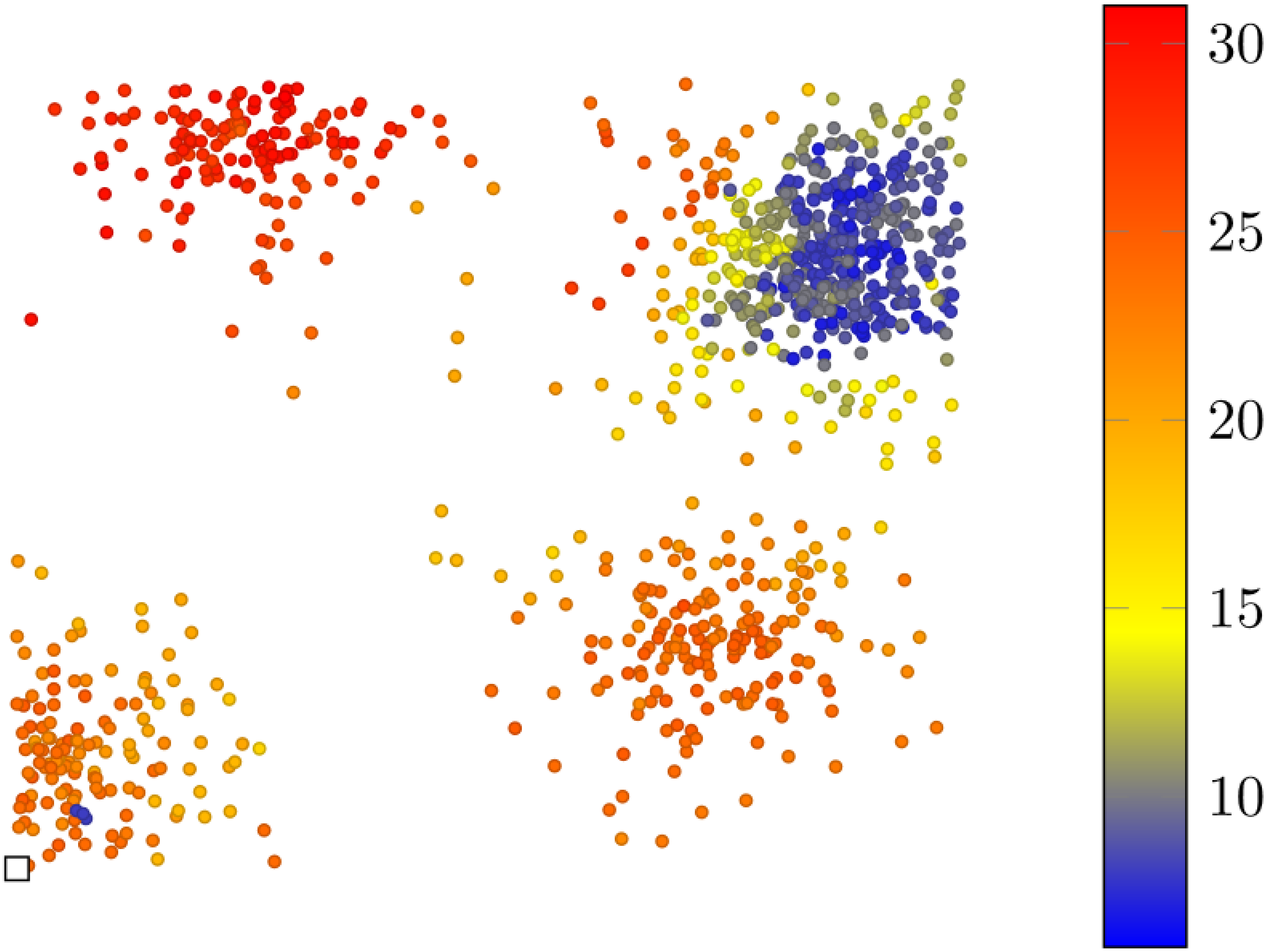


Update rule

$$\begin{cases} \omega_i = \omega_i - 1 & \text{if SHAKING TOO } \textbf{STRONG} \\ \omega_i = \omega_i + 1 & \text{if SHAKING TOO } \textbf{MILD} \\ \text{Randomly increase or decrease } \omega_i & \text{if SHAKING } \textbf{OK} \end{cases}$$

$i \in \overline{V_{\hat{S}}}$

A DECLARATIVE SELECTION OF SHAKING PARAMETERS $\overline{\omega}$



COMPUTATIONAL RESULTS

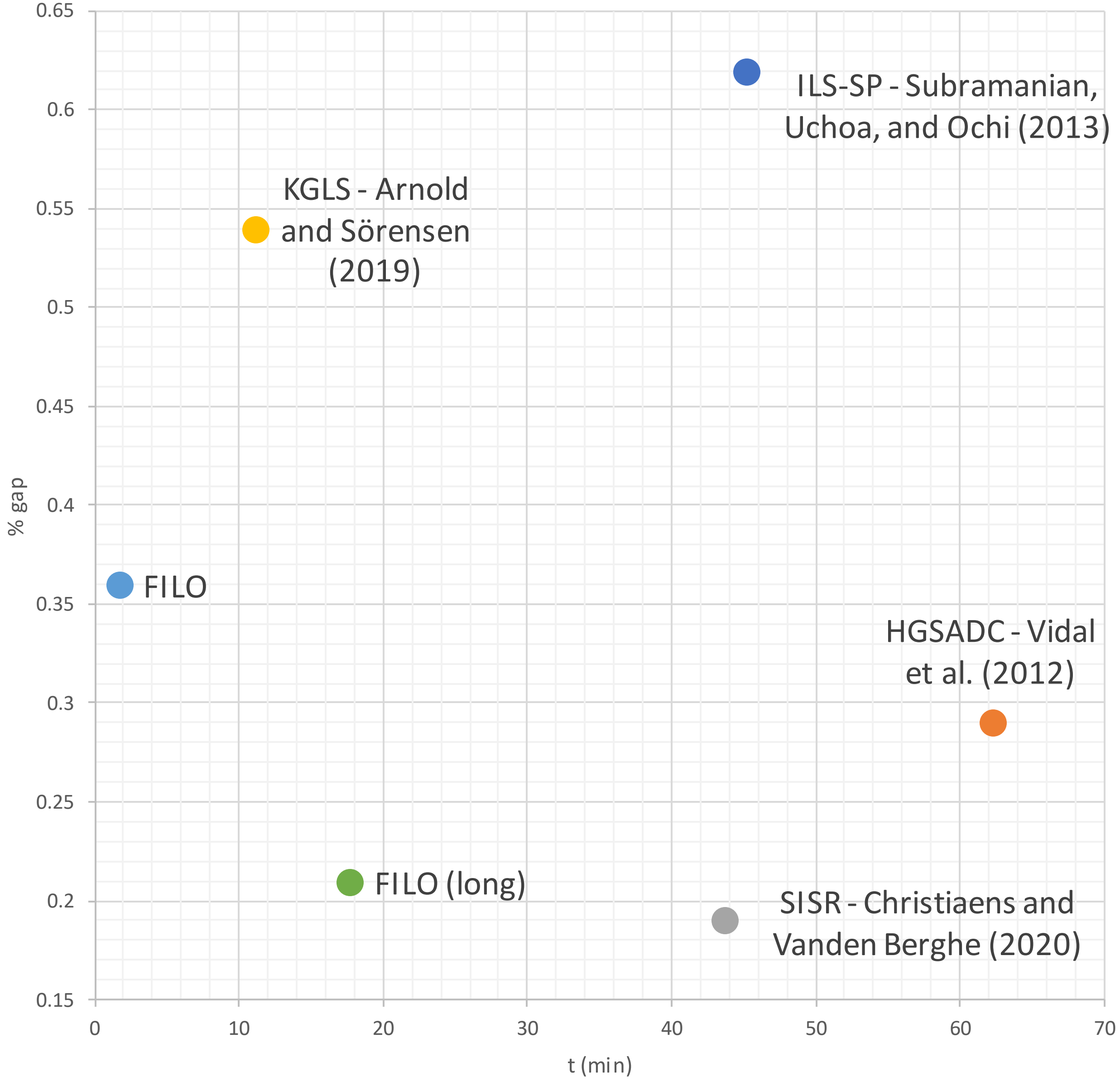
- Two versions of FILO
  - FILO $100K$ core optimization iterations
  - FILO (long) $1M$ core optimization iterations

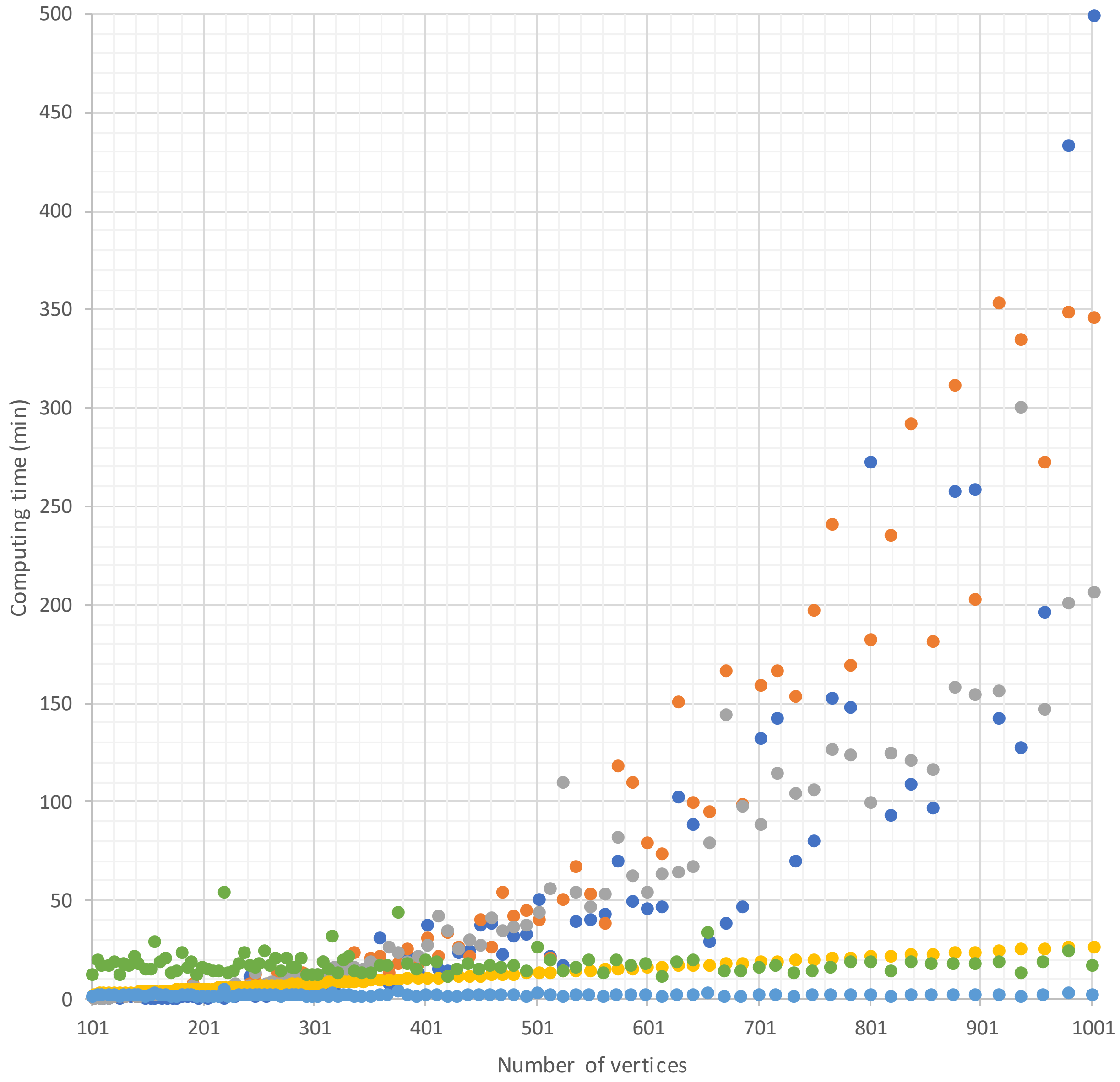
- On *standard* instances
  - **X** dataset by **Uchoa et al. (2017)**

- On very large-scale instances
  - **B** dataset by **Arnold, Gendreau, and Sörensen (2019)**
  - **K** dataset by **Kytöjoky et al. (2007)**
  - **Z** dataset by **Zachariadis and Kiranoudis (2010)**

X

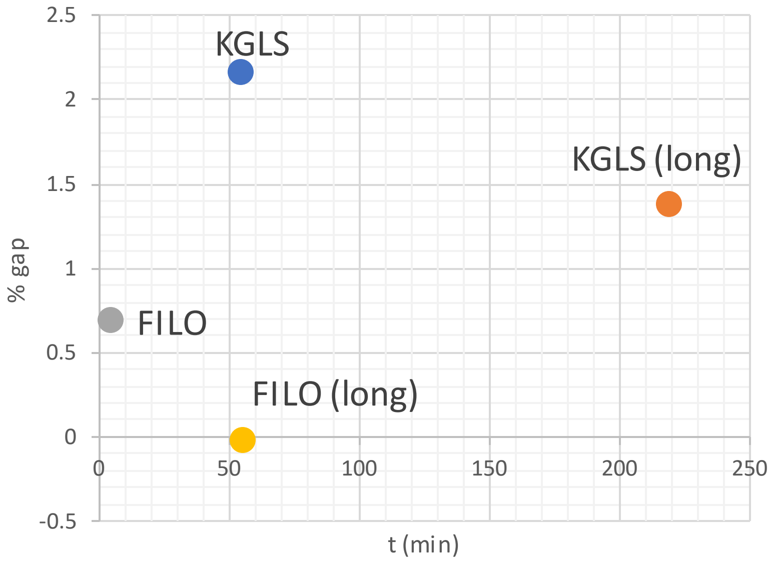
UCHOA ET AL. (2017)



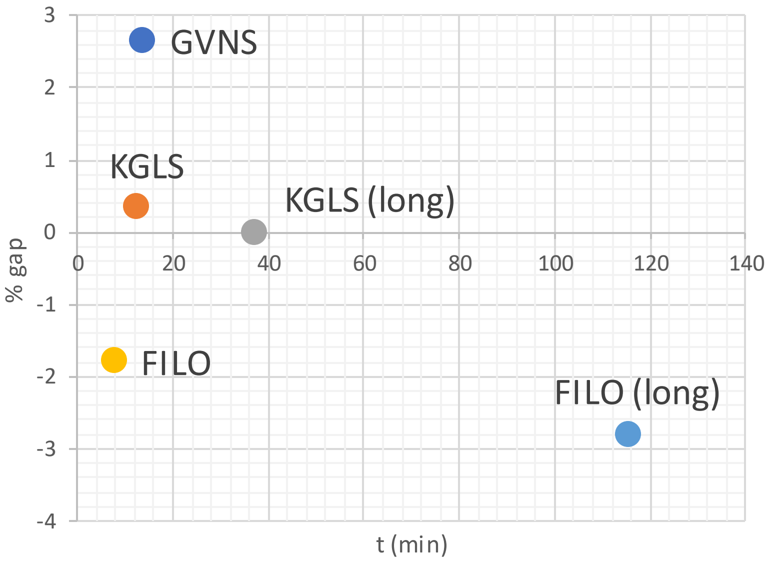


VERY LARGE INSTANCES

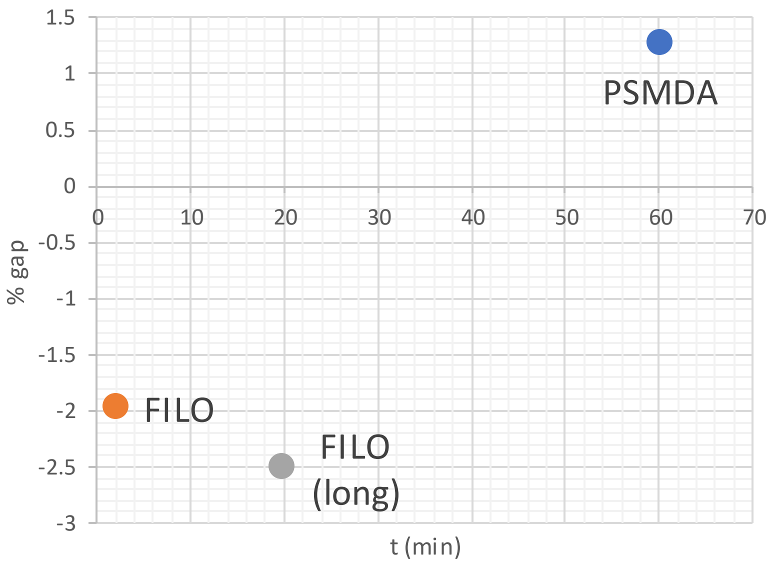
B (3K – 30K)
Arnold, Gendreau, and Sörensen (2019)



K (≈8K – 12K)
Kytöjoky et al. (2007)



Z (3K)
Zachariadis and Kiranoudis (2010)



Algorithms

- KGLS, KGLS (long) - Arnold, Gendreau, and Sörensen (2019)
- GVNS - Kytöjoky et al. (2007)
- PSMDA - Zachariadis and Kiranoudis (2010)

Thank You!

Report, slides and code

https://github.com/acco93/filo