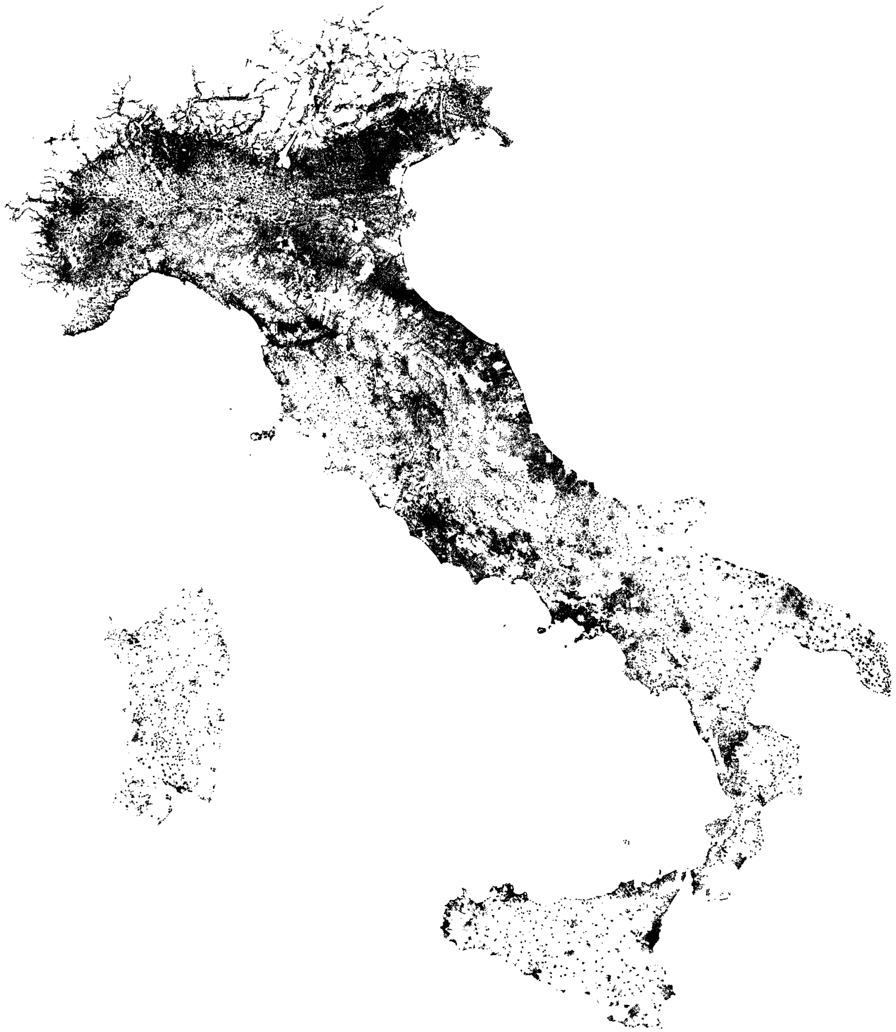
One Million… and Beyond ! Solving Huge-Scale Vehicle Routing Problems in a Handful of Minutes

A five years journey from FILO to FILO2 ... via FSPD

Daniele Vigo

DEI «Guglielmo Marconi», University of Bologna
CIRI ICT, University of Bologna

based on joint works with: L. Accorsi
and F. Cavaliere, D. Lagana, R. Musmanno



Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani Piano Nazionale di Ripresa e Resilienza | Alma Mater Studiorum Università di Bologna

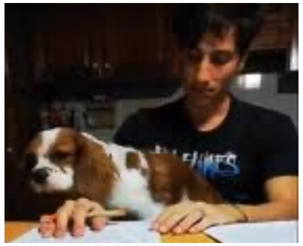
funded by PRIN2022, H2020 Tuples, AFORS

Outlook

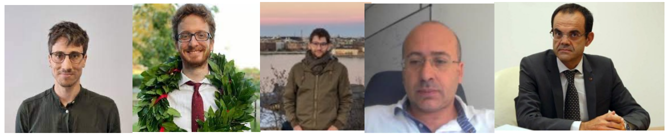
- Motivation and Introduction to VRP and CVRP
- FILO: A Fast and Scalable Heuristic for the Solution of Large-Scale Capacitated Vehicle Routing Problems (with L. Accorsi, TS, 2021)
- Extending FILO:
  - FSPD: Very Large-Scale VRPs with Pickup and Delivery
    (with F. Cavaliere, L. Accorsi, D. Lagana and R. Musmanno, submitted 2023)
  - FILO2: Huge-scale CVRPs instances (with L. Accorsi, C&OR 2024)

Motivation

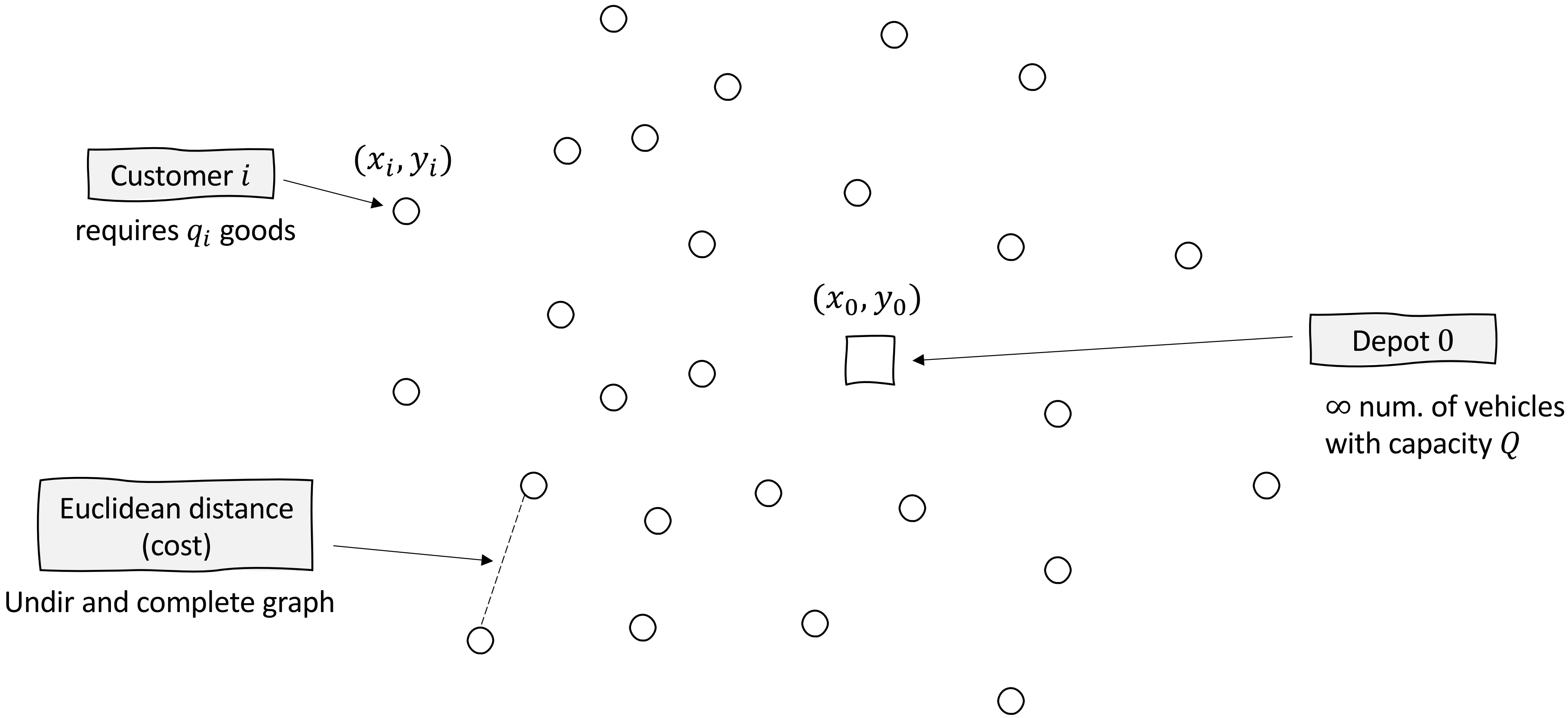
- I devoted the largest part of my research activity to VRP and its (heuristic) solution
- Some ideas (e.g. Granular TS, Decomposition …) gained some attention from the community … some others (e.g. adaptive guidance, visual beauty) much less …
- In the last years thanks to the PhD of Luca Accorsi I had the opportunity to combine many old and new ideas to build an innovative framework for the solution of large scale CVRP: FILO



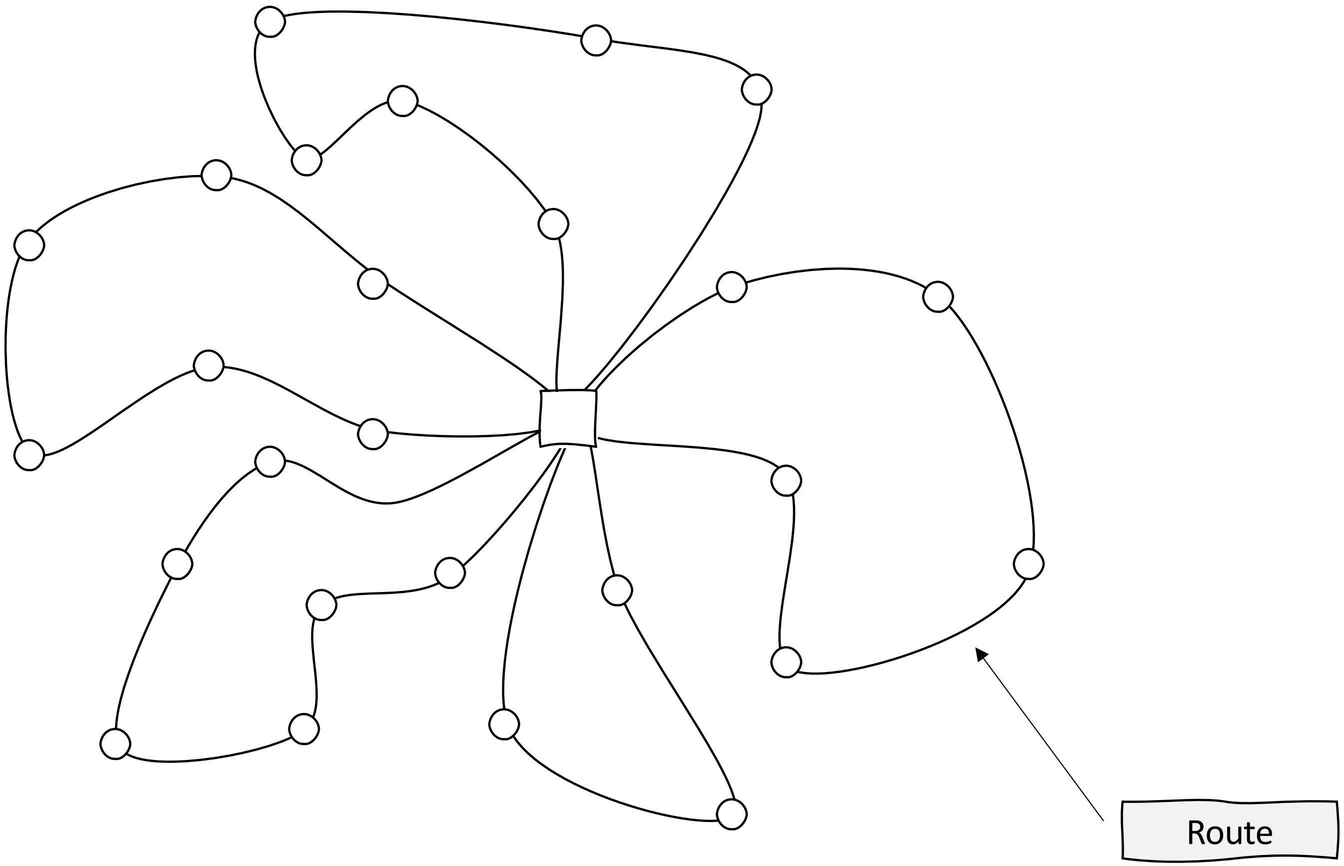
- With the help of several people FILO evolved in several directions



Capacitated Vehicle Routing Problem (CVRP) instance

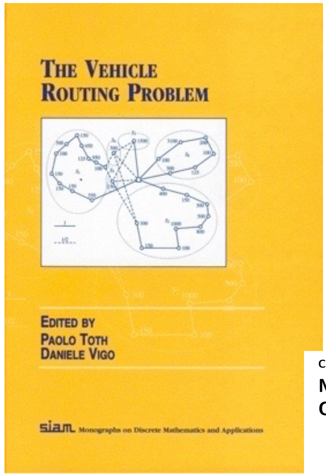


Capacitated Vehicle Routing Problem (CVRP) solution



Main references

- Classical Methods (1960-2000)



Chapter 5

Classical Heuristics for the Capacitated VRP

Gilbert Laporte
Frédéric Semet

5.1 Introduction

Several families of heuristics have been proposed for the *Vehicle Routing Problem* (VRP). These can be broadly classified into two main classes: *classical heuristics* developed mostly between 1960 and 1990, and *metaheuristics* whose growth has occurred in the last decade. Most standard construction and improvement procedures in use today belong to the first class. These methods perform a relatively limited exploration of the search space and typically produce good quality solutions within modest computing times. Moreover, most of them can be easily extended to account for the diversity of constraints encountered in real-life contexts. Therefore, they are still widely used in

rming a deep explo-
e methods typically
res, and recombina-
hods is much higher
ncreased computing
require finely tuned
lifficult. In a sense,
edures and they can
. However, because
ethods, they will be

ee categories. *Con-*
g an eye on solution
hase heuristics, the
ring of vertices into
back loops between
lasses: *cluster-first,*
the first case, ver-
e is constructed for

Chapter 6

Metaheuristics for the Capacitated VRP

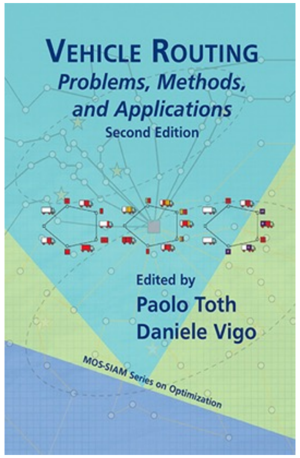
Michel Gendreau
Gilbert Laporte
Jean-Yves Potvin

6.1 Introduction

In recent years several *metaheuristics* have been proposed for the VRP. These are general solution procedures that explore the solution space to identify good solutions and often embed some of the standard route construction and improvement heuristics described in Chapter ??. In a major departure from classical approaches, metaheuristics allow deteriorating and even infeasible intermediary solutions in the course of the search process. The best known metaheuristics developed for the VRP typically identify better local optima than earlier heuristics, but they also tend to be more time consuming.

As far as we are aware, six main types of metaheuristics have been applied to the VRP: 1) Simulated Annealing (SA), 2) Deterministic Annealing (DA), 3) Tabu Search (TS), 4) Genetic Algorithms (GA), 5) Ant Systems (AS), and 6) Neural Networks (NN). The first three algorithms, SA, DA and TS, start from an initial solution x_1, and move at each iteration t from x_t to a solution x_{t+1} in the neighborhood $N(x_t)$ of x_t, until a stopping condition is satisfied. If $f(x)$ denotes the cost of x, then $f(x_{t+1})$ is not necessarily less than $f(x_t)$. As a result, care must be taken to avoid cycling. GA examines at each step a population of solutions. Each population is derived from the preceding one by combining its best elements and discarding the worst. AS is a constructive approach in which several new solutions are created at each iteration using some of the information gathered at previous iterations. As was pointed out by Taillard et al. [63], TS, GA and AS are methods that record, as the search proceeds, information on solutions encountered and use it to obtain improved solutions. NN is a learning mechanism that gradually adjusts a set of weights until an acceptable solution is reached. The rules governing the search differ in each case and these must also be tailored to the shape of the problem at hand. Also, a fair amount of creativity and experimentation is required. Our purpose is to survey some of the most representative applications of local search algorithms to the VRP. For generic articles and textbooks

- Recent Methods (>2000)



Chapter 4

Heuristics for the Vehicle Routing Problem

Gilbert Laporte
Stefan Ropke
Thibaut Vidal

4.1 ▪ Introduction

In recent years, several sophisticated mathematical programming decomposition algorithms have been put forward for the solution of the VRP. Yet, despite this effort, only relatively small instances involving around 100 customers can be solved optimally, and the variance of computing times is high. However, instances encountered in real-life settings are sometimes large and must be solved quickly within predictable times, which means that efficient heuristics are required in practice. Also, because the exact problem definition varies from one setting to another, it becomes necessary to develop heuristics that are sufficiently flexible to handle a variety of objectives and side constraints. These concerns are clearly reflected in the algorithms developed over the past few years. This chapter provides an overview of heuristics for the VRP, with an emphasis on recent results.

The history of VRP heuristics is as old as the problem itself. In their seminal paper, Dantzig and Ramser [19] sketched a simple heuristic based on successive matchings of vertices through the solution of linear programs and the elimination of fractional solutions by trial and error. The method was illustrated on an eight-vertex graph. It was not pursued, but may have inspired the developers of matching-based heuristics (see Altinkemer and Gavish [3], Desrochers and Verhoog [20], and Wark and Holt [91]). Since then, a wide variety of *constructive* and *improvement heuristics* have been proposed, culminating in recent years with the development of powerful *metaheuristics* capable of computing within a few seconds solutions whose value lies within less than one percent of the best known values.

The field of VRP heuristics is now so rich that it makes no sense to provide an exhaustive compilation of them in a book chapter such as this. Instead, we have decided to focus on methods and principles that have withstood the test of time or present some interesting distinctive features. For a more complete description of the classical heuristics and of the early metaheuristics, we refer the reader to the two chapters by Laporte and

Current State-of-the-Art Methods

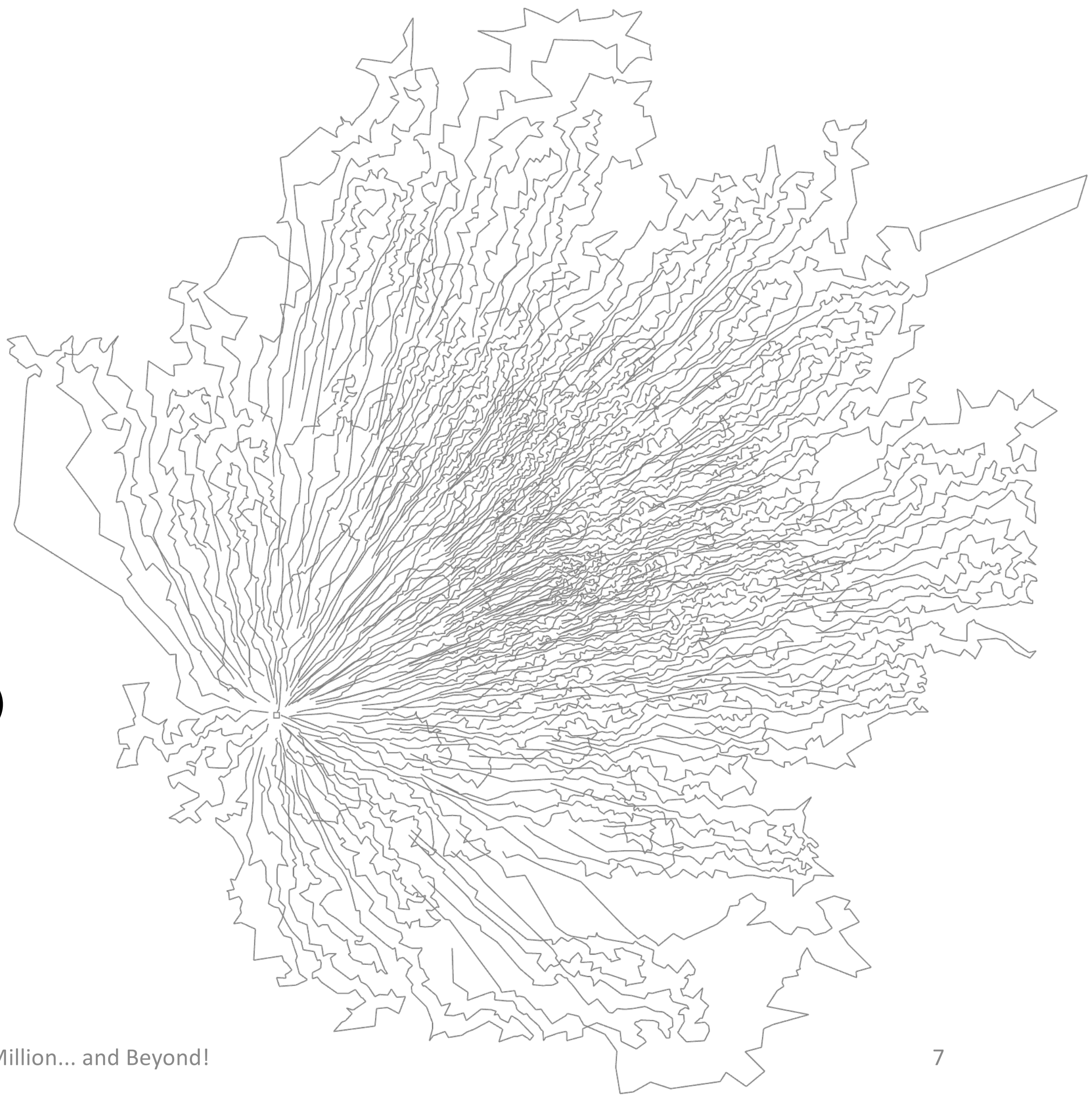
- Vidal et al. (2012): Hybrid Genetic Algorithm
- Subramanian, Ochi, Uchoa (2013): ILS+ SP
- Arnold and Sorensen (2019): Guided LS + ML penalization
- Christiaens and Vande Berghe (2020): Ruin and recreate based o string removal and insertion
- DIMACS VRP Challenge 2022-23

Part 1: FILO …
the godfather

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

Luca Accorsi and Daniele Vigo

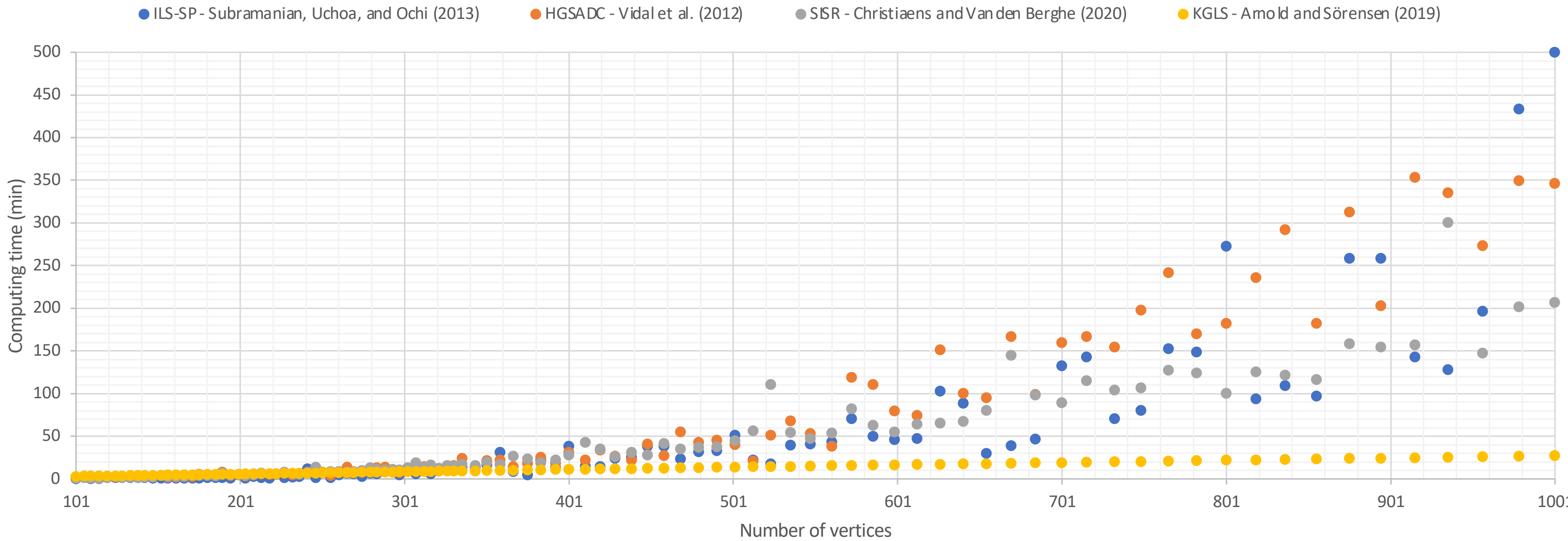
Transportation Science, 55(4):832-856 (2021)



Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

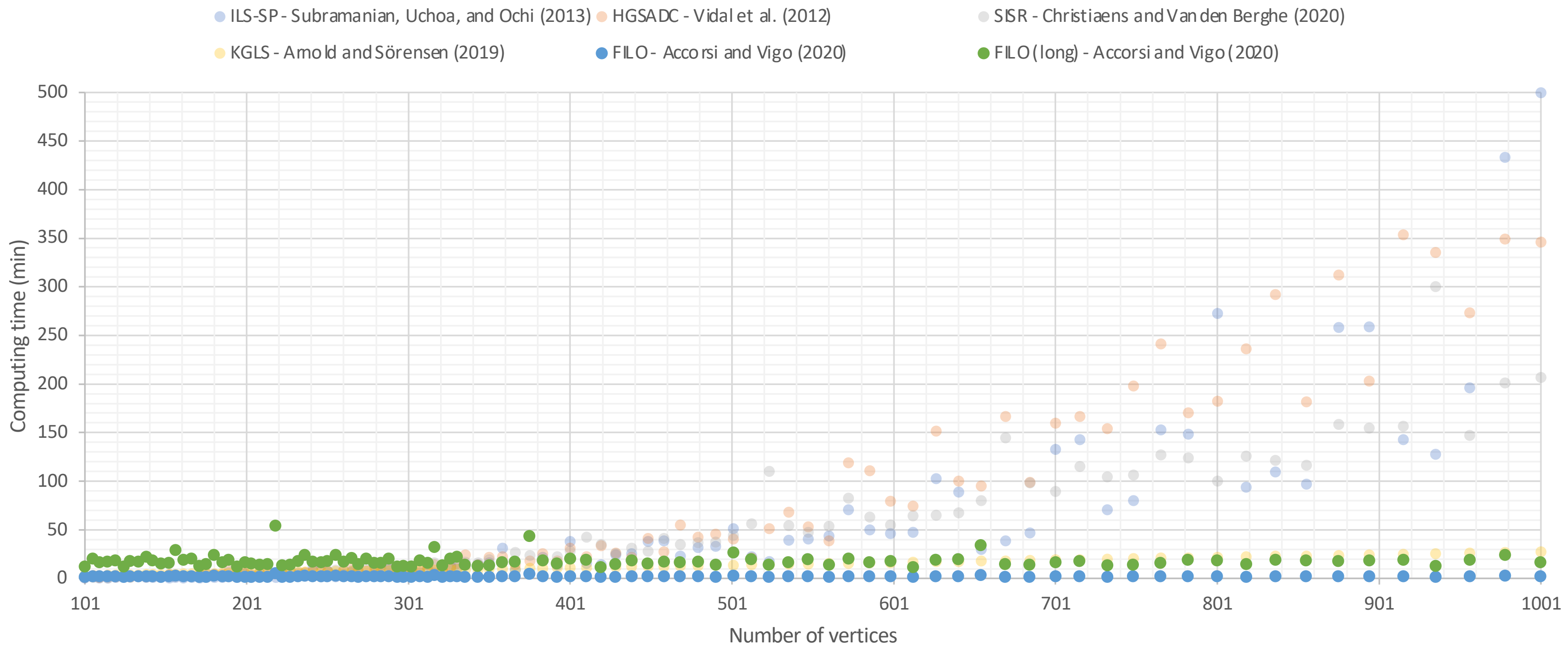
Motivation

- Best (heuristic) CVRP algorithms exhibit a quadratic growth
- 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
- Pruning Techniques
- Careful Design
- Careful Implementation
- Careful Parameters Tuning

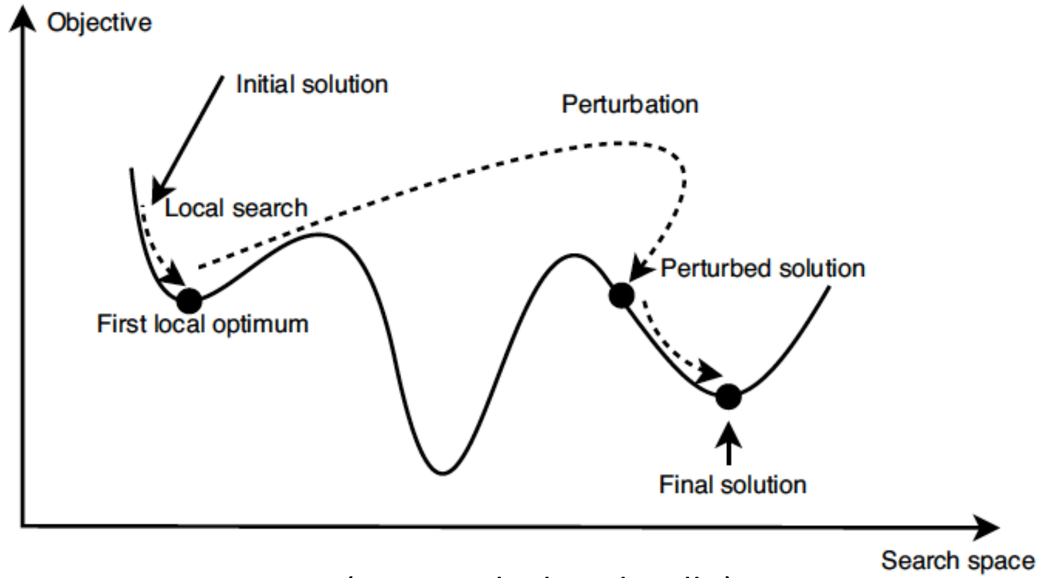
- … a lot of work and attention to details (where the devil hides !!!)

The basic ILS framework

- our approach is broadly based on the Iterated Local Search framework (Lourenço, Martin, Stützle, 2003)

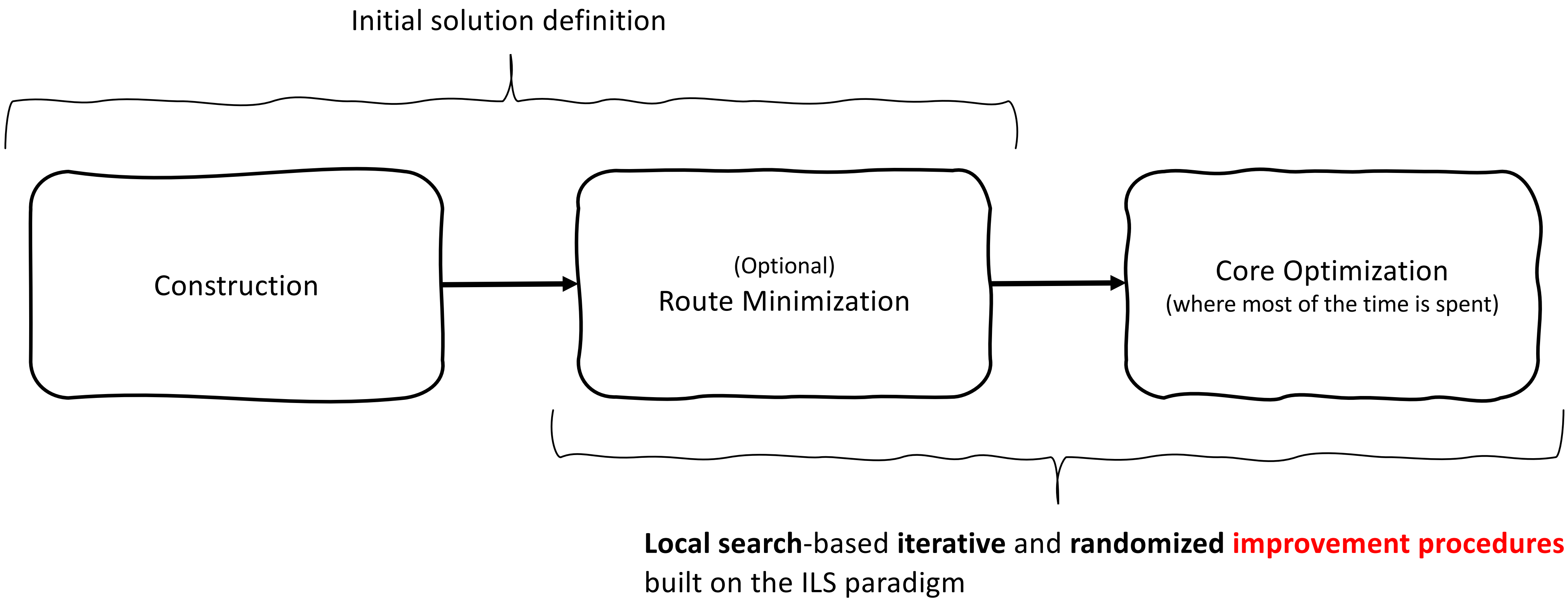
x* = starting solution
repeat
- perturb x*;
- x'=LS(x*)
- possibly replace x* with x'

until stop condition



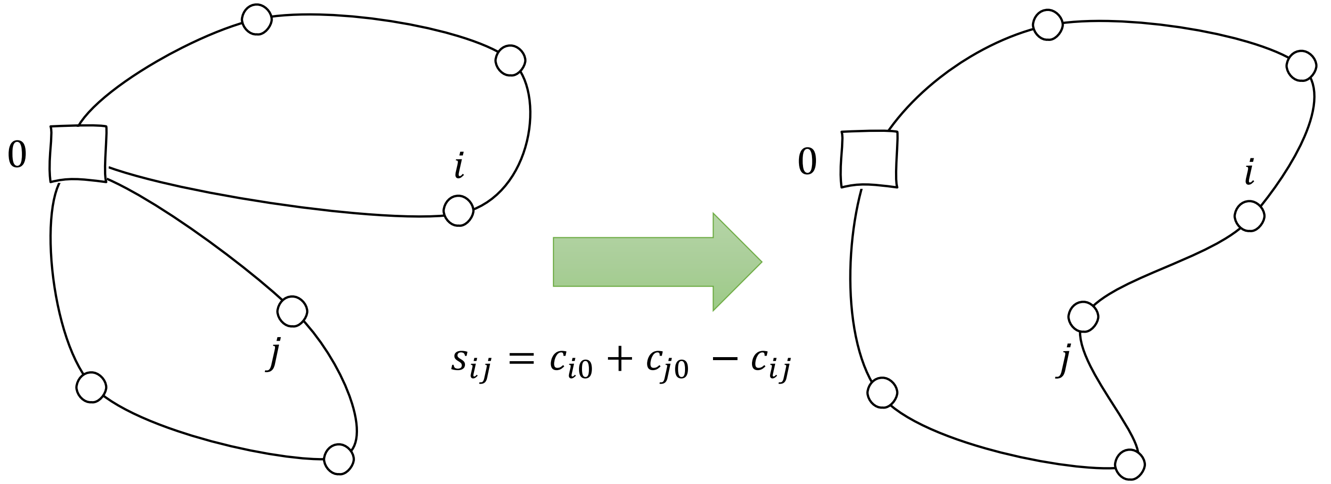
(source El-Ghazali Talbi)

Fast ILS Localized Optimization (FILO)



Construction

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



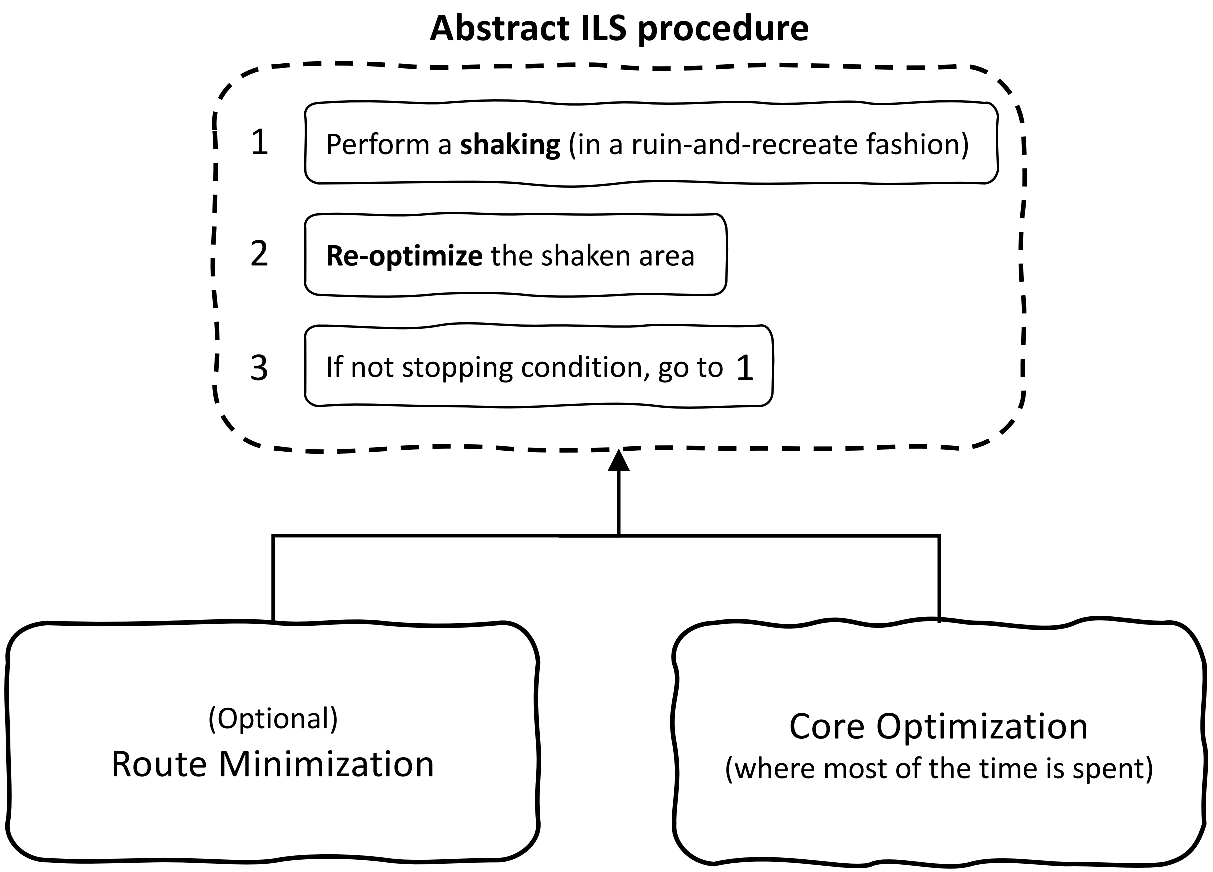
$$s_{ij} = c_{i0} + c_{j0} - c_{ij}$$

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

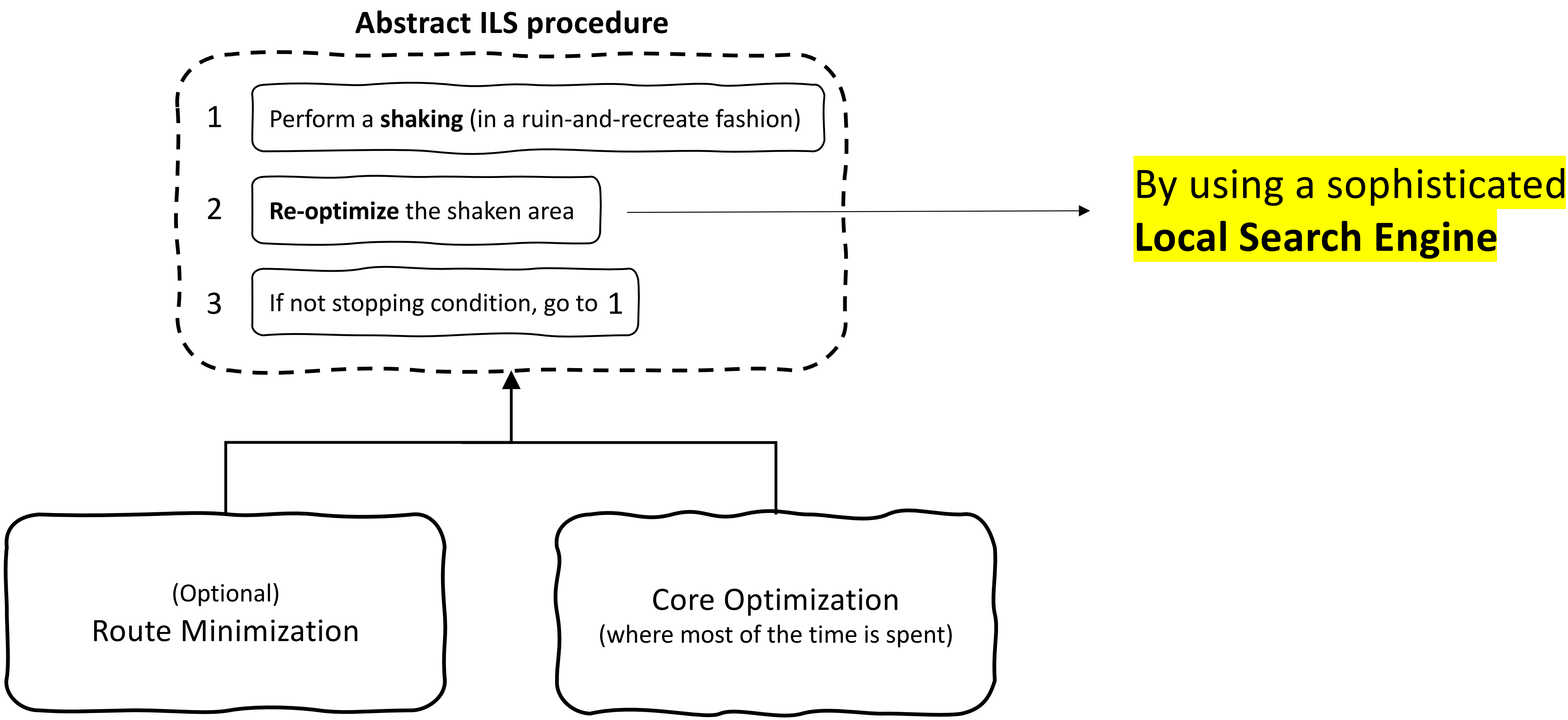
- For each i, compute s_{ij} only for $j \in Neighbors(i, 100)$ and $i < j$

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

Improvement procedures



Improvement procedures



Local search engine

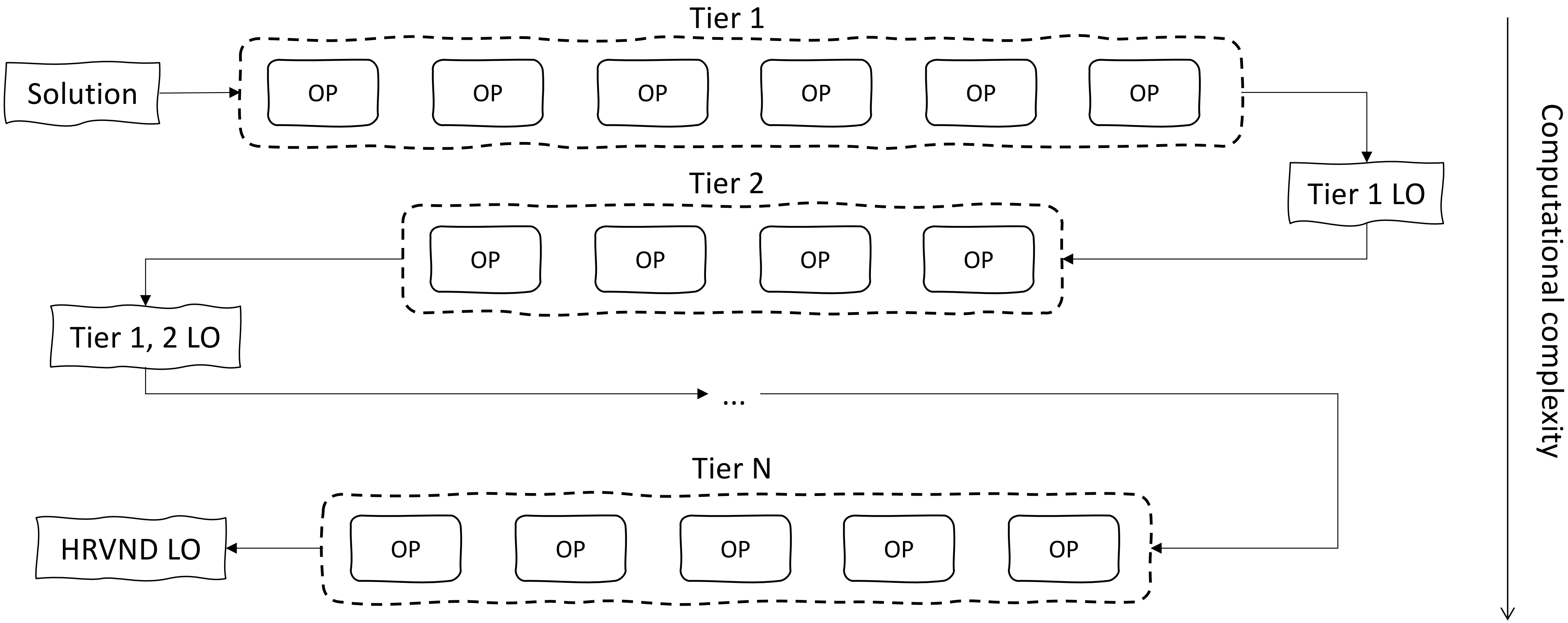
- Several operators explored in a VND fashion
  - Hierarchical Randomized Variable Neighborhood Descent

- Acceleration techniques for neighborhood exploration
  - Static Move Descriptors

- Pruning techniques
  - Granular Neighborhoods and Selective Vertex Caching

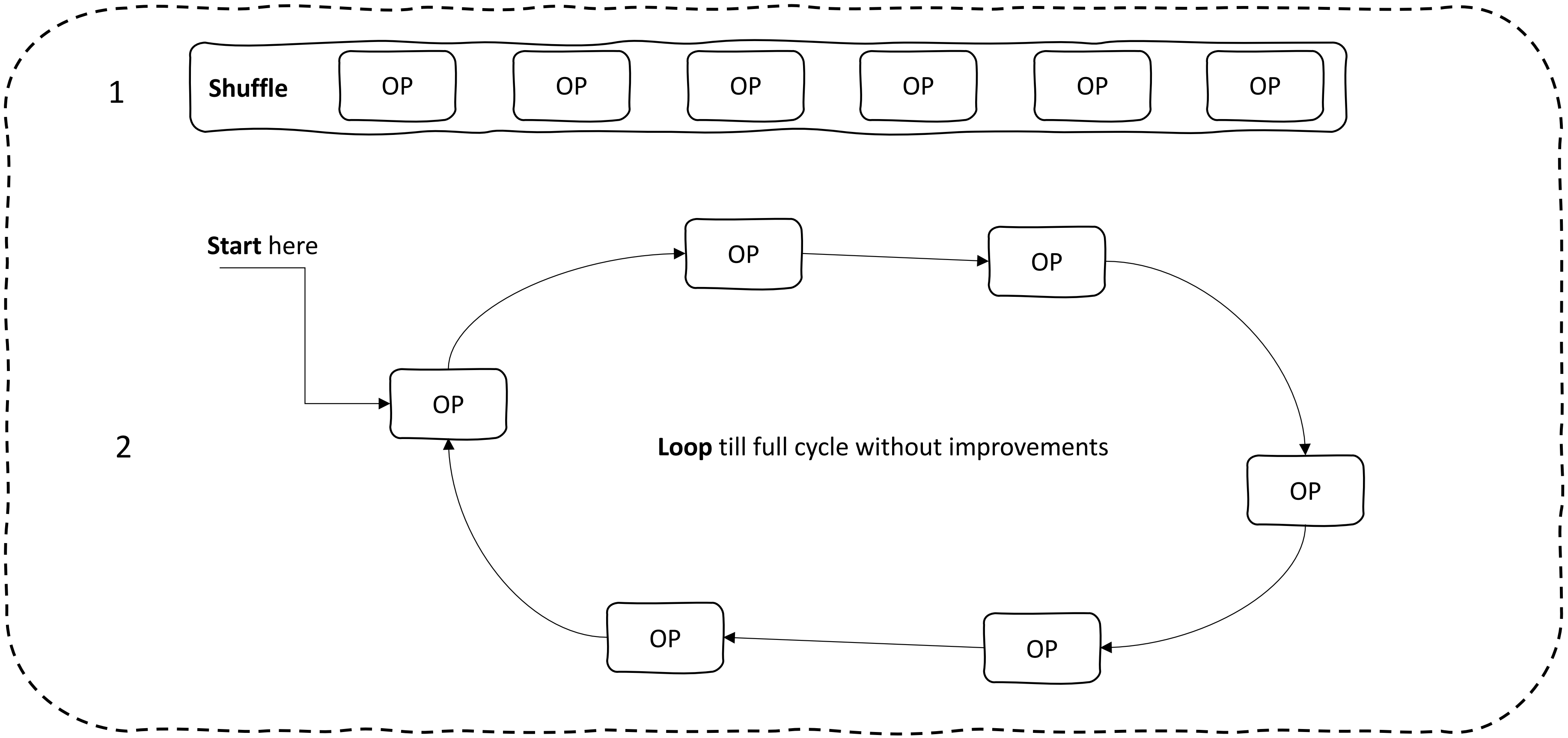
Hierarchical Randomized Variable Neighborhood Descent (HRVND)

An effective organization of several local search operators

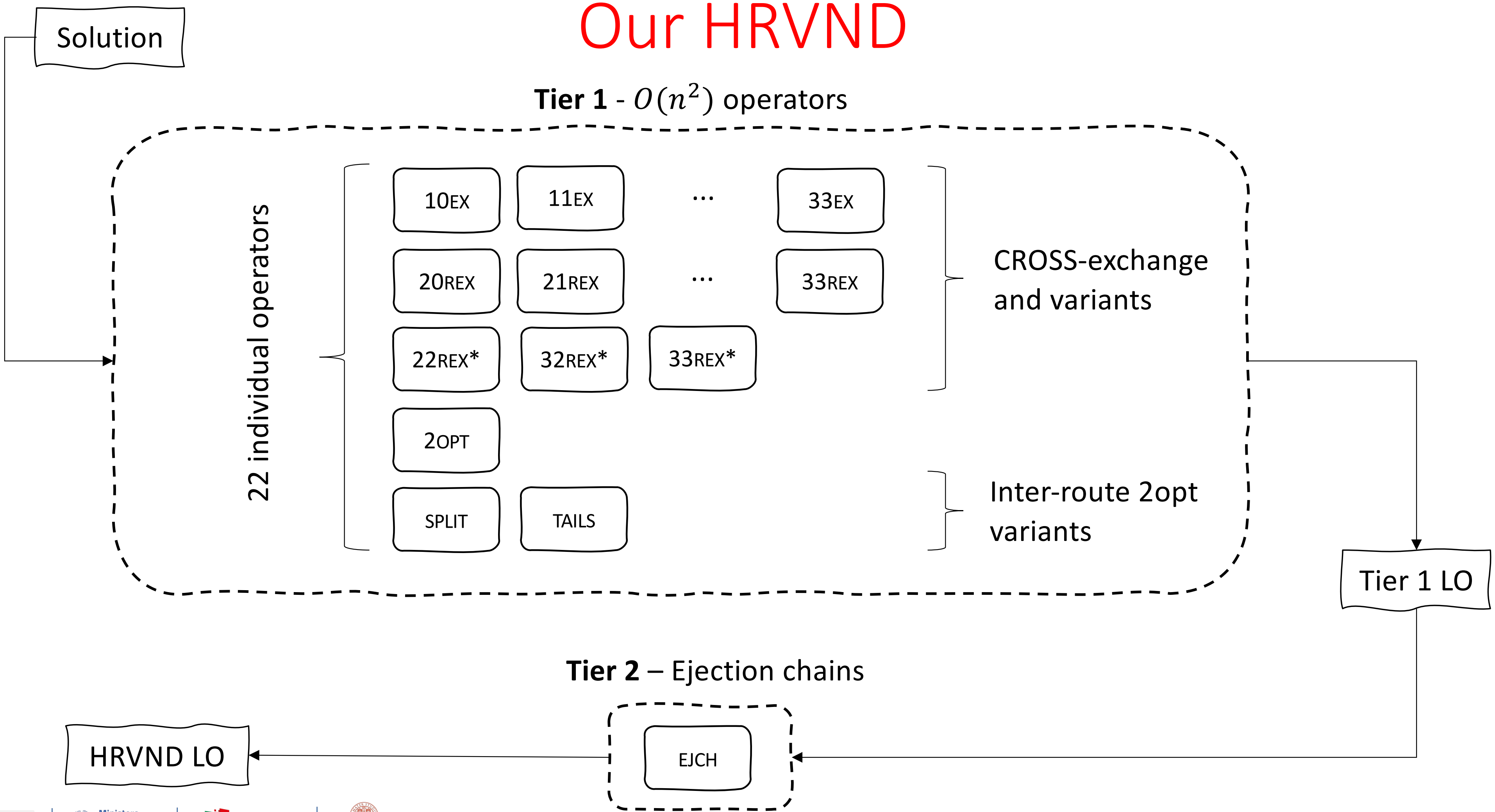


HRVND

Tier application (RVND)



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 chains
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

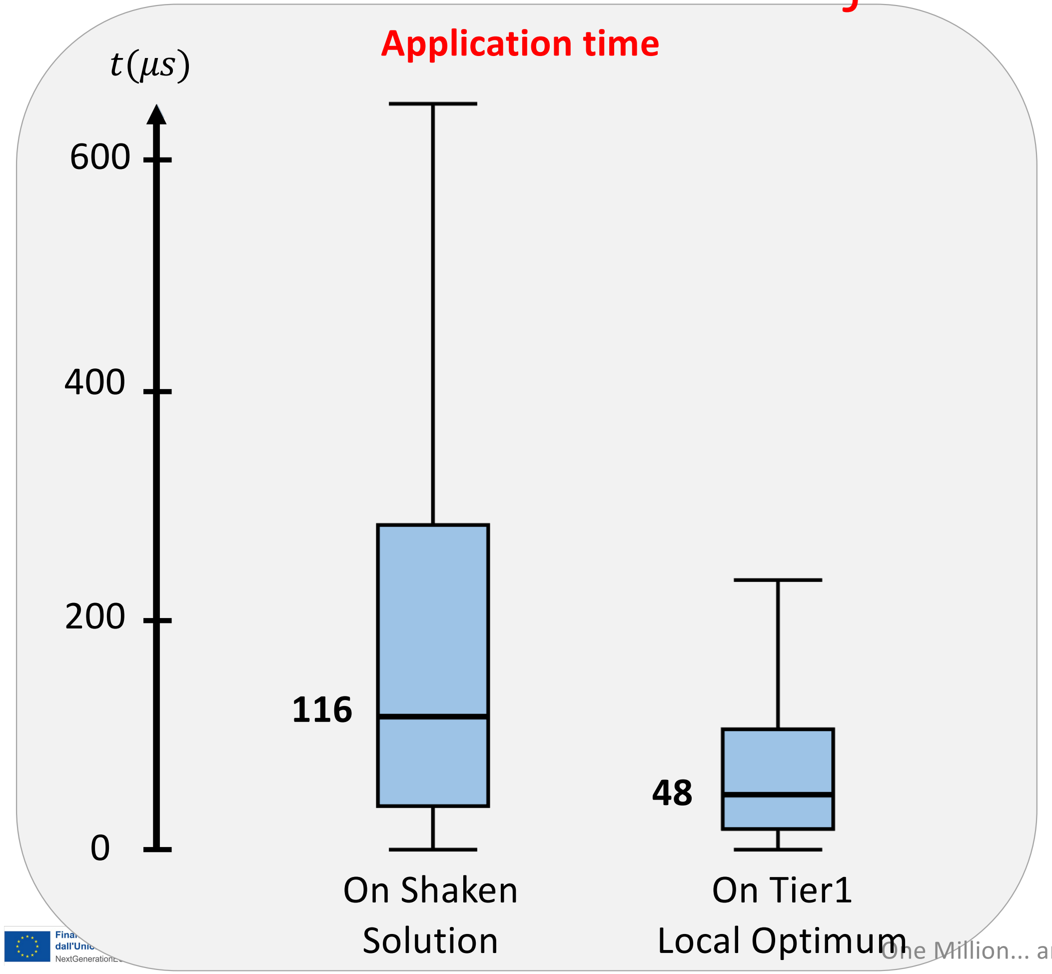
- From VND
  - more complex operators are executed after simpler ones in subsequent tiers
    - to further polish solutions and escape from local optima

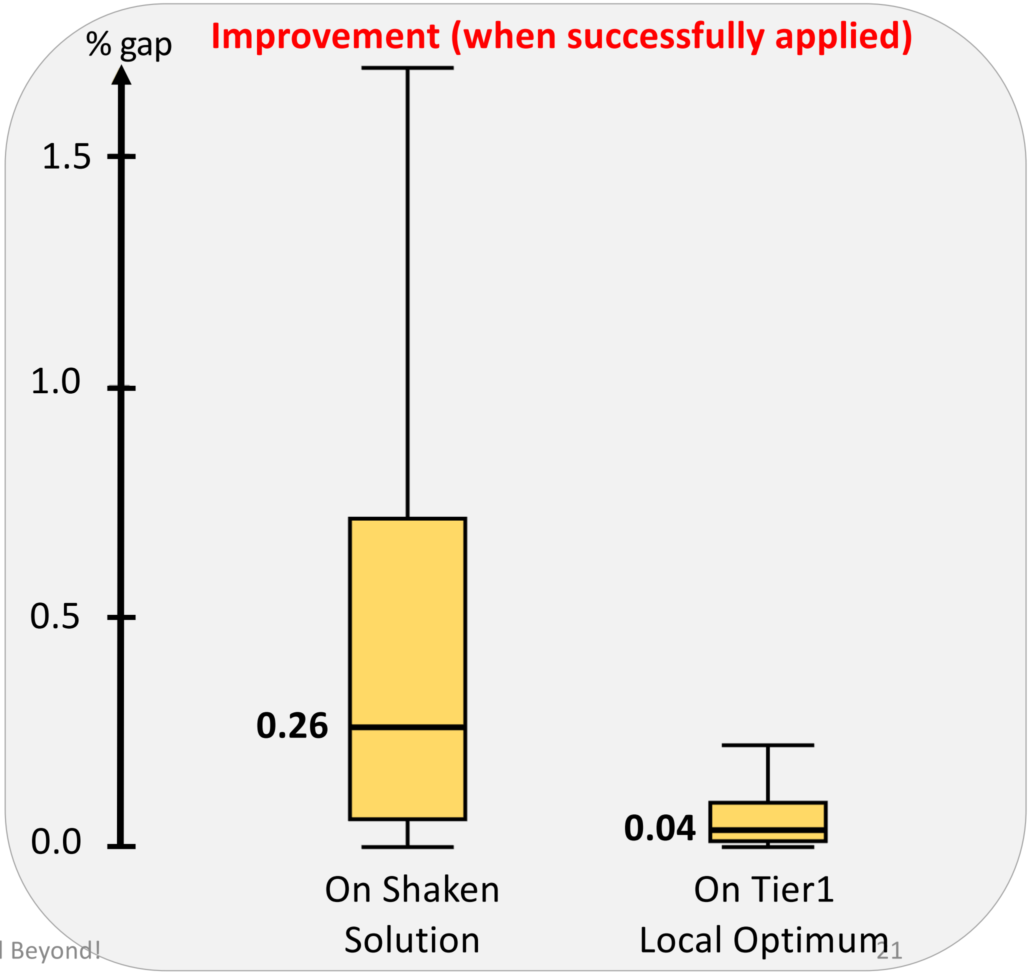
Complex operators expected application time (as well as their improvement) is reduced because they are applied on already high-quality solutions

HRVND motivation: ejection chain

Success ratio

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

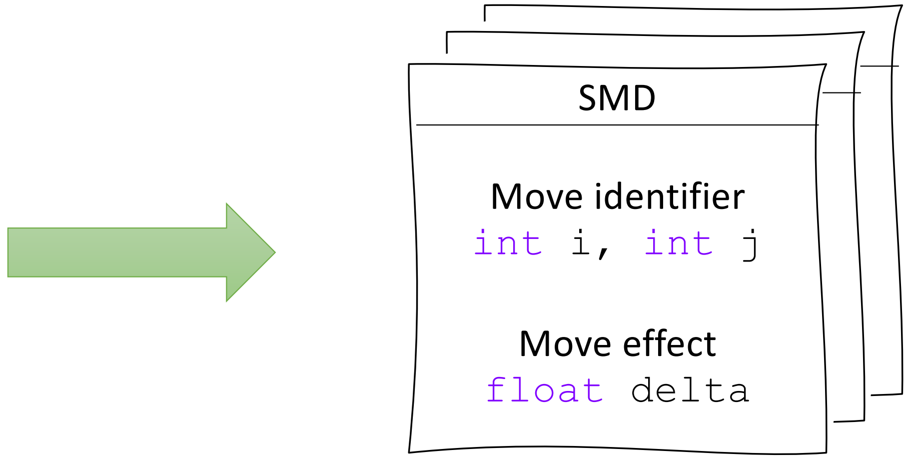




Static Move Descriptors (SMDs)

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)
  }
}
```

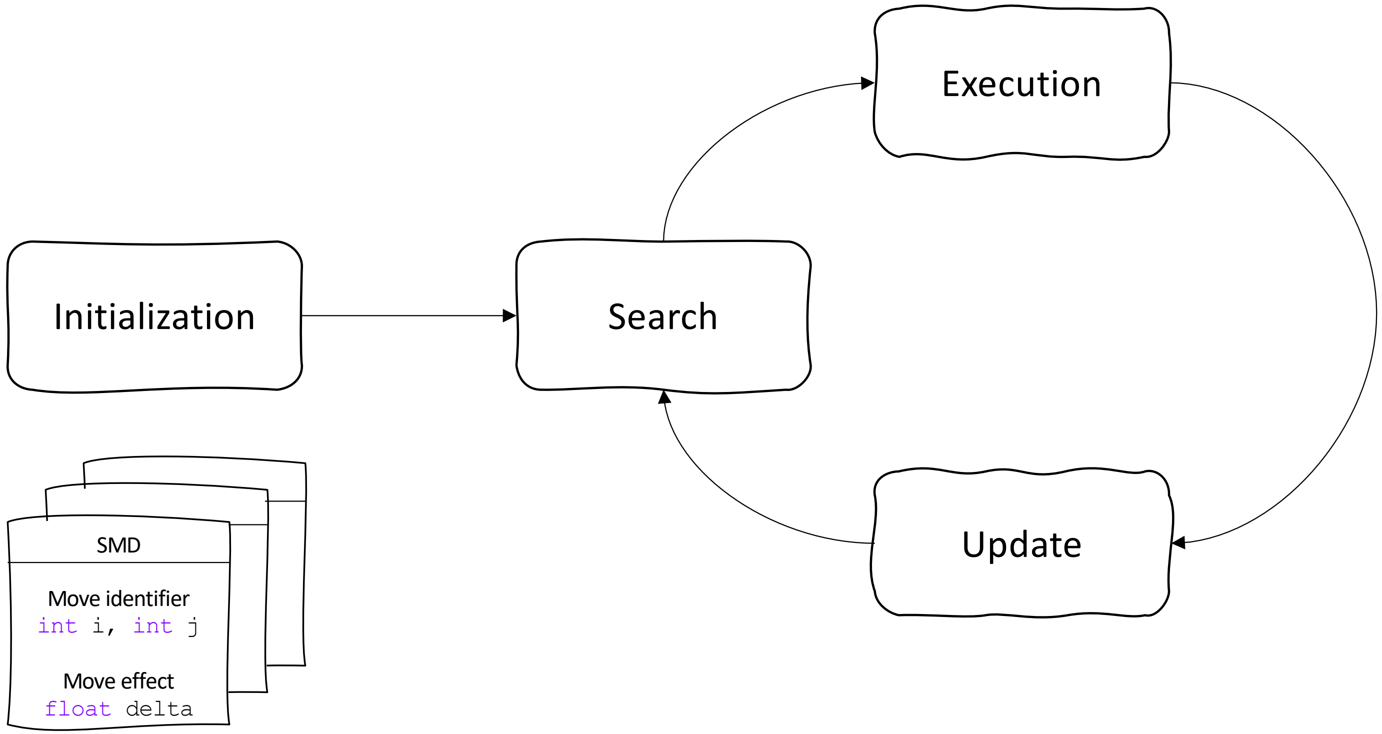


BIBLIOGRAPHY FOR SMDs

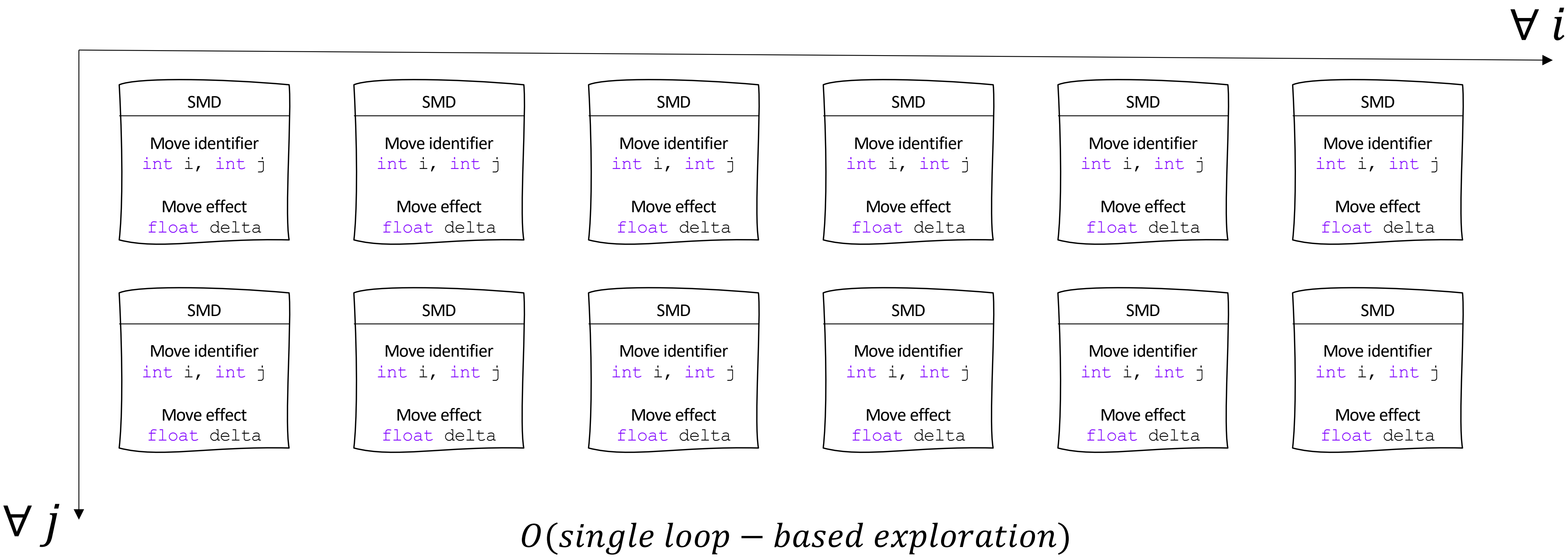
- Emmanouil E. Zachariadis, Chris T. Kiranoudis, A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem, Computers & Operations Research, Volume 37, Issue 12, 2010, Pages 2089-2105
- Onne Beek, Birger Raa, Wout Dullaert, Daniele Vigo, An Efficient 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

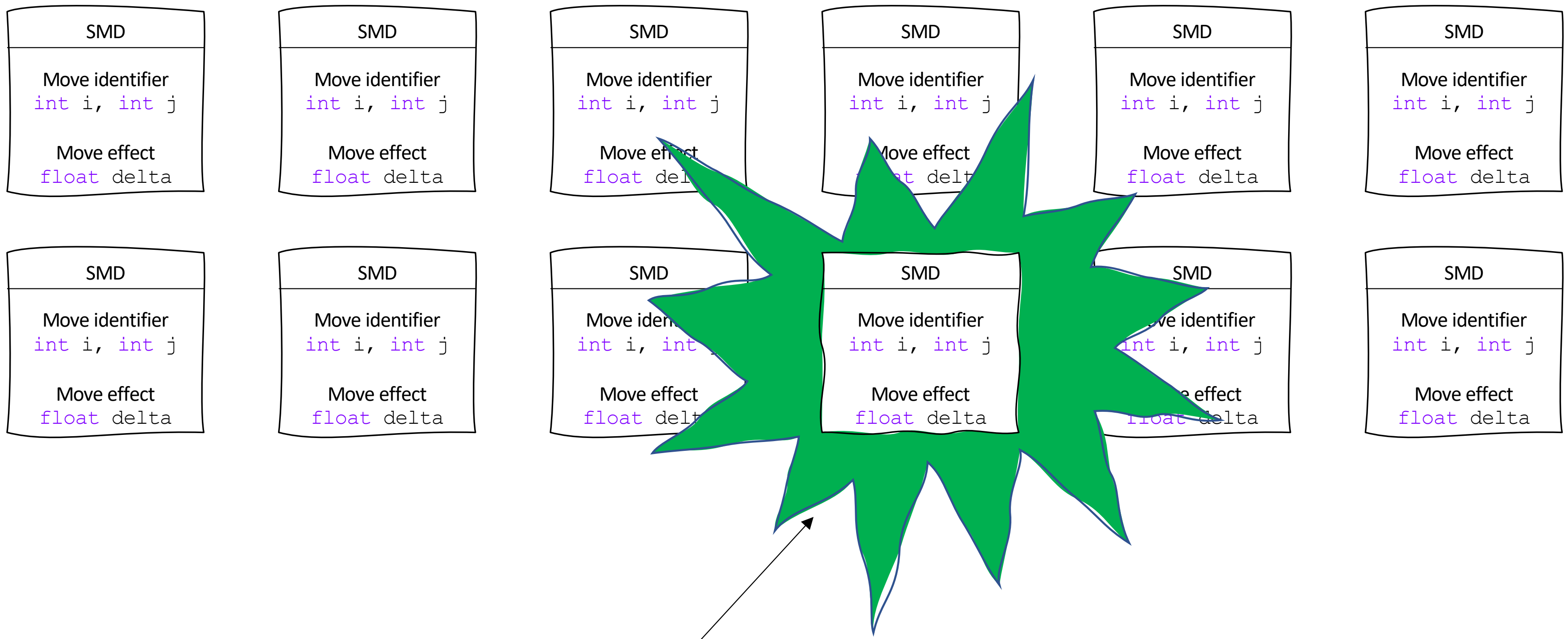


SMD Initialization



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

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

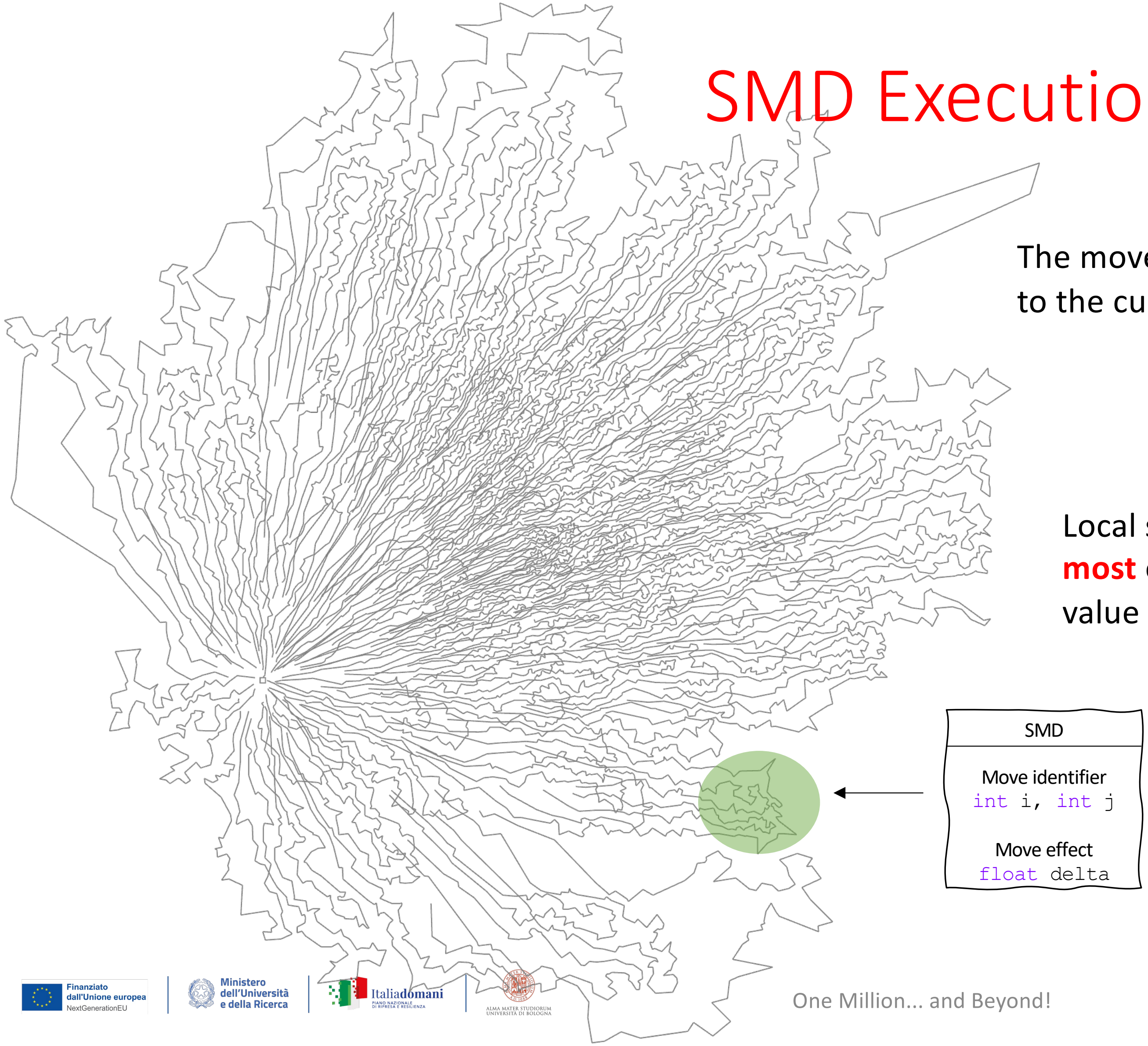
Beek et al. (2018) suggest to linearly scan the heap to avoid removal and reinsertion for each SMD not feasible

- No more guarantees of retrieving the best SMD …
- … However, the heap entries are roughly sorted

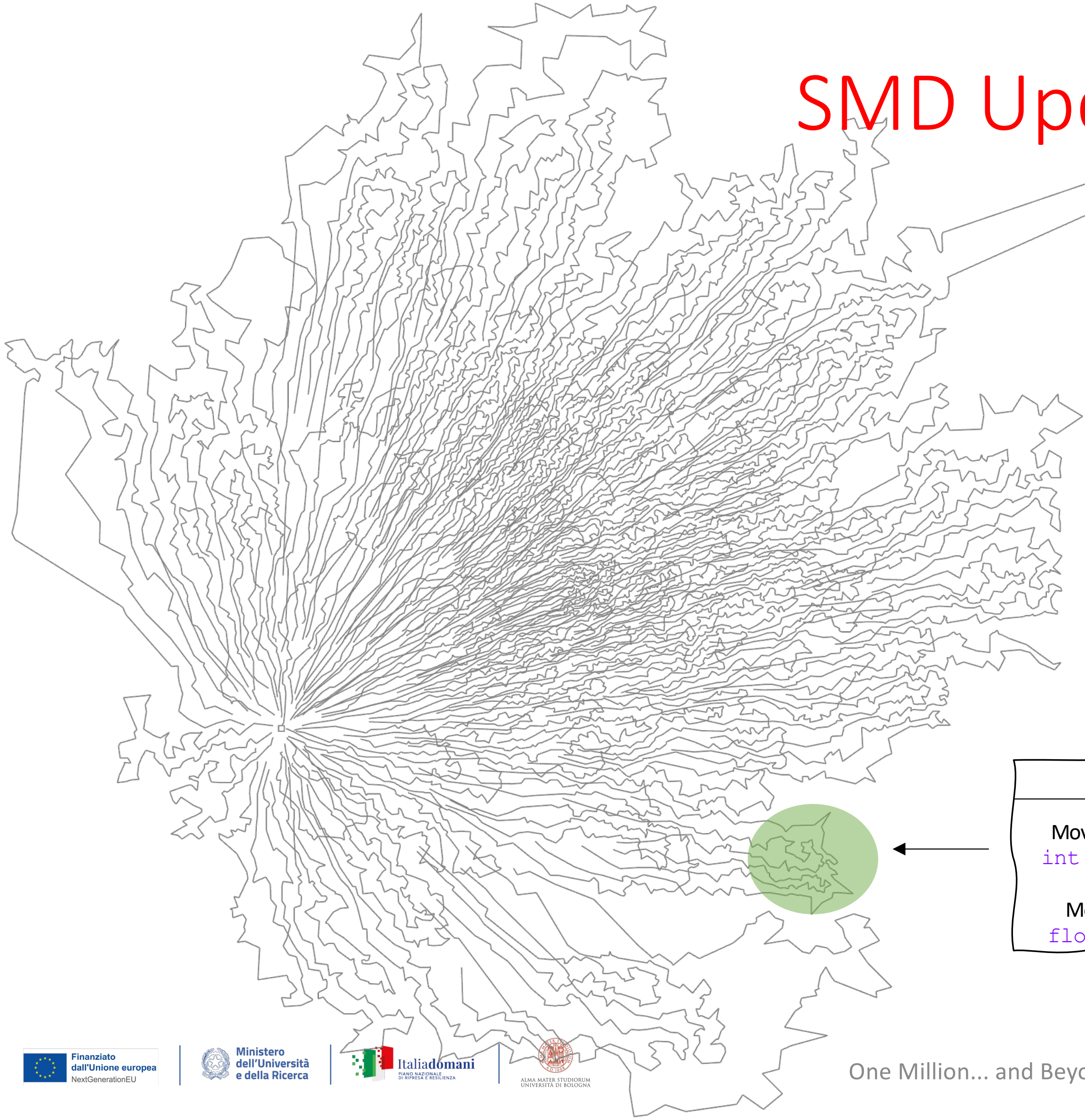
SMD Execution

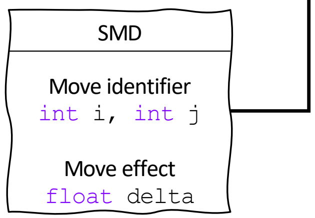
The move associated with the selected SMD is applied to the current solution

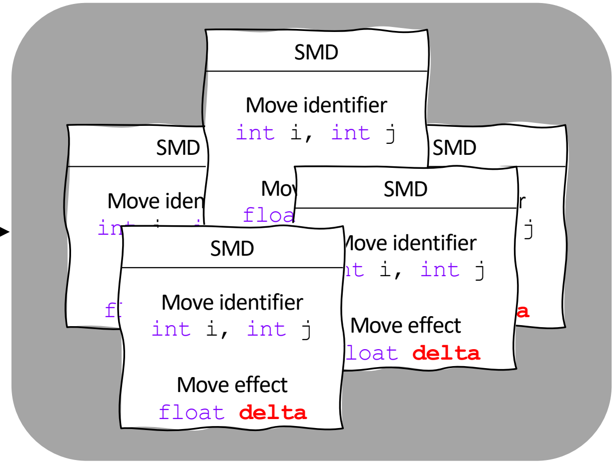
Local search operators perform local changes thus **most** of the SMDs will still hold a correct `delta` value after the move is executed



SMD Update



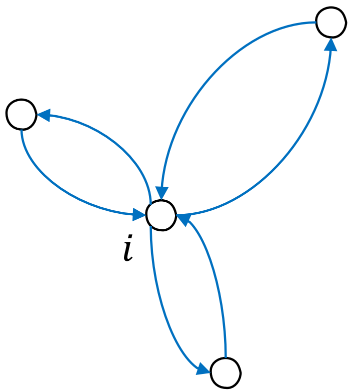




A move `(i,j)` of operator `XYZ` requires the update of the `delta` value of fixed set of SMDs

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 ∈ 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**

$L(T)$ | In use | Available but **not** in use

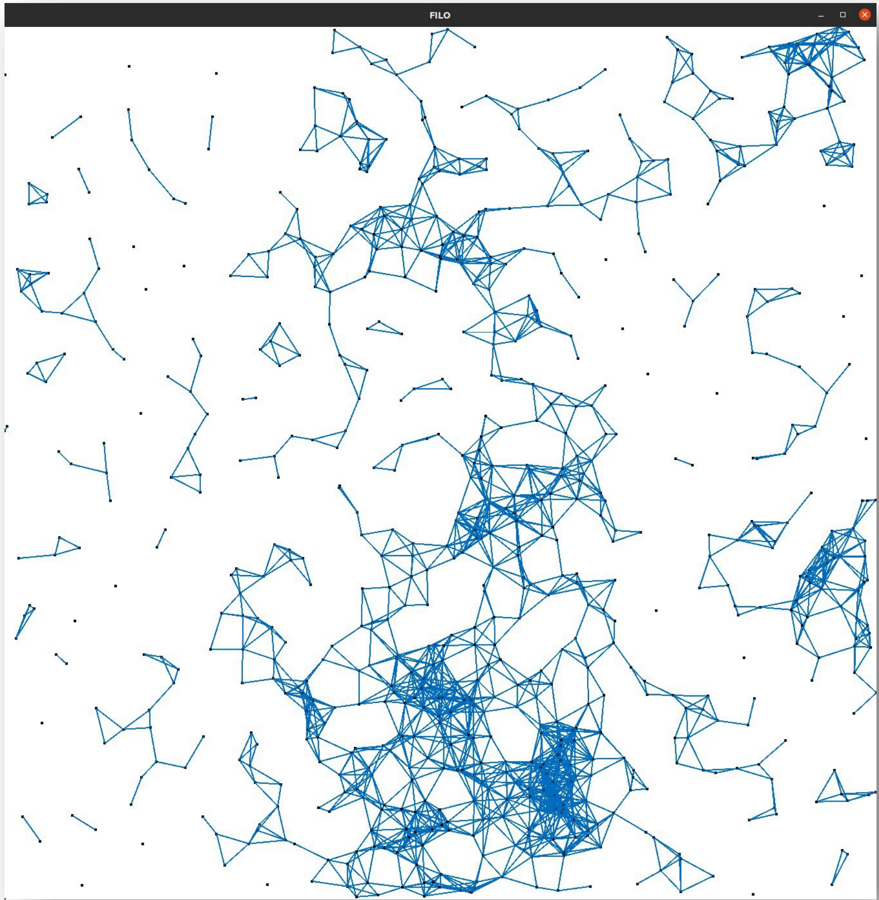
γ **sparsification factor**

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

Update rule

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

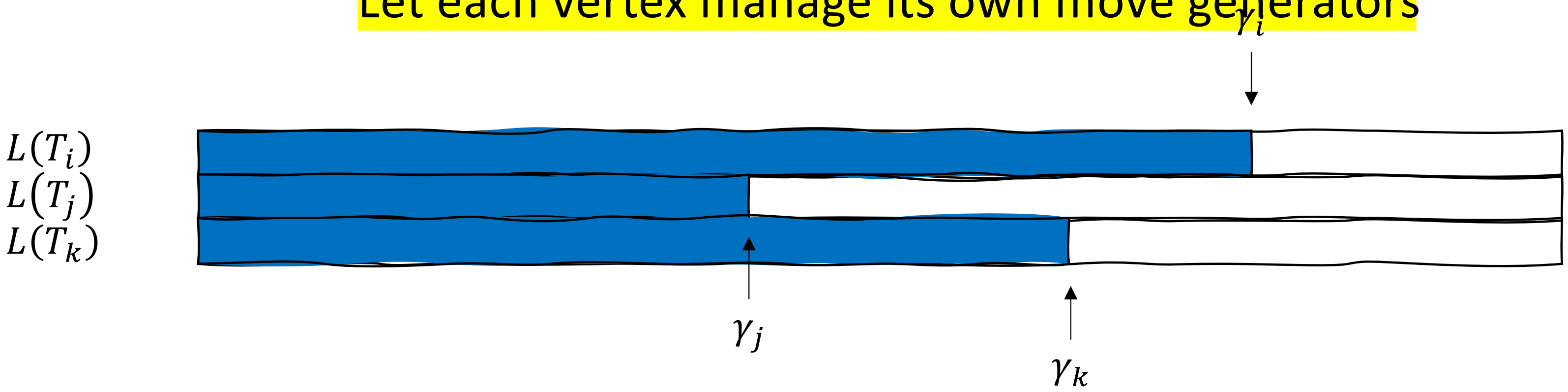
Dynamic GNs



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

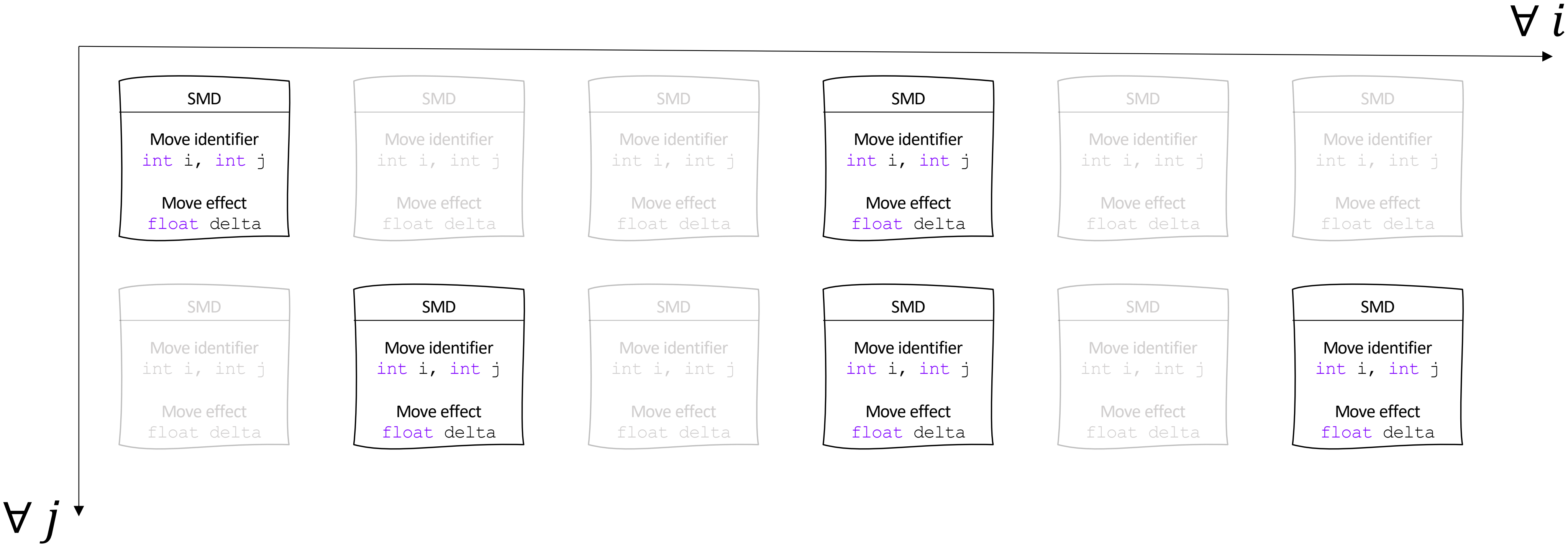
- A minimum number of move generators is guaranteed per vertex
- Tailored intensification: move generators are increased only for areas that more likely require a stronger intensification
- Intensification is globally increased at a slower rate
  - faster local search for more optimization iterations

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)

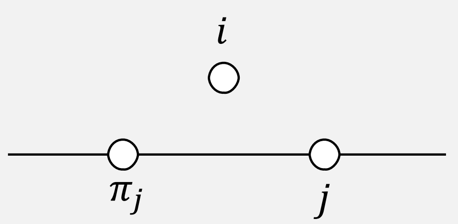
A granular neighborhoods counterpart for vertices

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

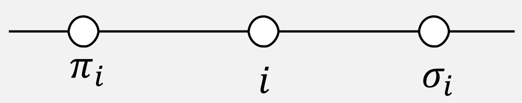
INTERESTING

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

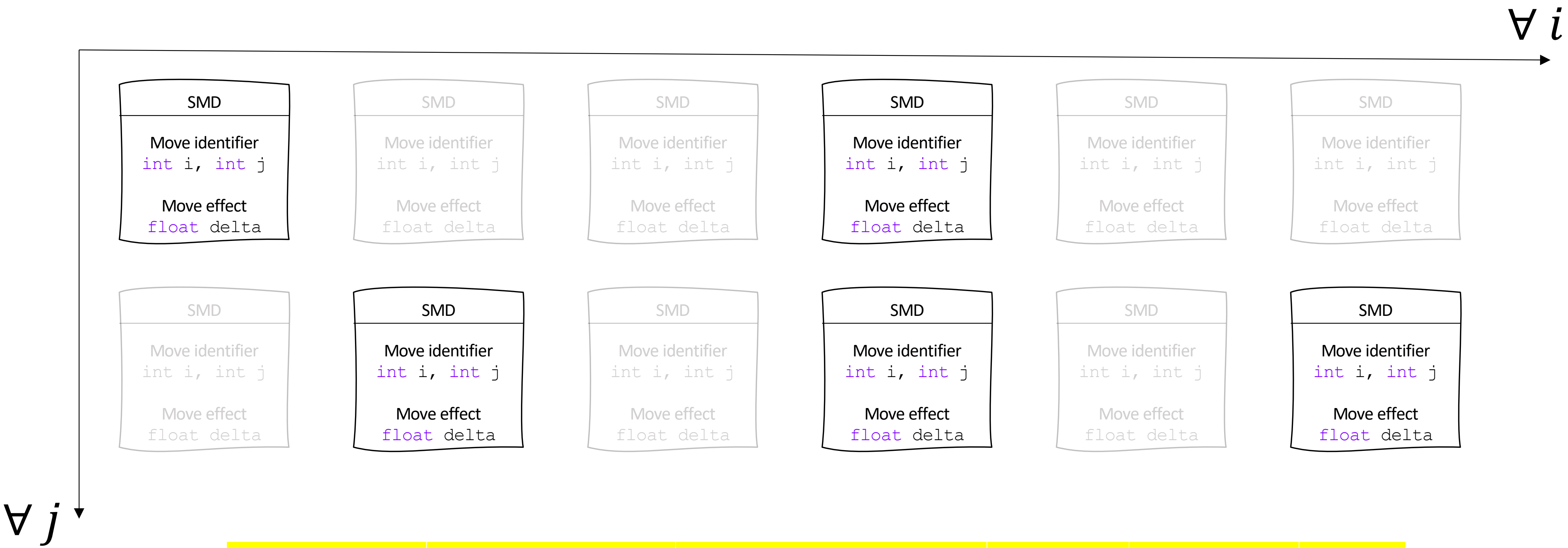


RECENTLY

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

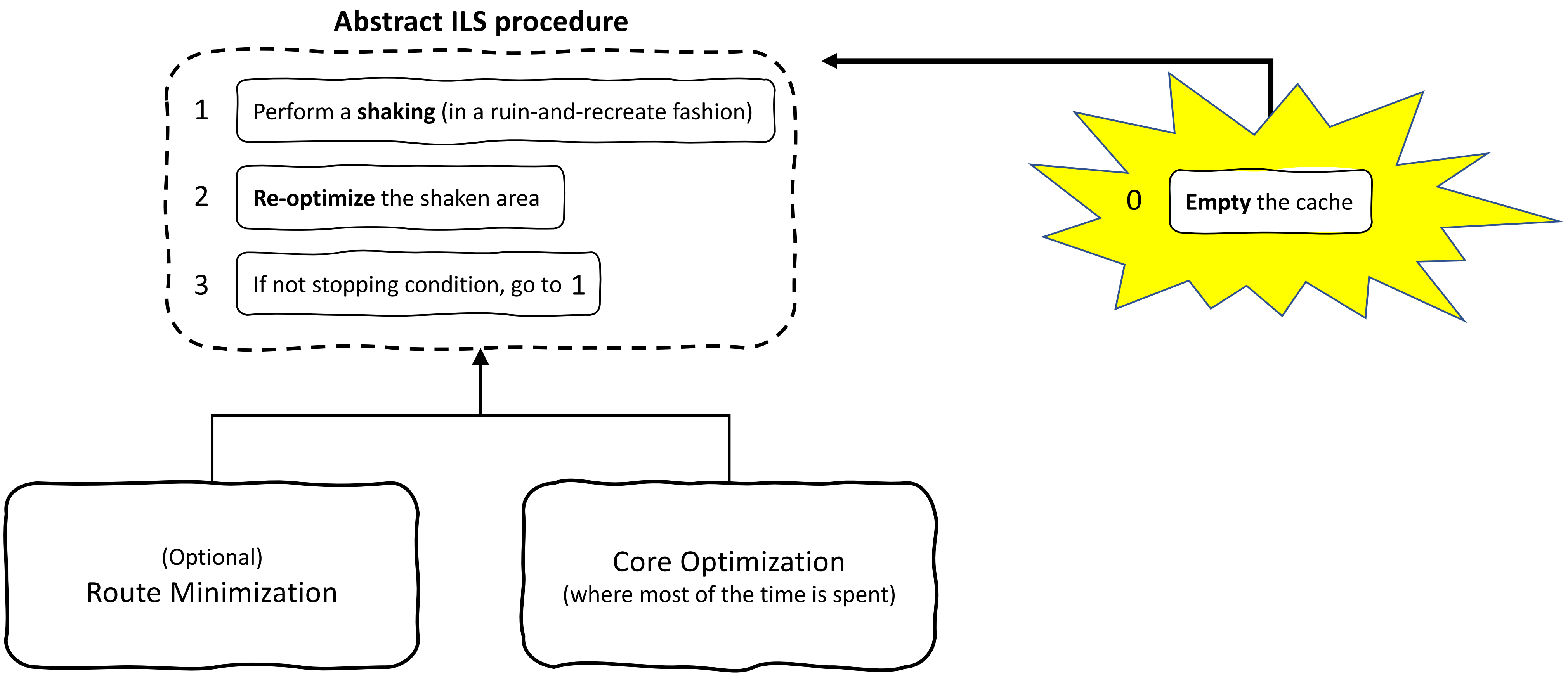
SVC to Restricted SMD Initialization

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



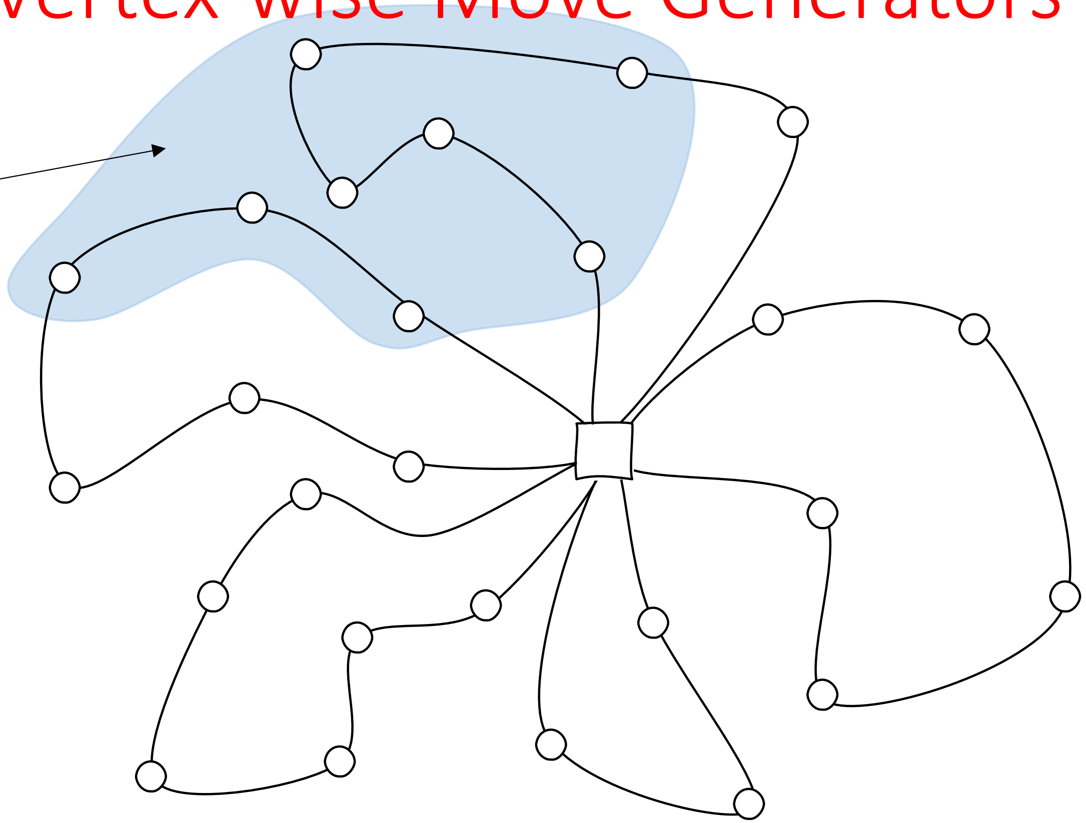
Subsequent SMD Updates may incrementally include additional SMDs

SVC to Focus Local Search Applications



SVC to Update Vertex-wise Move Generators

Cached vertices
after HRVND execution



Update rule

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

Core Optimization

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

Loop

2 Perform a **random walk** ruin-and-recreate application on S to obtain $\hat{S}$

3 Optimize $\hat{S}$ using the LS engine to obtain S'
- **If** $cost(S') < cost(S^*)$ | $S^* = S'$ | Reset $\overline{\gamma}$
- Otherwise | Update $\overline{\gamma}$

4 Update $\overline{\omega}$

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

6 $t = c \cdot t$

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 — $i = j$

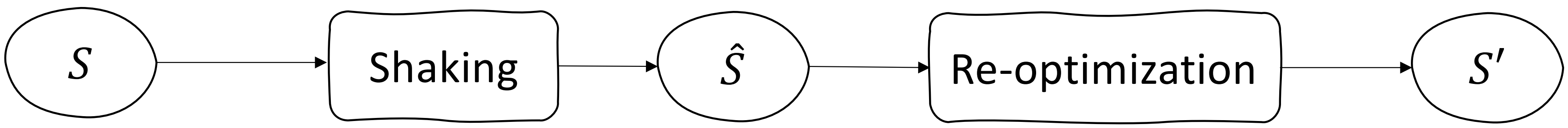
For each removed customer i

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 $\bar{\boldsymbol{\omega}}$

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

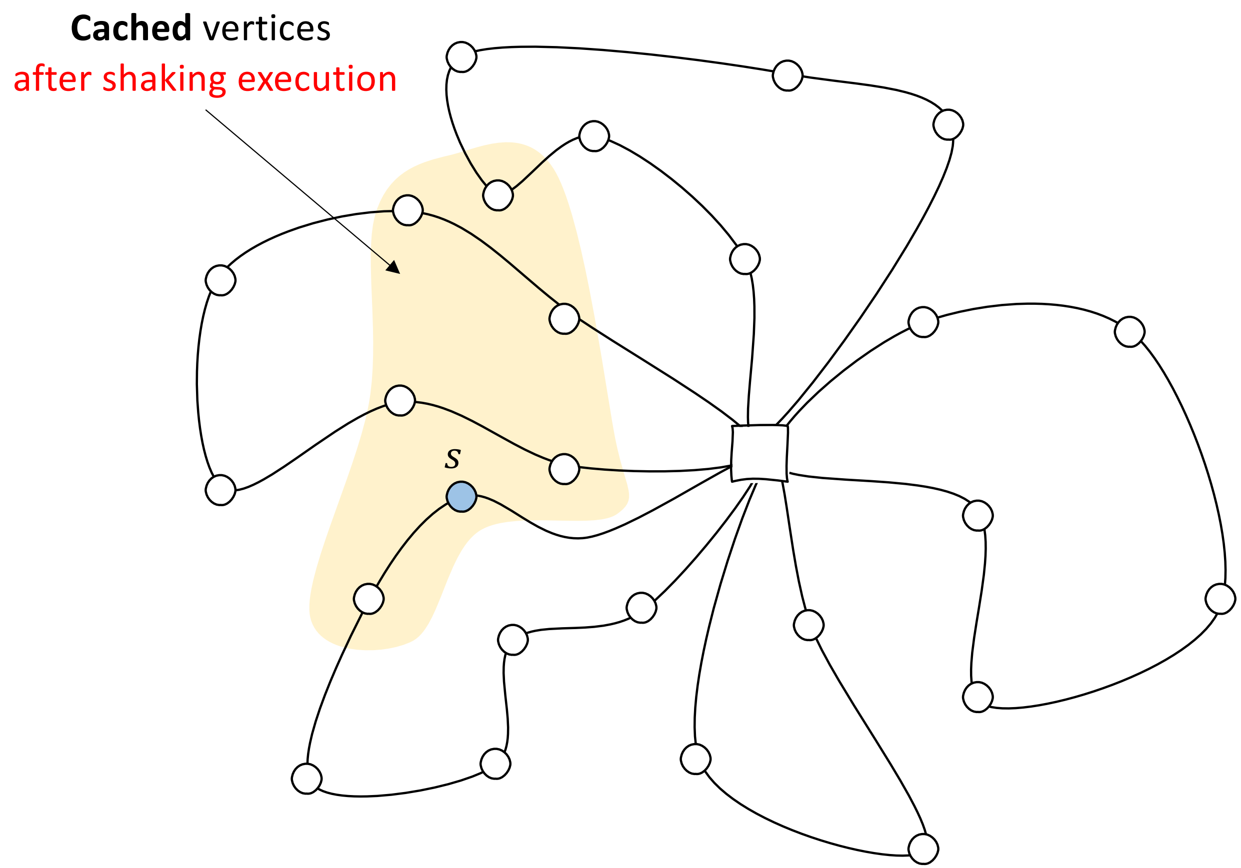


Random walk of length ω_s
from a seed customer s

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

A declarative selection of shaking parameters $\bar{\boldsymbol{\omega}}$

SVC to Update Shaking Parameters



Update rule

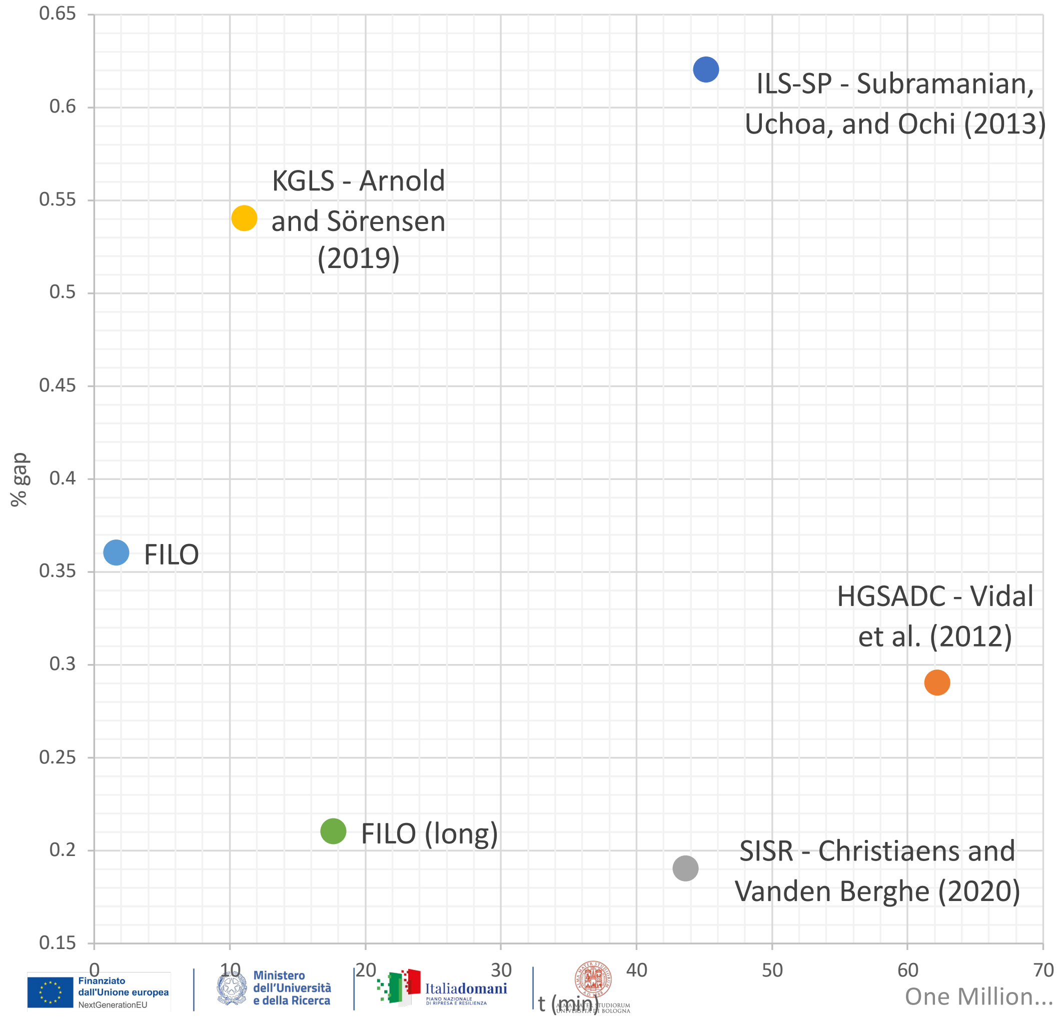
$$\begin{cases} \omega_i = \omega_i - 1 & \text{if SHAKING TOO STRONG} \\ \omega_i = \omega_i + 1 & \text{if SHAKING TOO MILD} \\ \text{Randomly increase or decrease } \omega_i & \text{if SHAKING OK} \end{cases}$$

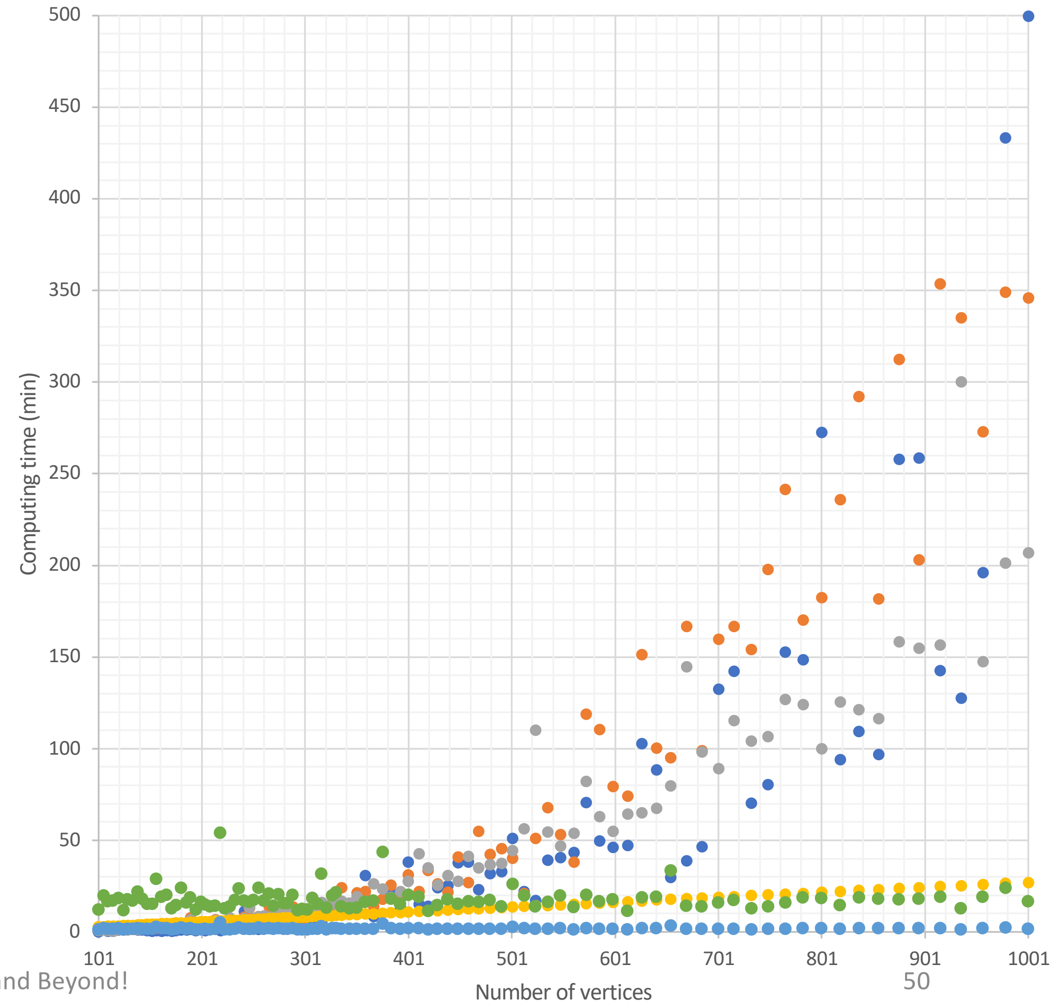
$i \in \overline{V_{\hat{S}}}$

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)**

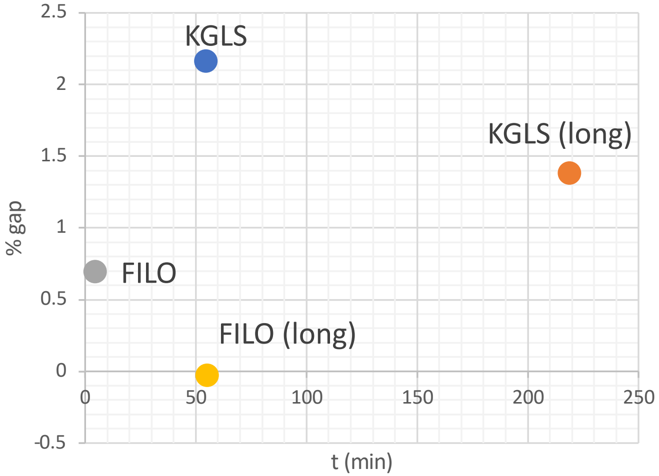
X: Uchoa et al. (2017)



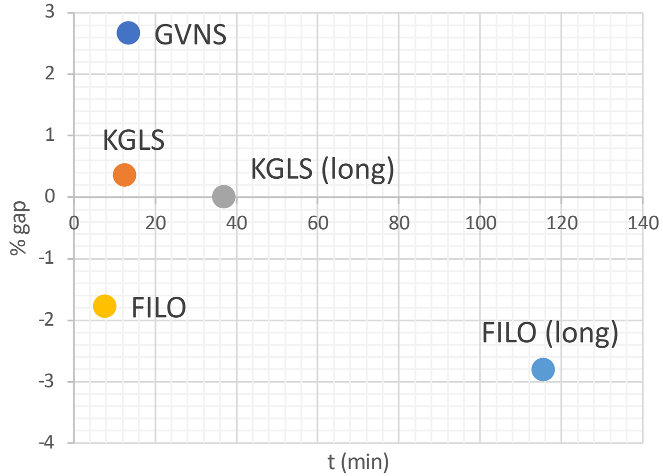


Very large instances

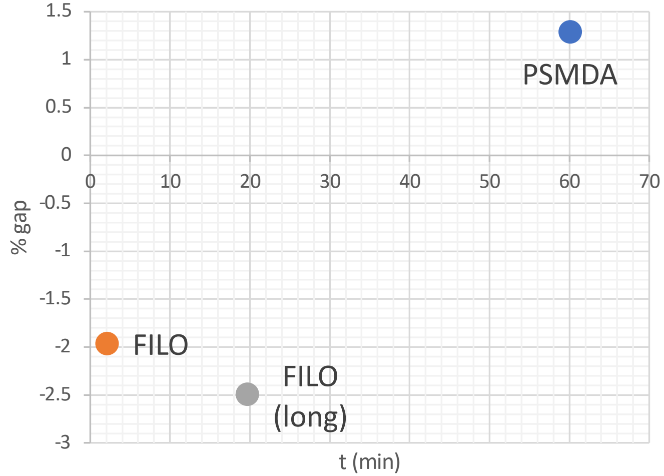
B (3K – 30K)
Arnold, Gendreau, and Sörensen (2019)



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)

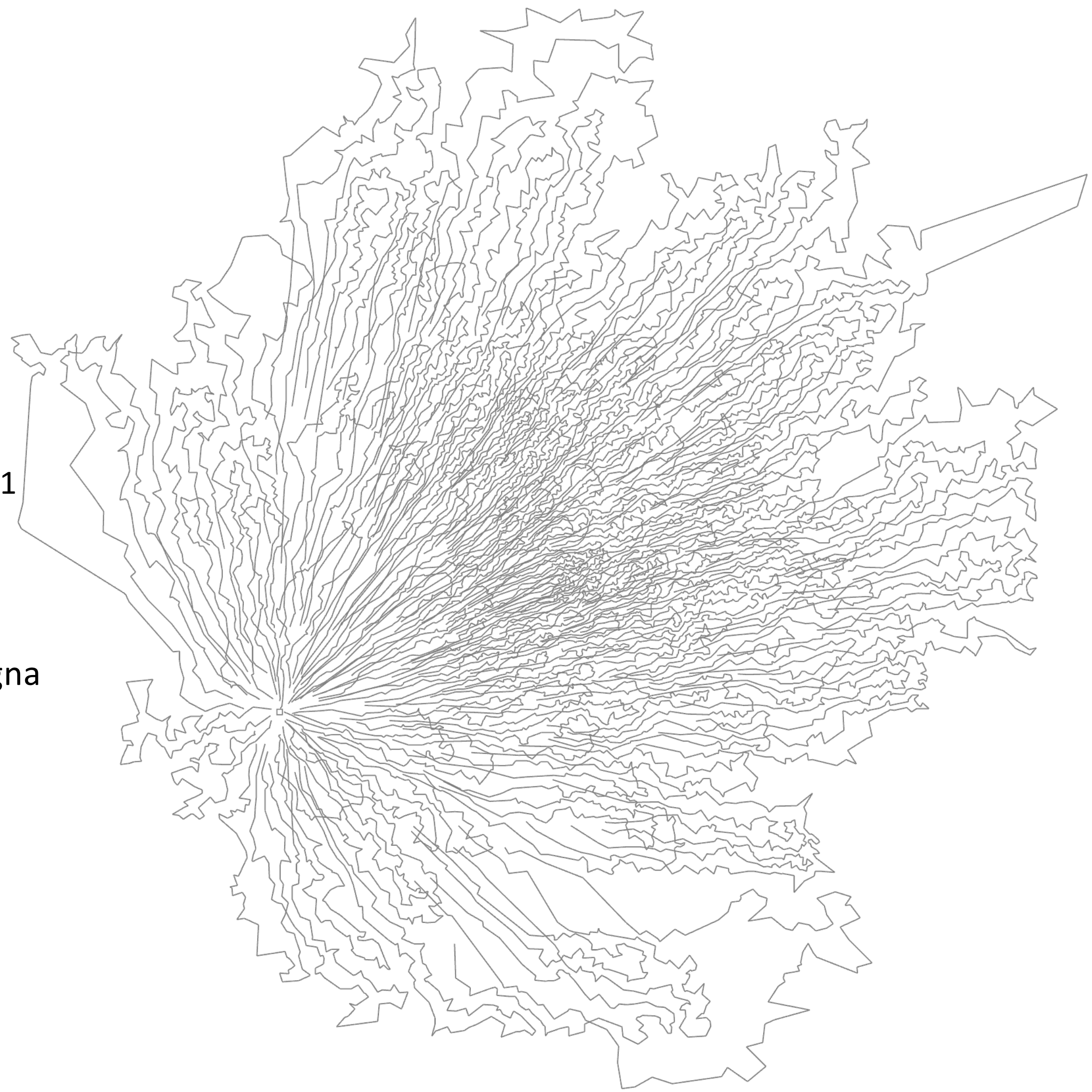
FSP4D

FILO + SP for DIMACS

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

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

VRP
DIMACS Implementation Challenge



MAJOR CHANGES WITH RESPECT TO FILO

- Revamp of the LS engine to improve Data Locality
- Added two 2-opt based chained operators in the 2nd tier of the LS engine
- Multistart with additional sophisticated Set Partitioning-based polishing of solutions
  - Main objective minimizing the Primal Integral measure

SET PARTITIONING PHASE (1/2)

Set Partitioning Problem
Given a set of columns, select a subset that cover all the rows once and minimize the cost sum

As to VRP
- Columns are feasible routes
- Column cost is the route length
- Rows are customers

(Restricted) Set partitioning formulation of the VRP
Given a (restricted) set of routes, select a subset that visits all the customer once and minimize the cost sum

$$\min \sum_{p \in \Omega} c_p \theta_p$$

$$\sum_{p \in \Omega} \theta_p = k$$

$$\sum_{p \in \Omega_i} \theta_p = 1, \quad \forall i \in N$$

$$\theta_p \in \{0, 1\}$$

SET PARTITIONING PHASE (2/2)

Can be used as

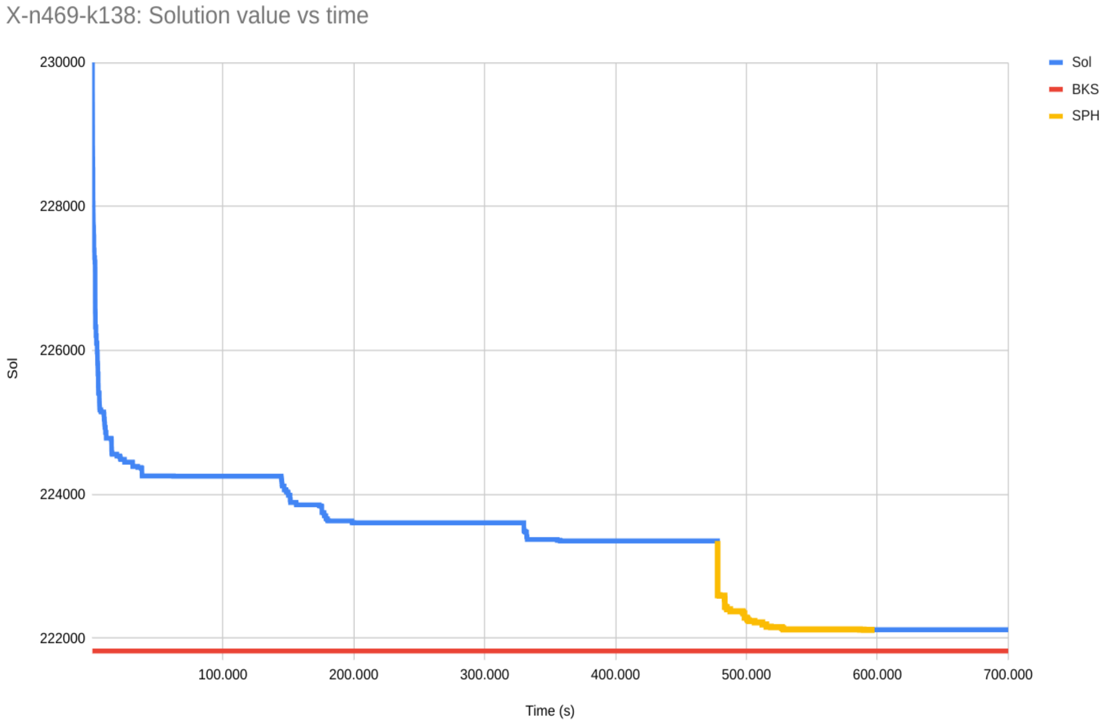
- Short periodic phase that "merges" routes found in independent runs of FILO
- Post-optimization phase at the very end

Pros

- Requires very little time
- Effective with some difficult instances where FILO struggles in combining routes together

Con

- Often improvements are small
- Work best after multiple independent runs of FILO



Achieved results

- Ranking was based on Primal Integral of solution, favoring methods which find quickly good solutions.
- Instances had n≤1000 (relatively small for FILO)
- FSP4D ranked overall 6th (3rd on the large instances 300≤n≤1000)
- In the preliminary phase FSP4D ranked (by far) first on Belgium instances
- Solver Alkaid-X, which ranked 1st hybridized FILO with the HGS algorithm by Vidal et al.

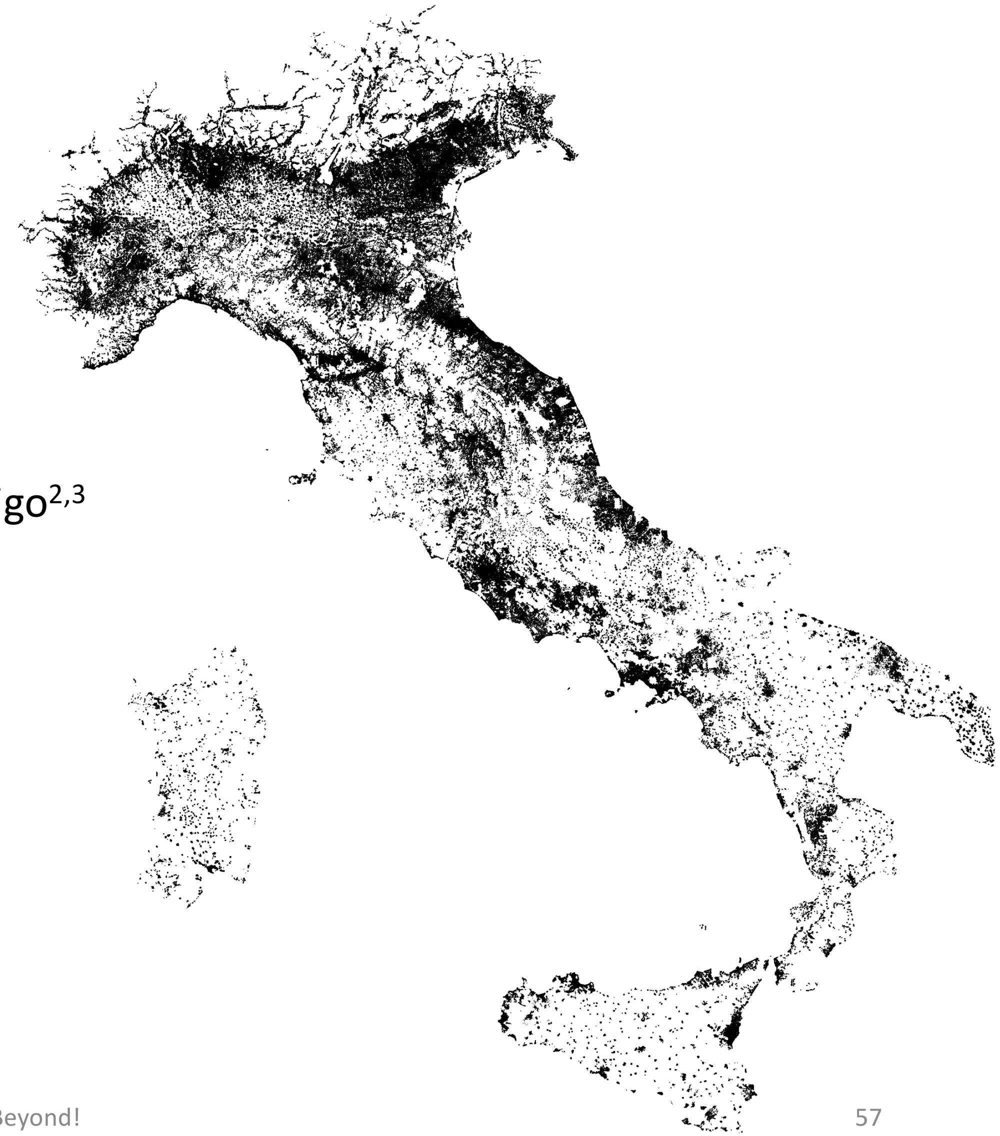
FSPD

An Efficient Heuristic for Very Large-Scale Vehicle Routing Problems with Simultanueous Pickup and Deliverly

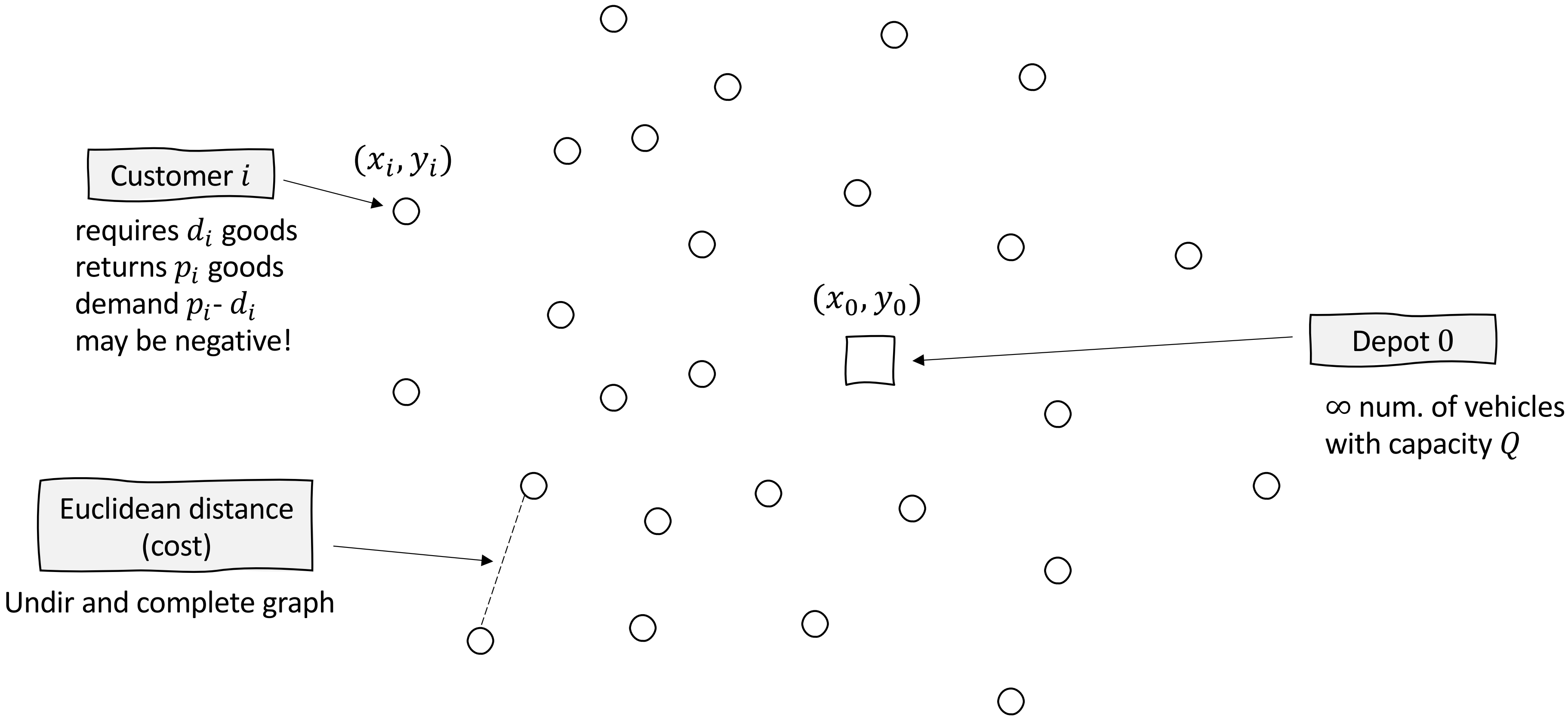
F.Cavaliere[2], L.Accorsi[1], D.Lagana[4], R.Musmanno[4] and D. Vigo[2,3]

[1] Google
[2] DEI «Guglielmo Marconi», University of Bologna
[3] CIRI ICT, University of Bologna
[4] DIMEG, University of Calabria

Submitted, 2024 🤞

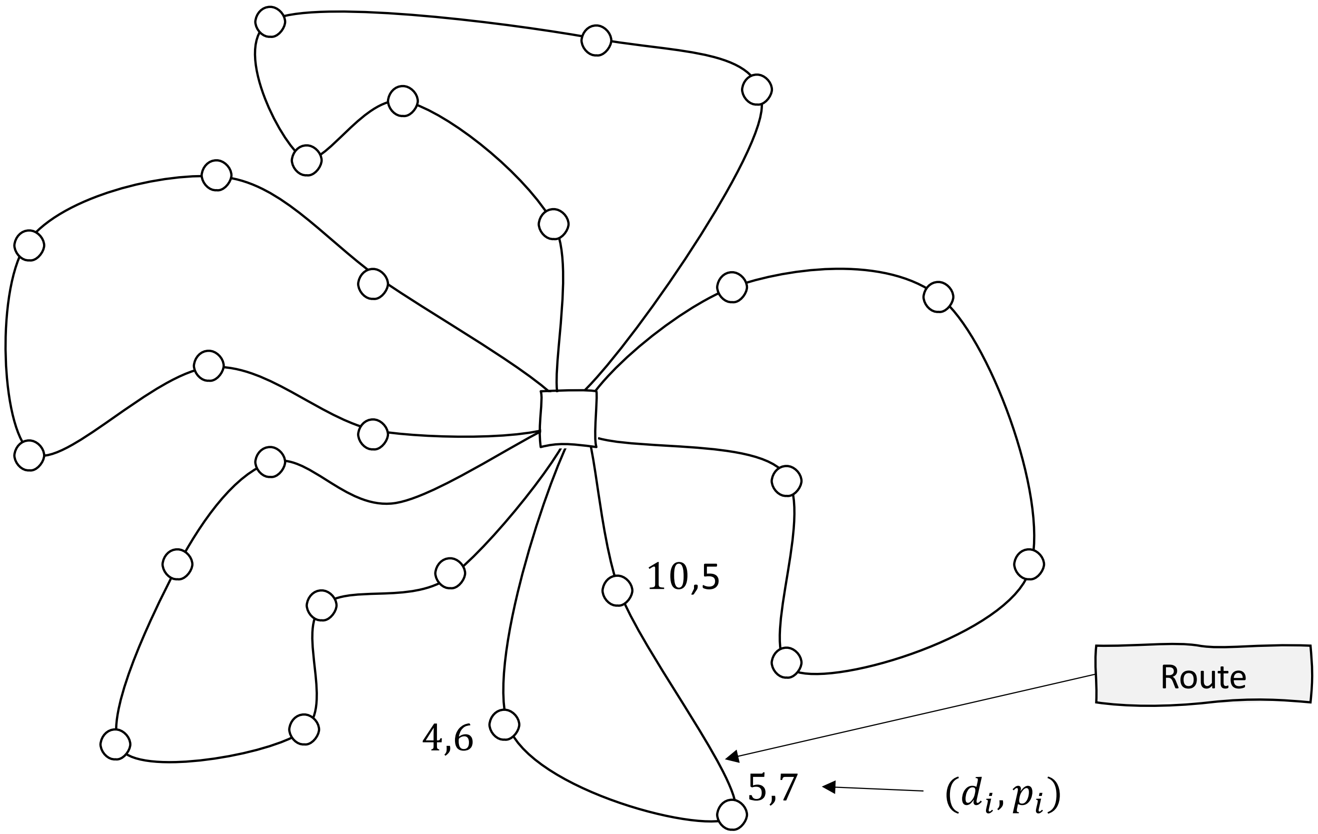


VRP with Simultaneous P&D (VRPSPD) Instance



VRP with Simultaneous P&D (VRPSPD) solution

Along a route the load on the vehicle does not monotonically decrease as in CVRP !



Current State-of-the-Art Methods

- Vidal et al. (2012): Hybrid Genetic Algorithm
- Subramanian, Ochi, Uchoa (2013): ILS+ SP
- Hof and Schneider (2019): ALNS+Path Relinking
- Christiaens and Vande Berghe (2020): Ruin and recreate based o string removal and insertion
- …

- Popular Benchmark Set by Sahly&Nagy (1999) with n=50:199

The challenge

- Extending FILO to handle additional constraints (in general, but to be tested on VRPSPD) ➔FSPD framework !

- Main issue:
  - re-engineering LS engine to handle general feasibility check
  - Solution: extending FILO to incorporate Resource Extension Functions for feasibility check

Resource Extension Functions (REFs)

- Proposed by Desaulniers et al (1998), Irnich (2008)
  - Each route may be partitioned in segments
  - Each segment is associated to a set of R resources so that feasibility check can be done in O(R)
  - Given two segments a REF returns the feasibility of a concatenation of them
- Example CVRP: R is demand-sum of the segment
  - given s_1, R_1 and s_2, R_2, for $s_1 \oplus s_2$ we have $R_{s_1 \oplus s_2} = R_1 + R_2$
-

Resource Extension Functions (REFs)

- For VRPSPD we need 3 resources
  - M: maximum load;
  - P: pickup-sum;
  - D: delivery-sum
- s_1, M_1, P_1, D_1 and s_2, M_2, P_2, D_2, for $s_3 = s_1 \oplus s_2$ we have
  - $M_3 = \max\{M_1 + D_2, M_2 + P_1\}$
  - $P_3 = P_1 + P_2$
  - $D_3 = D_1 + D_2$
- LS operators must be reimplemented to handle REFs
- Several implementation tricks must be employed to control memory and time (not all possible segments may be stored)

The challenge (cont.d)

- Minor issues:

  1) Adapt R&R to handle additional constraints when removing and adding customers to a route
     - Solution: careful implementation of general insertion and removal and resulting resource computation

  2) Generating a feasible initial solution
     - can be obtained by adapting the C&W and using the general removal and insertion functions

  3) Keep memory requirement controlled due to resource storage

- Testing the scalability of the approach on constrained VRPs
  - generate new benchmark instances with 10^3-10^4 customers

Computational results

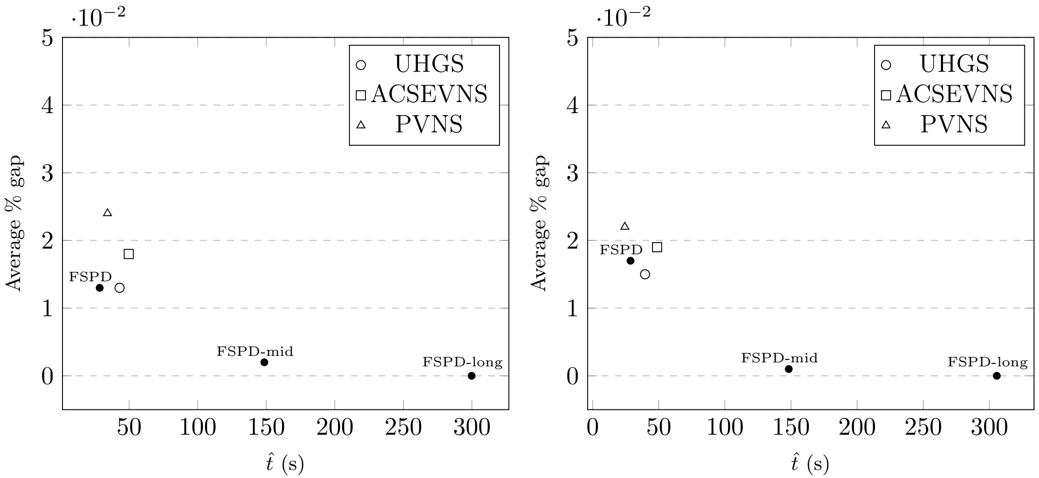
- Three versions of FSPD
  - FSPD $100K$ core optimization iterations
  - FSPD (mid) $500K$ core optimization iterations
  - FSPD (long) $1M$ core optimization iterations
- On *standard* instances
  - **CMTX, CMTY** dataset by **Salhy and Nagy (1999)** (50-199 cust.)
    - some algorithms were only tested on the first 7 instances of each dataset
  - **D** dataset by **Dethloff (2001)** (50 customers)
  - **M** dataset by **Montané and Galvao (2006)** (100-400 customers)
- On very large-scale instance (by adapting CVRP instances)
  - **X** dataset **by Uchoa et al (2017)** (100-1000 customers)
  - **XXL** dataset by **Arnold, Gendreau, and Sörensen (2019)** (3K-30K customers)

Competitors

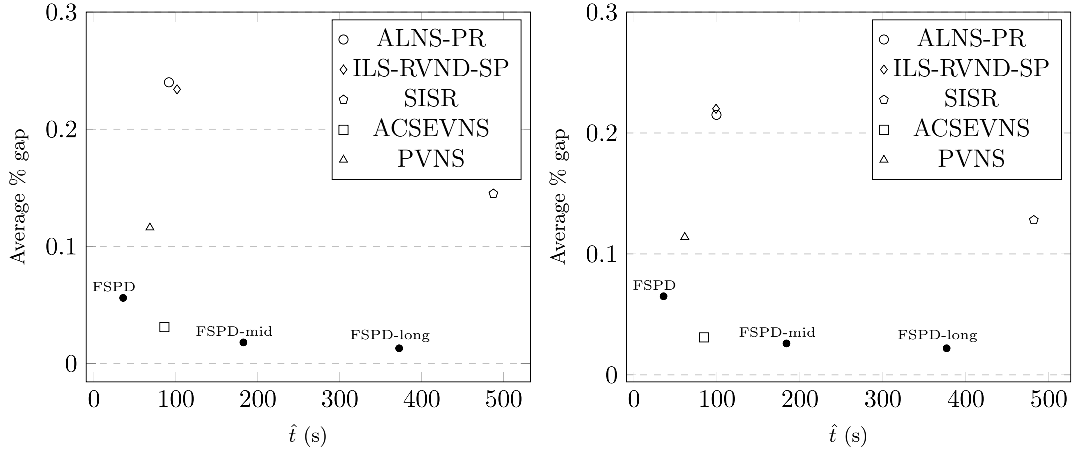
- ALNS-PR: the hybrid algorithm combining adaptive large neighborhood search (ALS) and path relinking of Hof and Schneider (2019).
- ILS-RVND-SP: the ILS heuristic of Subramanian, Uchoa, and Ochi (2013).
- SISR: the ruin-and-recreate algorithm of Christiaens and Vanden Berghe (2020b).
- UHGS: the population-based method of Vidal et al. (2014).
- h_PSO: the hybrid discrete particle swarm optimization of Goksal, Karaoglan, and Altiparmak (2013).
- ACSEVNS: the hybrid heuristic based on ant colony and variable neighborhood search of Kalayci and Kaya (2016).
- PVNS: the perturbation-based variable neighborhood search algorithm of Polat et al. (2015).
  - Note that, for this algorithm, the computing times reported are those of the best out of 10 runs (in terms of solution quality).
- ILS-RVND-TA: the hybrid ILS of Öztaş and Tuş (2022).
- VLBR: the adaptive memory approach of Zachariadis, Tarantilis, and Kiranoudis (2010).
  - Note that, for this algorithm, the computing times reported are those to reach the best solution and not the total ones.

Results on CMTX and CMTY

first 7 instances



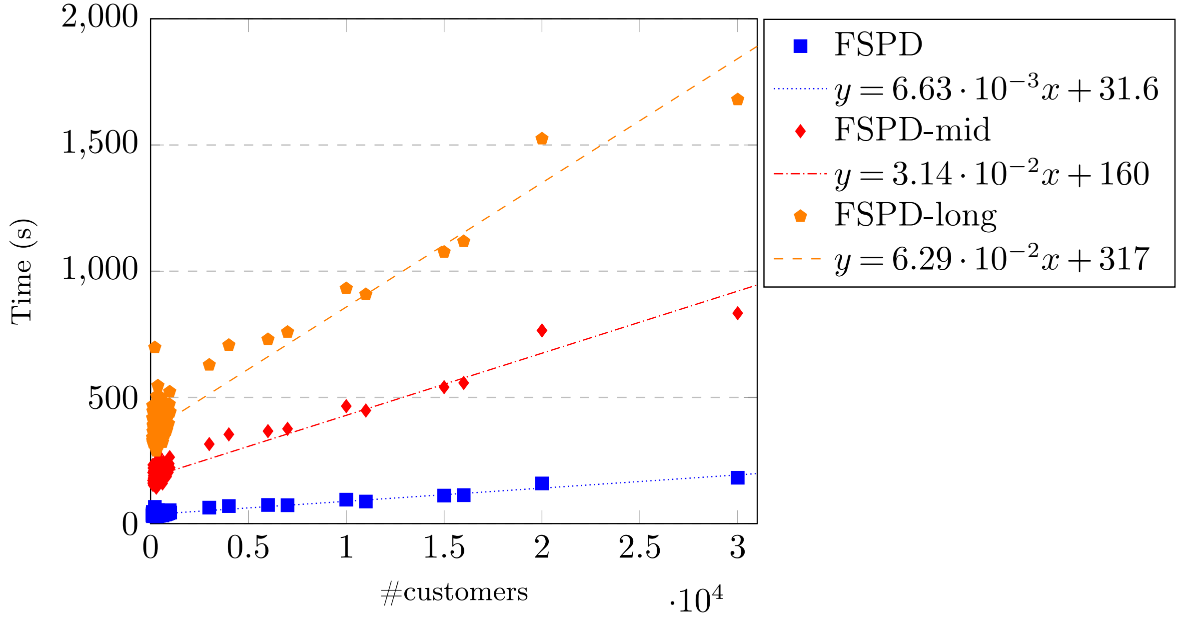
all instances



similar results on D and M datasets and also on VRPMPD

Results on X and XXL instances

- Checking the linear scaling of FSPD



Results on X and XXL instances

- Good improvement when increasing the n. of iterations but still limited computing time

Table 7 Results on the new large-scale VRPSPD XX, XY instances.

Algorithm	XX			XY		
	Avg	Time*	Time	Avg	Time*	Time
FSPD	0.769	28.905	36.019	0.776	28.359	35.922
FSPD-mid	0.365	135.261	180.511	0.382	134.260	179.671
FSPD-long	0.279	267.001	361.192	0.253	261.489	358.289

Table 8 Results on the new very large-scale VRPSPD XXLX, XXLY instances.

Algorithm	XXLX			XXLY		
	Avg	Time*	Time	Avg	Time*	Time
FSPD	2.814	104.867	105.059	2.741	104.007	104.221
FSPD-mid	0.936	516.885	518.687	0.897	511.611	513.295
FSPD-long	0.305	1040.386	1045.203	0.226	1025.770	1028.897

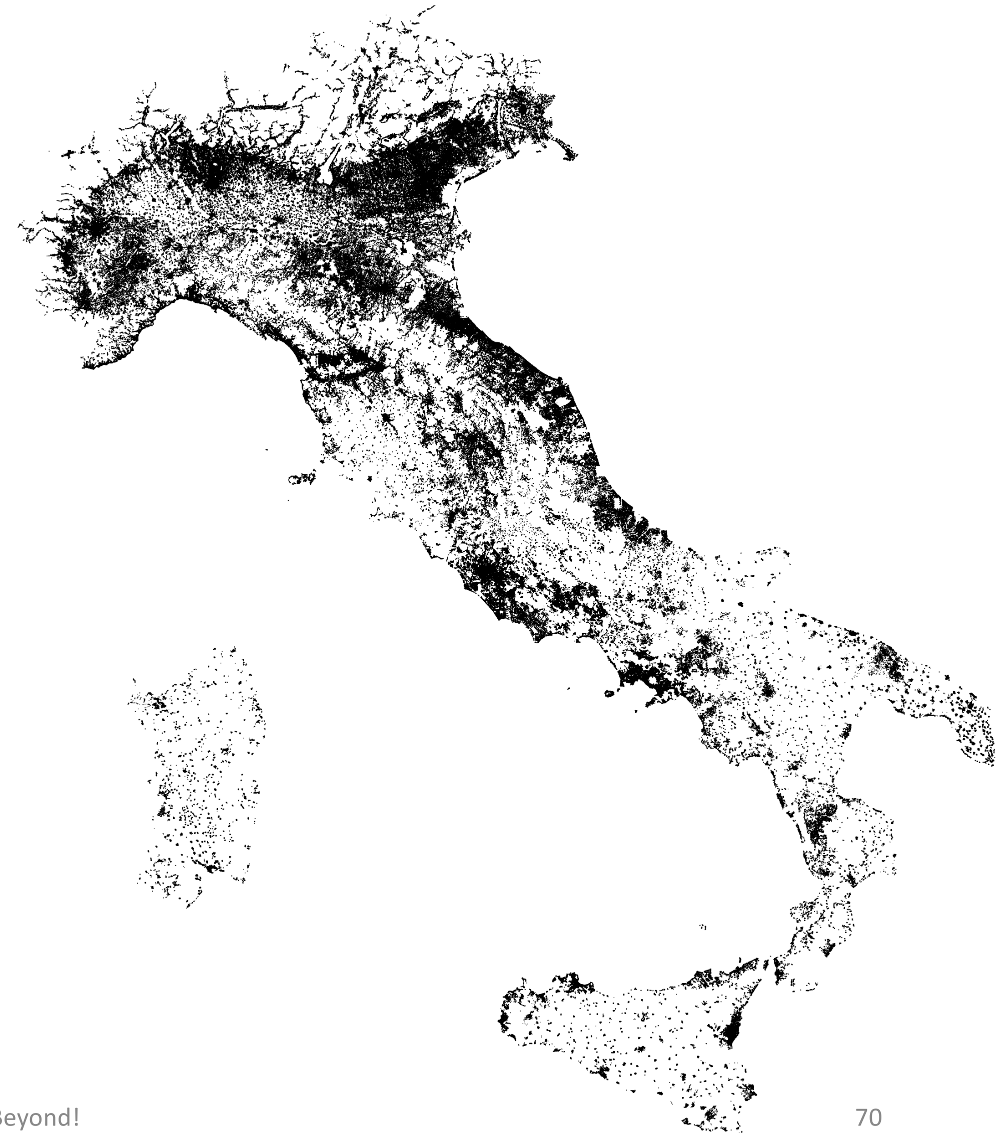
FILO2

Routing one million customers in a handful of minutes

Luca Accorsi[1] and Daniele Vigo[2,3]

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

Computers & Operations Research, 2024



Motivation

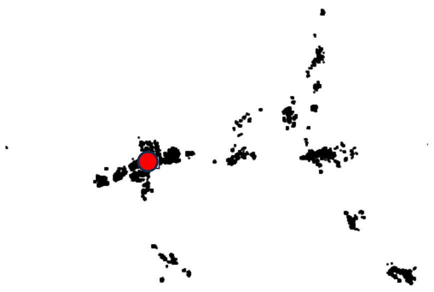
- Funny research exercise
- Challenging target
  - Push the limits of CVRP
  - Inspire new research on efficient and effective algorithms

- Make all Italian regions known around the world!

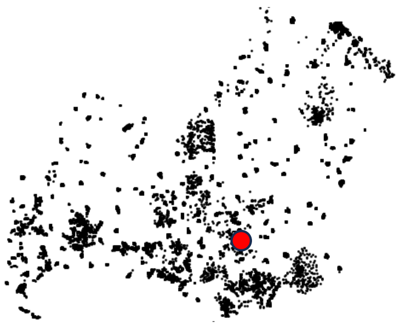
The Datasets

- 20 XXL instances having between 20k-1M customers built similarly to the Belgium instances
  - Customer demand in [1, 3]
  - Vehicle capacity 50, 150, 200
  - Half instances require relatively short routes, half longer ones
- 2D vertex coordinates coming from real addresses in Italian regions
  - OpenAddresses
  - Different layouts and customer densities following actual cities distribution
  - Depot in the regional capital (internal, eccentric, frontier…)

The Datasets

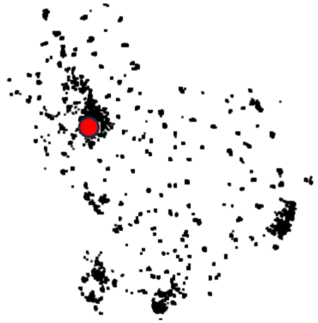


Valle d'Aosta (20k)



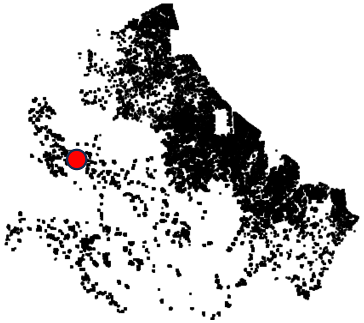
Molise (50k)

Trentino-Alto Adige (100k)



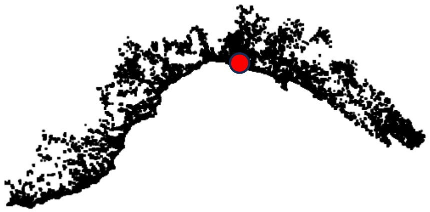
Basilicata (150k)

Umbria (200k)

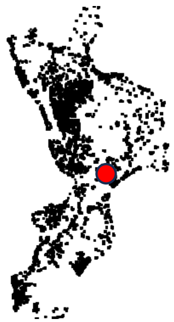


Abruzzo (250k)

Friuli-Venezia Giulia (300k)



Liguria (320k)



Calabria (380k)

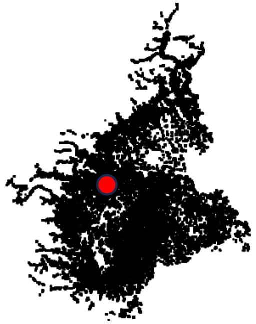
Marche (420k)

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani | Alma Mater Studiorum Università di Bologna

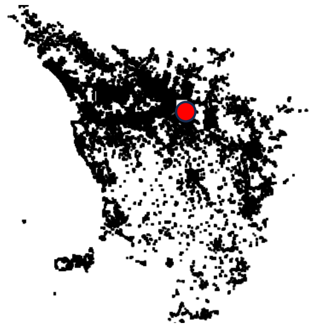
The Datasets

Sardegna (470k)

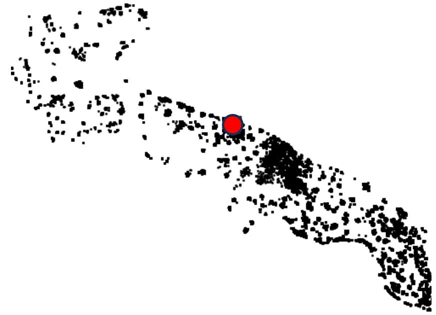
Campania (500k)



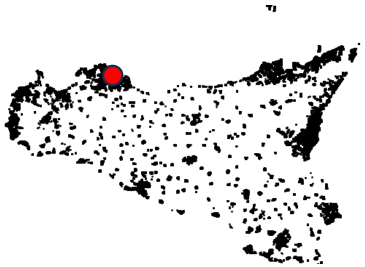
Piemonte (600k)



Toscana (700k)

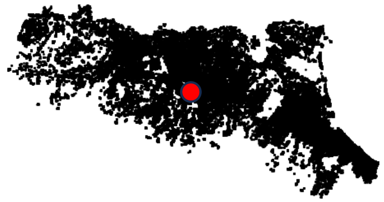


Puglia (750k)



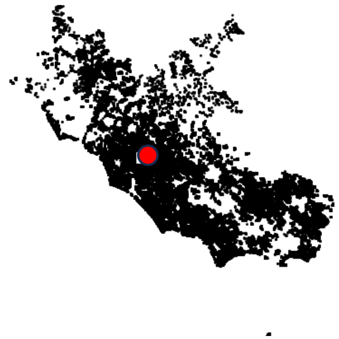
Sicilia (800k)

Veneto (850k)



Emilia-Romagna (900k)

Lombardia (950k)



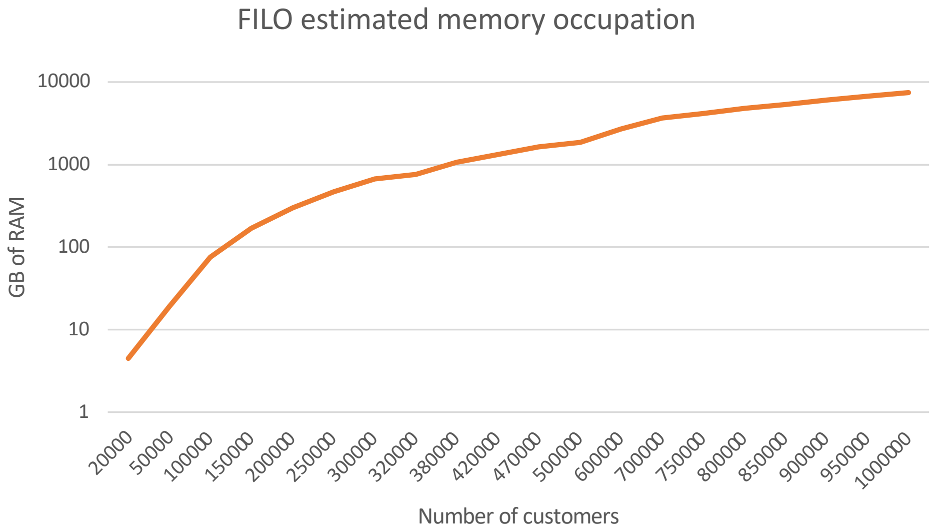
Lazio (1M)

Goal

- Show that granular neighborhoods, static move descriptors, and selective vertex caching are already powerful enough techniques making FILO scale to huge-scale sizes

Goal

- Show that granular neighborhoods, static move descriptors, and selective vertex caching are already powerful enough techniques that makes FILO scale to these sizes
- But first... let's develop FILO2 to improve certain FILO aspects

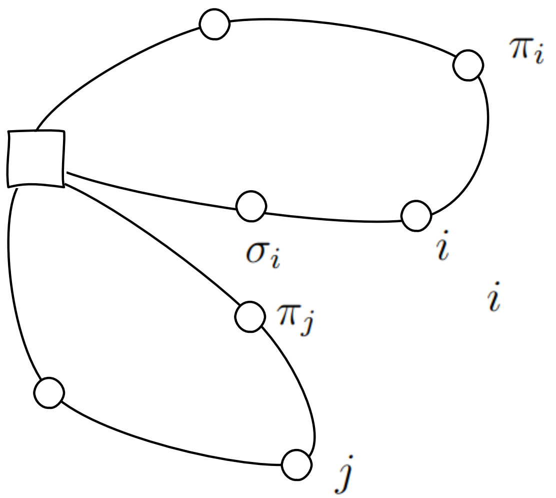


1st Challenge: Memory Requirements

- Memory-demanding data structures (~quadratic in n)
  - Cost matrix
    - Direct access to arc costs
    - Necessary to evaluate solution changes
  - Neighbors lists
    - Restricted Savings algorithm
    - Ruin step
    - Move generators definition
- Both are critical for the main algorithm procedures

Cost Matrix

- The explicit cost matrix is removed and replaced with
  - On-demand computation of arc costs from coordinates
  - Storage of arc costs in the current solution into the solution data structure
  - Storage of arc costs in move generators data structures



Relocation of i before j

$$c_{\pi_i \sigma_i} - c_{\pi_i i} - c_{i \sigma_i} + c_{\pi_j i} + c_{ij} - c_{\pi_j j}$$

Only 2 out of 6 costs are computed on-demand in practice

On-demand vs Cached costs

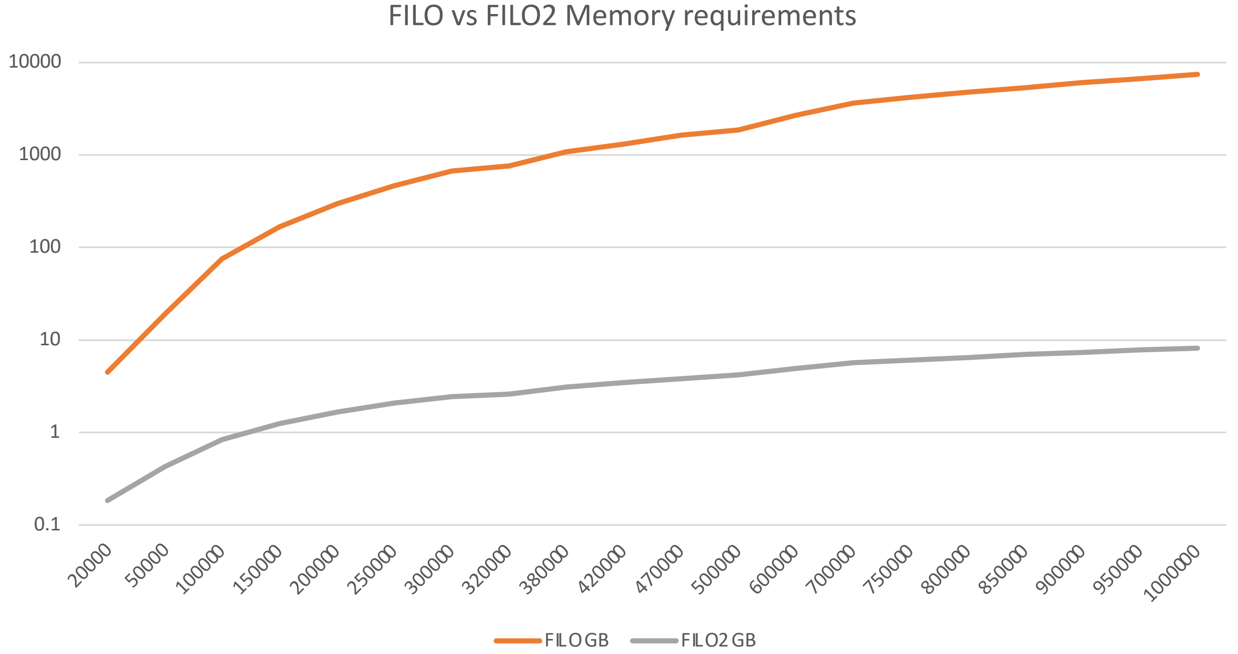
- Cache proposed by Bentley (1990) for the TSP
  - Effective for algorithms showing a great locality in cost computation
  - FILO definitely has this property (see hit ratio)
  - However, Cache management overhead does not pay-off

Configuration	Time percentage increase wrt baseline	Cache hit ratio
On-demand+	Baseline	
On-demand	10%	
Cached+	13%	84%
Cached	27%	91%

+ Some costs are retrieved in constant time from solution and move generators

1st Challenge: Memory Requirements

- FILO2 uses the on-demand+ strategy
  - We approach an instance with 1M customers on an ordinary laptop!



Neighbors Lists

- We no longer compute exhaustive lists of neighbors
  - We only compute n_{nn} of them
- This can be done efficiently in a preprocessing phase by using a kd-tree built on top of vertex coordinates
  - Build tree: $O(n \log n)$
  - Find n_{nn} neighbors: $O(n_{nn} \log n)$
  - Compute neighbors lists: $O(n \; n_{nn} \log n)$
- Neighbors of different vertices are independent
  - Easy parallelization!
  - But in this work we sticked to the classical single-thread setting typical of this type of OR works

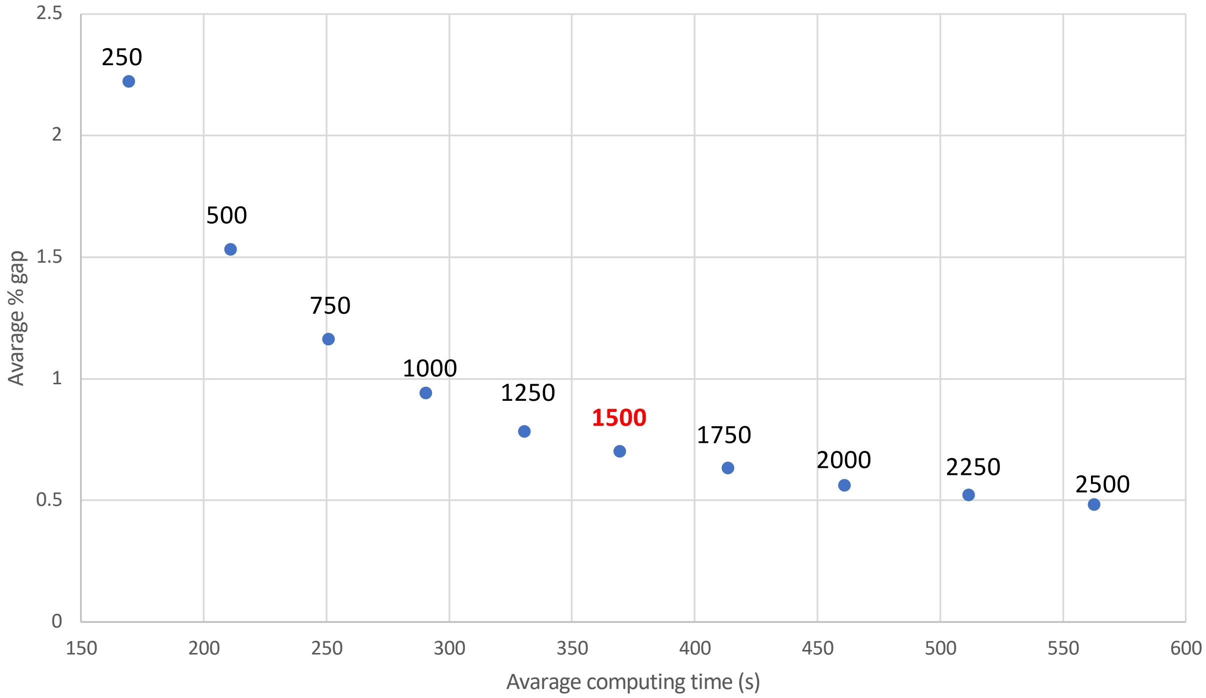
Neighbors Lists

- kd-tree based neighbors lists computation still takes a relevant portion of the overall computing time!
  - Full sorting is impossible

Instance	Neighbors list comput (s)	FILO2 (%)	FILO2 (long) (%)
Valle d'Aosta (20k)	4	2.09	0.15
Molise (50k)	10	6.86	0.53
Trentino-Alto Adige (100k)	23	11.86	1.00
...	...	...	...
Emilia-Romagna (900k)	235	46.82	7.19
Lombardia (950k)	242	43.04	3.03
Lazio (1M)	258	48.58	6.57
Average	**122**	**32.89**	**3.50**

Neighbors Lists

- n_{nn} affects the final solution value



2nd Challenge: Recreate Strategy

- Given a un-routed customer, searching for the best insertion position is too expensive and seldom useful
  - In XXL instances, it's unlikely that this position is on the opposite side of where the customer is positioned
- In FILO2, we only consider neighbor customers (available from neighbors lists) when searching for a candidate insertion position

	Time (s)	Gap
Best insertion	1224	1.02
Limited best insertion	370	0.70

- A limited best insertion experimentally shown to be effective on final solution quality
  - See also the blink strategy in SISR, Christiaens and Vanden Berghe (2020)

3rd Challenge: Simulated annealing temperature

- FILO uses a SA temperature based on the average instance arc cost
  - Computing this value can be extremely expensive
- In FILO2 we simply rely on a random sample of N instance arc costs

Instance	Exact temperature	Exact time (s)	Approx temperature	Approx time (ms)
Valle d'Aosta (20k)	1784.73	1.32	1780.85	0.00
Molise (50k)	3558.74	8.27	3553.99	0.00
Trentino-Alto Adige (100k)	4809.19	33.11	4810.42	1.20
…				
Emilia-Romagna (900k)	8527.99	2686.10	8526.11	59.20
Lombardia (950k)	6767.92	2993.95	6768.97	65.50
Lazio (1M)	5711.48	3315.50	5709.98	65.70
Average	**6451.97**	**1095.73**	**6452.09**	**29.51**

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani Piano Nazionale di Ripresa e Resilienza | Alma Mater Studiorum Università di Bologna

4th Challenge: Solution copies

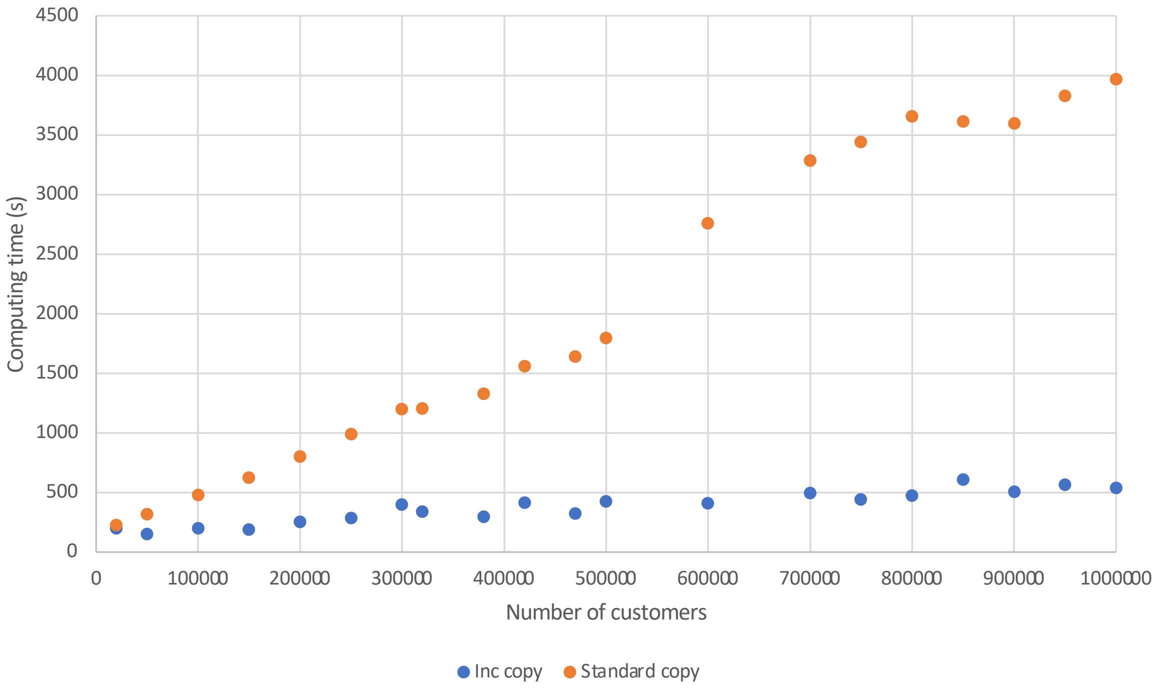
- Solution data structure copy
  - Performed at every algorithm iteration
    1. S current solution
    2. S’ = S
    3. Apply ruin & recreate + local search to S’ to obtain an actual neighbor of S
  - Step 2 is very expensive for large scale instances!
- Step 3 is very localized thanks to the SVC
- Full copy is not necessary
  - Difference between S and S’ is minimal if
    - Instance is large enough
    - SVC max capacity is limited

Sync Solutions by using Incremental Changes

- Create two identical solutions S and S’
  - This requires a single full copy
- During ruin & recreate + local search applied to S’
  - Record individual changes into a do-list D
    - E.g., remove vertex i from route r' and insert vertex i before j in route r
  - Record individual inverse changes into an undo-list U
    - E.g., remove vertex i from route r and insert it in its previous position in route r'
- To make S equal to S’
  - Apply changes in D to S
- To make S’ equal to S
  - Apply changes in U to S’ in reverse order

Sync Solutions by using Incremental Changes

- Solution copy is no longer a linear procedure
  - But it is bound to the actual number of changes performed during neighbor generation
  - In FILO, neighbor generation is very efficient by design

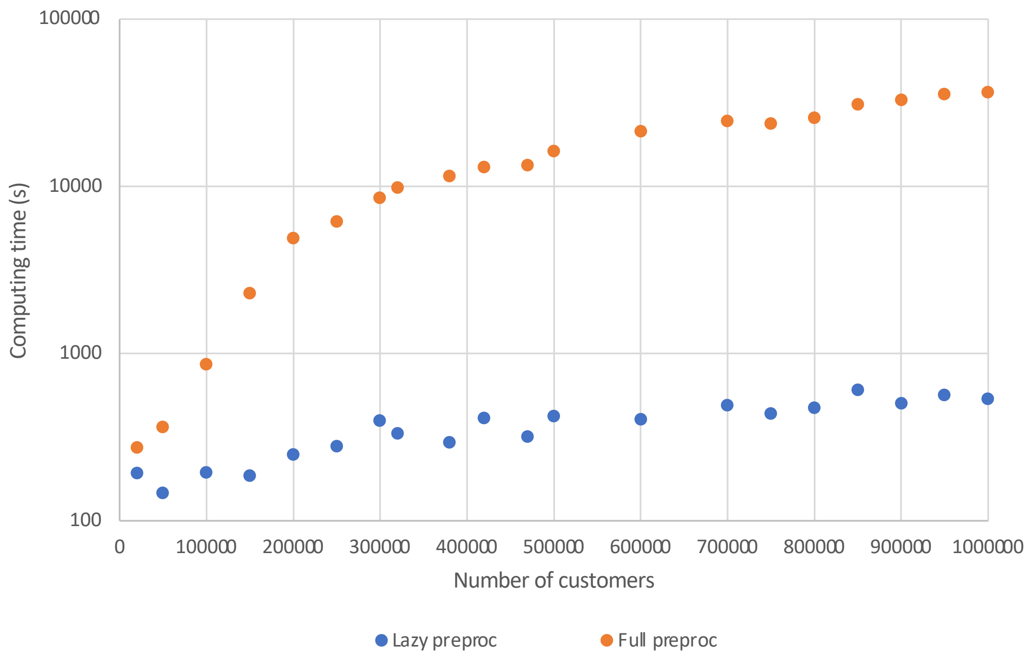


5th Challenge: Local Search Operators Preprocessing

- Some local search operators benefit from some preprocessing
  - E.g., inter-route 2 opt (called SPLIT and TAILS in FILO)
  - Feasibility check in costant time if cumulative demands are available
- Performing a full preprocessing is expensive (and useless!)
  - In FILO we were computing the cumulative demands for every customer and route...
  - As the local search is very localized, there is no real need to perform a full solution preprocessing

Lazy Local Search Operators Preprocessing

- Preprocess a route only when required
  - E.g., whenever a feasibility check involving such a route is requested
- Cache the preprocessed data until the route is changed



6th Challenge: HRVND

- FILO uses 20 local search operators organized as HRVND
  - 2 tiers (RVND): all quadratic cardinality operators, then ejection chain
  - In every tier we loop through the operators multiple times until we are in a local optimum for such a tier, before moving to the next tier
  - To save a bit of time, in FILO2, we only perform a single loop per tier
    - We are already re-applying the whole HRVND whenever an improvement was found
    - Multiple passes within the same tier are unlikely to cause significant quality improvements

	Time (s)	Gap
Standard	413	0.69
Single loop	370	0.70

Computational Testing

- Testing goal
  - Compare FILO vs FILO2 on literature instances (X and Belgium)
  - Provide some results for the new I instances

- Testing on a mini-computer
  - AMD Ryzen 5 PRO 4650GE CPU (3.3 GHz), used in single-thread
  - 16 GB RAM

- Algorithm versions
  - Standard (100k core opt iters)
  - Long (1M core opt iters)

- All numbers refer to the average of 10 runs!

Testing on X Instances

- Main reference literature dataset for the CVRP
  - 100 instances having from 100 to 1000 customers
  - Several customer demand distributions and vertex layouts

	FILO		FILO2	
Vertices	**Avg**	**t(s)**	**Avg**	**t(s)**
101-247	0.18	78	0.17	75
251-491	0.39	73	0.36	73
502-1001	0.53	75	0.50	82
Average	**0.37**	**75**	**0.34**	**76**

	FILO (long)		FILO2 (long)	
Vertices	**Avg**	**t(s)**	**Avg**	**t(s)**
101-247	0.08	827	0.08	807
251-491	0.25	771	0.23	769
502-1001	0.32	763	0.29	831
Average	**0.22**	**786**	**0.20**	**801**

	SISR	HGS
Vertices	**Avg**	**Avg**
101-247	0.11	0.01
251-491	0.23	0.08
502-1001	0.24	0.25
Average	**0.20**	**0.11**

Reference state-of-the-art algorithms when performed for 240n/100 seconds

- HGS: Hybrid Genetic Search, Vidal (2022)
- SISR: Slack Induction by String Removals, Christiaens and Vanden Berghe (2020)

Results taken from Vidal (2022)

Testing on Belgium Instances

- Large scale dataset for the CVRP
  - 10 instances having from 3k to 30k customers

	FILO		FILO2	
	Avg	t(s)	Avg	t(s)
Average	1.15	207	1.08	121

	FILO (long)		FILO2 (long)	
Vertices	Avg	t(s)	Avg	t(s)
Average	0.42	2315	0.37	1371

	KGLS (short)		KGLS	
	Gap*	t**(s)	Gap*	t**(s)
Average	2.63	2944	1.77	11774

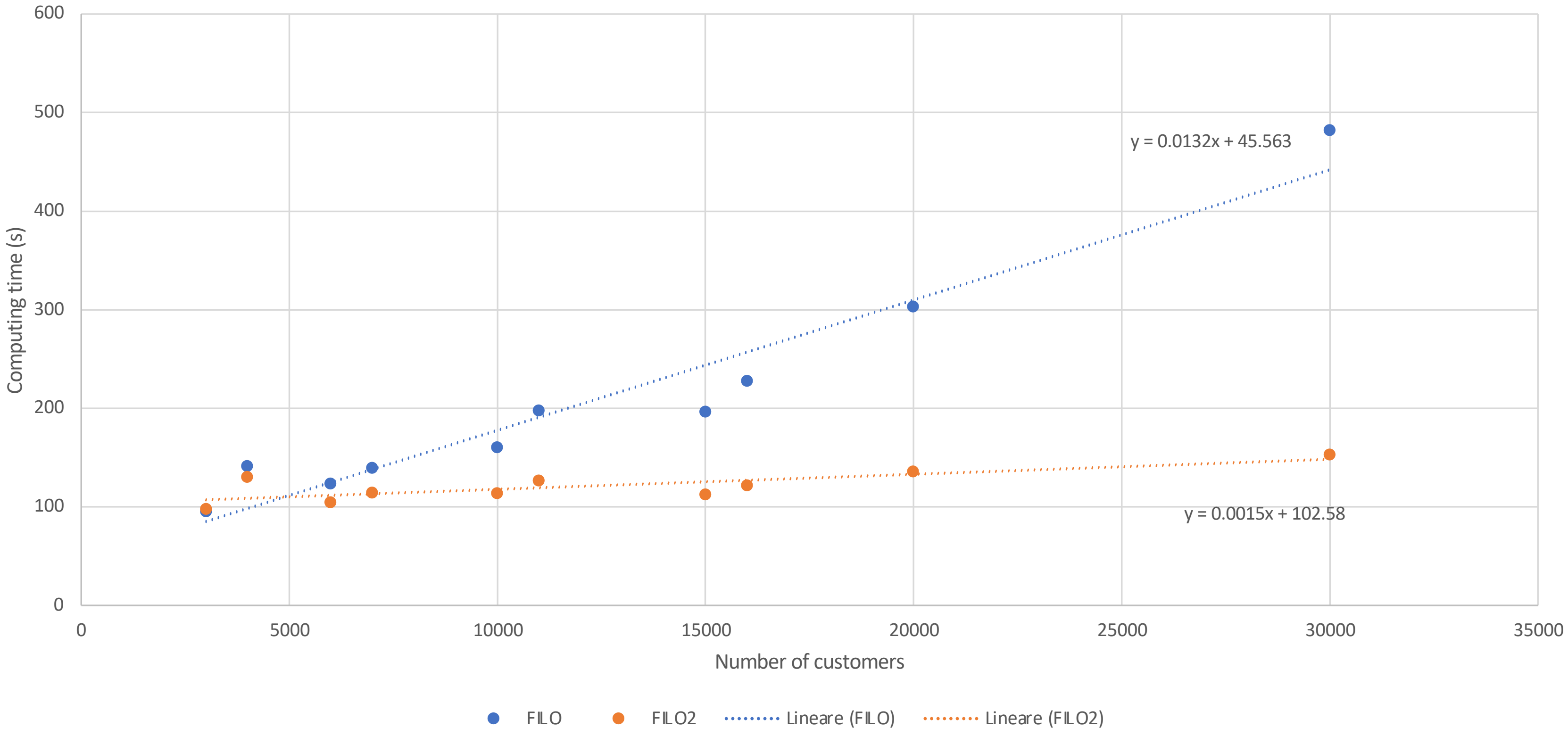
Reference state-of-the-art algorithm

- KGLS: Knowledge Guided Local Search, Arnold et al. (2019)

* Single run as KGLS is deterministic

** Roughly scaled to match our CPU

Testing on Belgium Instances



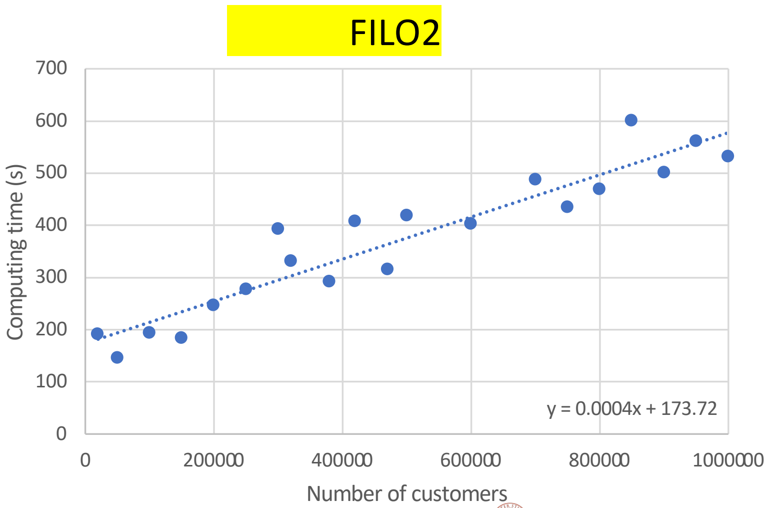
The two linear functions met approximately at 4876 customers

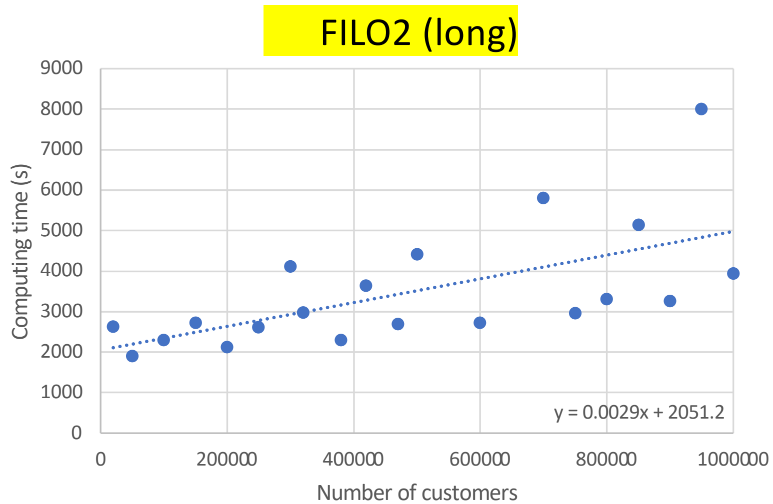
Testing on I Instances

- No competitors yet!
  - FILO can only be executed on the smallest instance with 20k customers (Vd'A)

	FILO2		FILO2 (long)	
	Avg*	**t(s)**	**Avg***	**t(s)**
Average	**0.70**	**370**	**0.30**	**3474**

* Wrt the best solution we found during all our experimentation



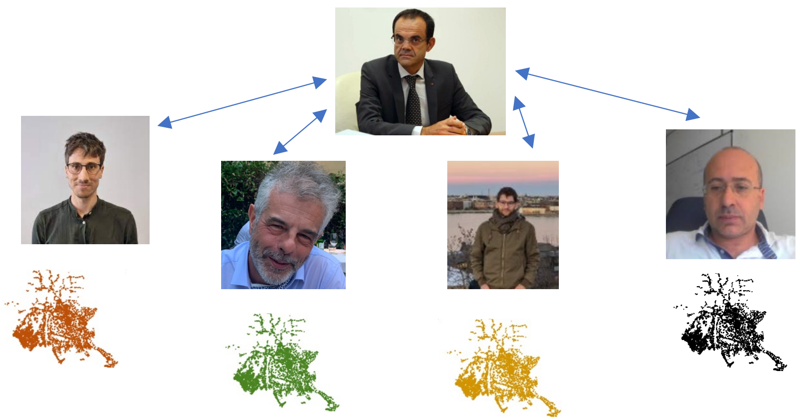


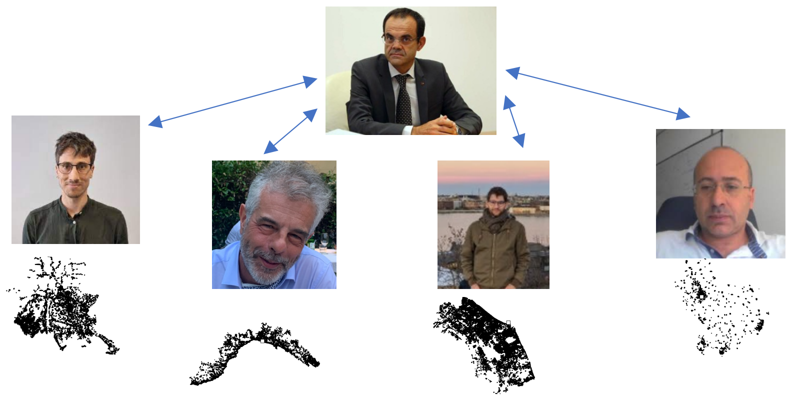
What's next ?

FILO goes PARALLEL !

Ongoing joint work with F. Michelotto, L. Accorsi, D. Laganà, R. Musmanno

- Using several FILO solver in parallel working with different settings/solutions
- Decomposing the instance and letting each solver working on a different part





Thank You!

- Report, slides and code
- https://github.com/acco93/filo