Usage of Iterated Local Search to improve Firewall Evolvability

Author: Geert Haerens

Presenter: Geert Haerens
Enterprise IT Architect @ Engie
PhD student at Antwerp University
geert.haerens@engie.com

Universiteit Antwerpen

IARIA

Presenter: Geert Haerens

- Geert Haerens holds a degree in industrial engineering (electricity and automation) and civil engineering (computer science and mechatronics). After having worked for 4 years at the NMBS and 8 years at AB Inbev, he started working for Engie/Electrabel as an IT Architect. In his pursuit for professionalizing the work of the IT Architect, he became a certified EA at the University of Carnegie Mellon and got his Master in Enterprise IT Architect at the Antwerp Management School. In addition to his job at Engie, he is currently doing research at the University of Antwerp on the applicability of the Normalized Systems theory on IT Infrastructure systems.

Content

1

Introduction

Introduction: The TCP/IP Firewall

Source	Destination	Service	Action
1.1.5.3	1.1.2.1	TCP/22	Deny
1.1.1.1 … 1.1.1.20	1.1.2.1	TCP/22	Allow
1.1.1.1	1.1.2.1	UDP/10-25	Deny

Introducing Normalized Systems

What is it about?

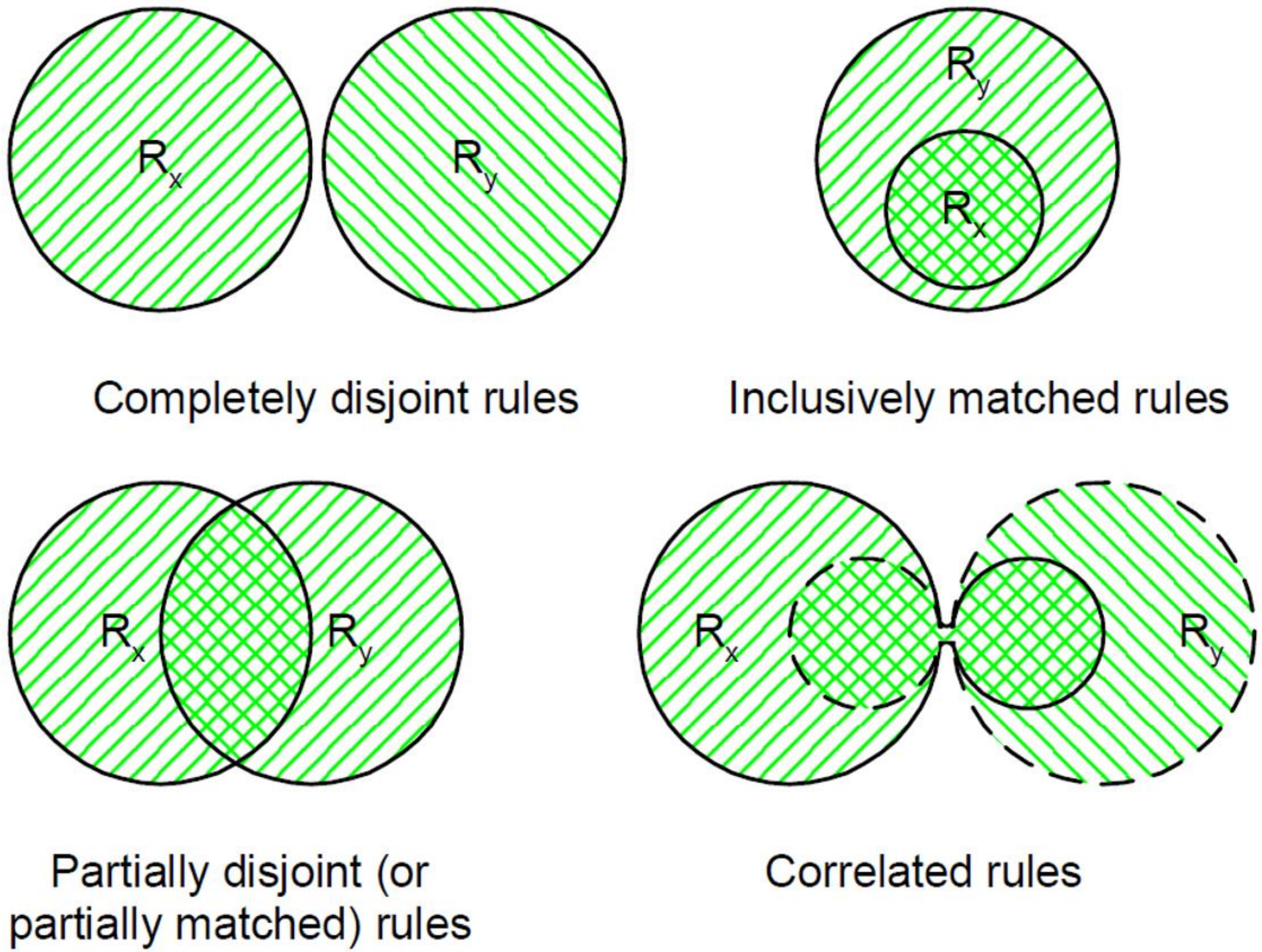
- Studies the evolvability of modular software systems.
- Defines 4 theorems as the necessary conditions a modular structure must adhere to, for evolvability.
- A systems is considered evolvable when it is stable under change.
- Stable under changes = Bounded input leads to Bounded output.

- A limited functional change (bounded input) must lead to a limited change in software modules (bounded output).
- If not, a Combinatorial Effect is observed: change is proportional to the system itself.

2

The Problem

Problem: Relationships between rules



3

The Artifact

Artifact: Previous work and requirements

A "Green Field" Artifact:
Enforce disjointness of service definitions – use destination definitions that represent host/service combinations.
Provides and evolvable rule base with respect to anticipated changes.

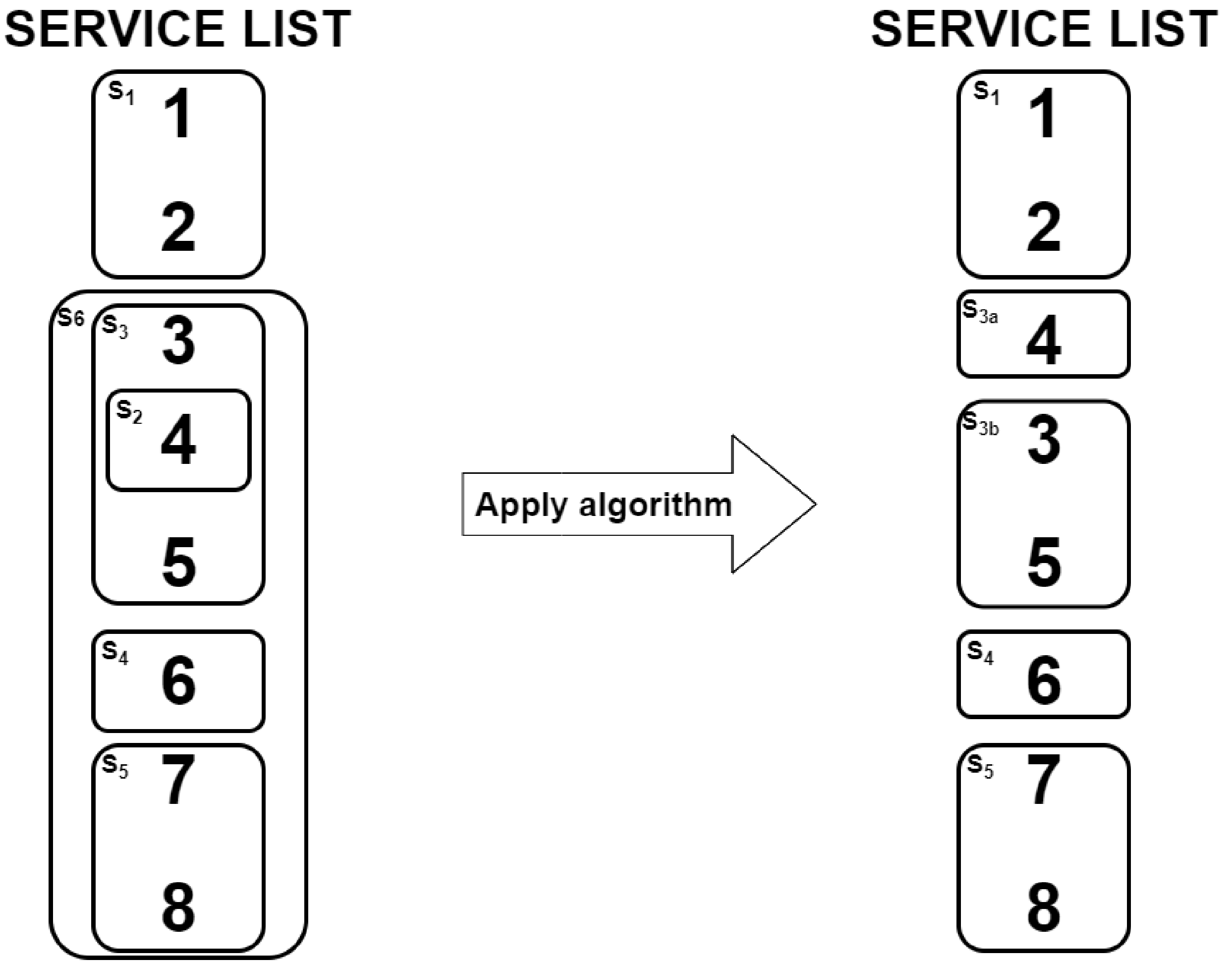
ADD	RESULT
a rule	no CE
new service	no CE
new host with existing service	no CE
new host with new service	no CE
a client	no CE

REMOVE	RESULT
a rule	no CE
a service	no CE
a host	no CE
service from a host	no CE
a client	CE at client level

A "Brown Field" Artifact:
Convert an existing rule base into an evolvale rule base
Necessary condition (not sufficient): disjoint Service Definitions

Artifact: Previous work and requirements

Break Relationships - Disentangling Services



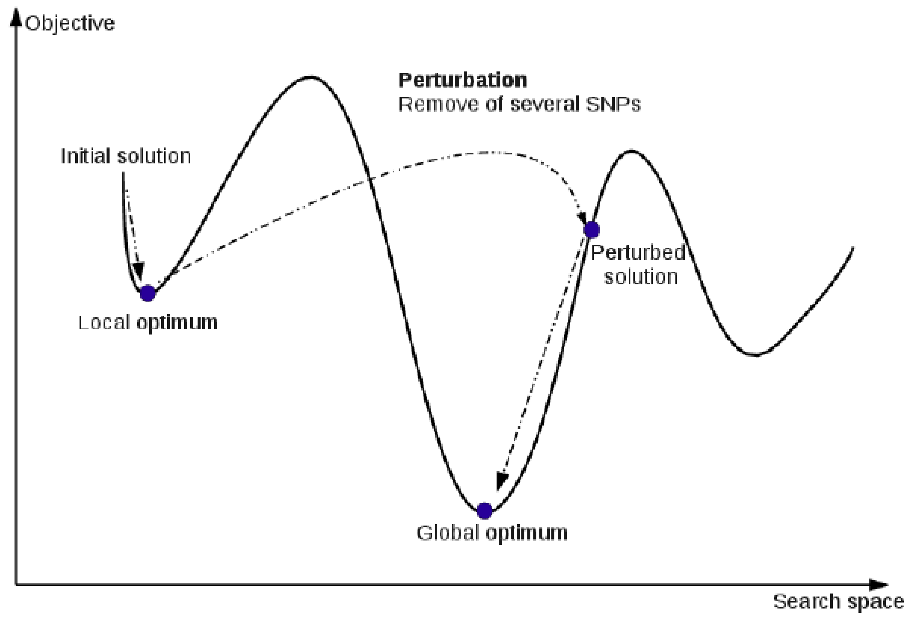
Artifact: Iterated Local Search Metaheuristic

REPEAT:

Do a local search until a local optimum is reached

Perform a perturbation

UNTILL (Stop Condition)

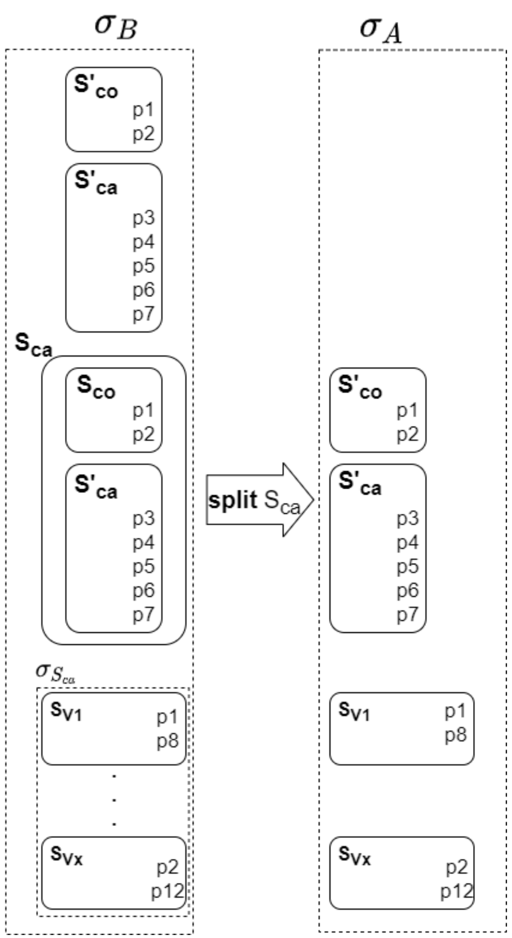


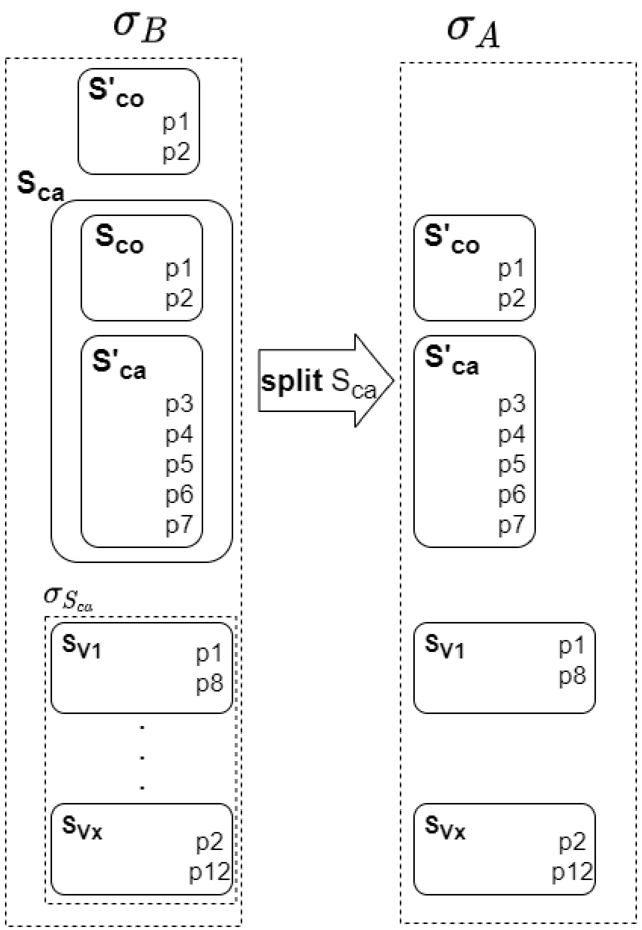
Artifact: Algorithm components

- **Port Frequencies:** how many times is a port used in service definitions.
- **Disjointness Index:** sum of all Port Frequencies of a service definition, divided by the number of ports.

- **Initial Solution:** a give rule base – the service definitions
- **Neighbourhood:** DI of all service definitions
- **Objective Function:** Sum of the DI's of all service definitions in the solution
- **Move Type:** Split a service – carve out all existing subgroups
- **Move Strategy:** Split service with highest DI
- **Perturbation:** Split service – according to overlap
- **Stop Conditions:** full neighbourhood searched, full disjointness reached

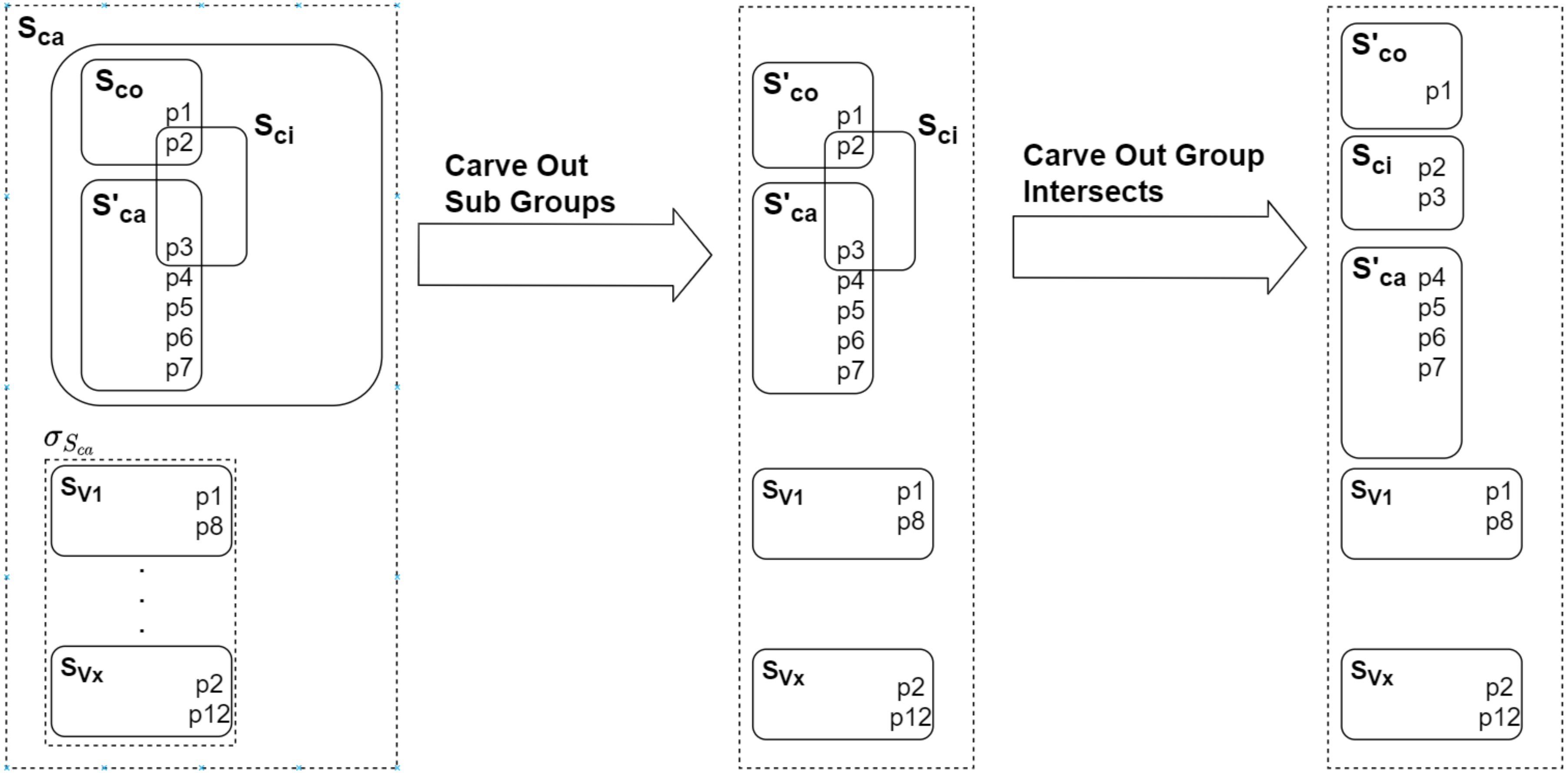
Artifact: Move – subgroup carve out





Always improves (= decent) the Objective Function = SUM of DI

Artifact: Perturbation – intersect carve out



Sometimes improves (= decent) the Objective Function = SUM of DI

Artifact: Iterated Local Search

Algorithm 1: Iterated Local Search for service list normalization

```
sl = load_initial_solution(filename);
pfl = portfrequencies_list_constructor(sl).get_portfrequencies_list();
sdil = service_di_list_creator(sl, pfl).get_service_di_list();
of = service_di_list.get_objective_function();
fully_disjoint = FALSE;
end_of_neighbourhood = FALSE;
objective_function_improvement = FALSE;
while NOT full_disjoint AND NOT end_of_neighbourhood do
    neighbourhood = sdil.sort;
    neighbourhood_pointer = 1 (top of list)
    objective_function_improvement = FALSE;
    while NOT improvement_objective_function AND NOT
     fully_disjoint AND NOT end_of_neighbourhood do
        servicer_to_split =
         neighbourhood.get_element(neighbourhood_pointer);
        service_split_evaluator(service_to_split, sdil, pfl);
        objective_function_improvement =
         service_split_evaluator.get_objective_function_improved();
        if objective_function_improvement = TRUE then
            sl= service_split_evaluator.get_service_list();
            pfl = service_split_evaluator.get_portfrequencies_list();
            sdil = service_split_evaluator.get_service_di_list();
            fully_disjoint = service_di_list.is_fully_disjoint_check();
        else
            neighbourhood_pointer ++
        end
        end_of_neighbourhood =
         sdil.end_of_list_check(nieghbourhood_pointer);
    end
    if end_of_neighbourhood then
        service_perturbation_exists = service_perturbation.perturbation
         exists(sl,pfl);
        if service_perturbation_exists then
            sl= service_split_evaluator.get_service_list();
            pfl = service_split_evaluator.get_portfrequencies_list();
            sdil = service_split_evaluator.get_service_di_list();
            fully_disjoint = service_di_list.is_fully_disjoint_check();
            end_of_neighbourhood = FALSE;
        else
            end_of_eighbourhood = TRUE;
        end
    end
end
if fully_disjoint then
    PRINT "Probably the Global Optimum has been found";
else
    PRINT "Local Optimum found";
end
PRINT "Solution = " + sl.get_overview();
```

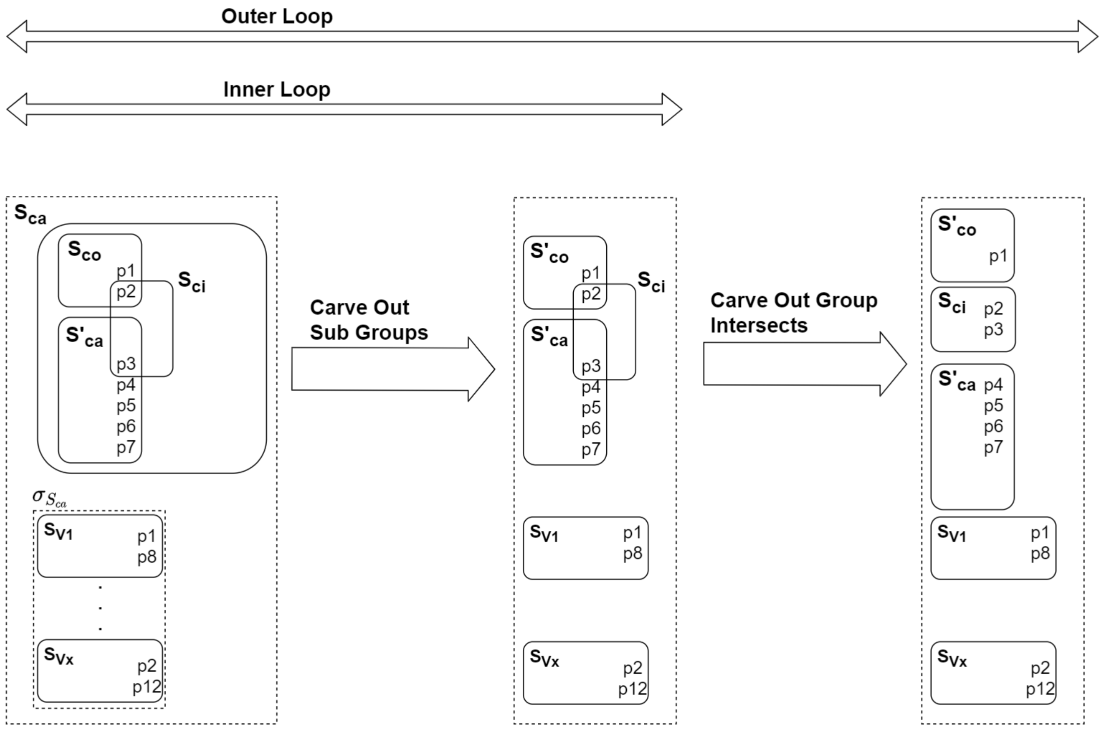
While (end conditions not reached)

While (there are still subgroups)

Do a full cave out

Make a perturbation

result



4

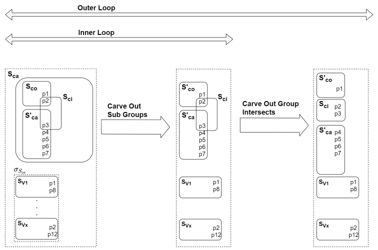
Demonstration

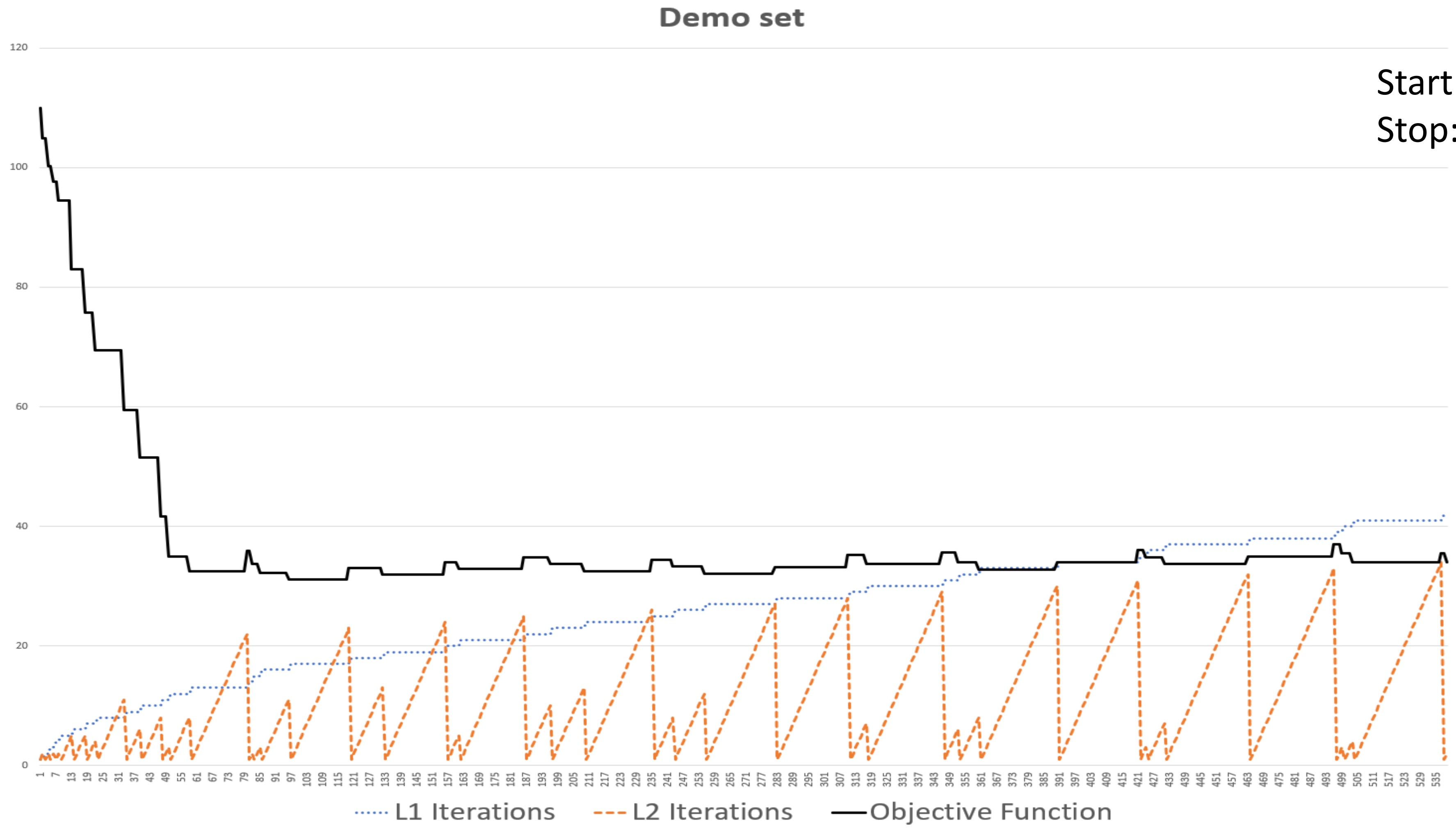
Demonstration

3 Data sets are used:

- A Demo set: including a lot of exceptions
- A Tractebel set: operational firewall connecting a branch office to the company network
- A Engie IT DC set: operational firewall connecting tooling and management systems to client systems

Demonstration – demo set

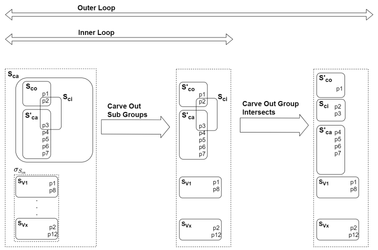


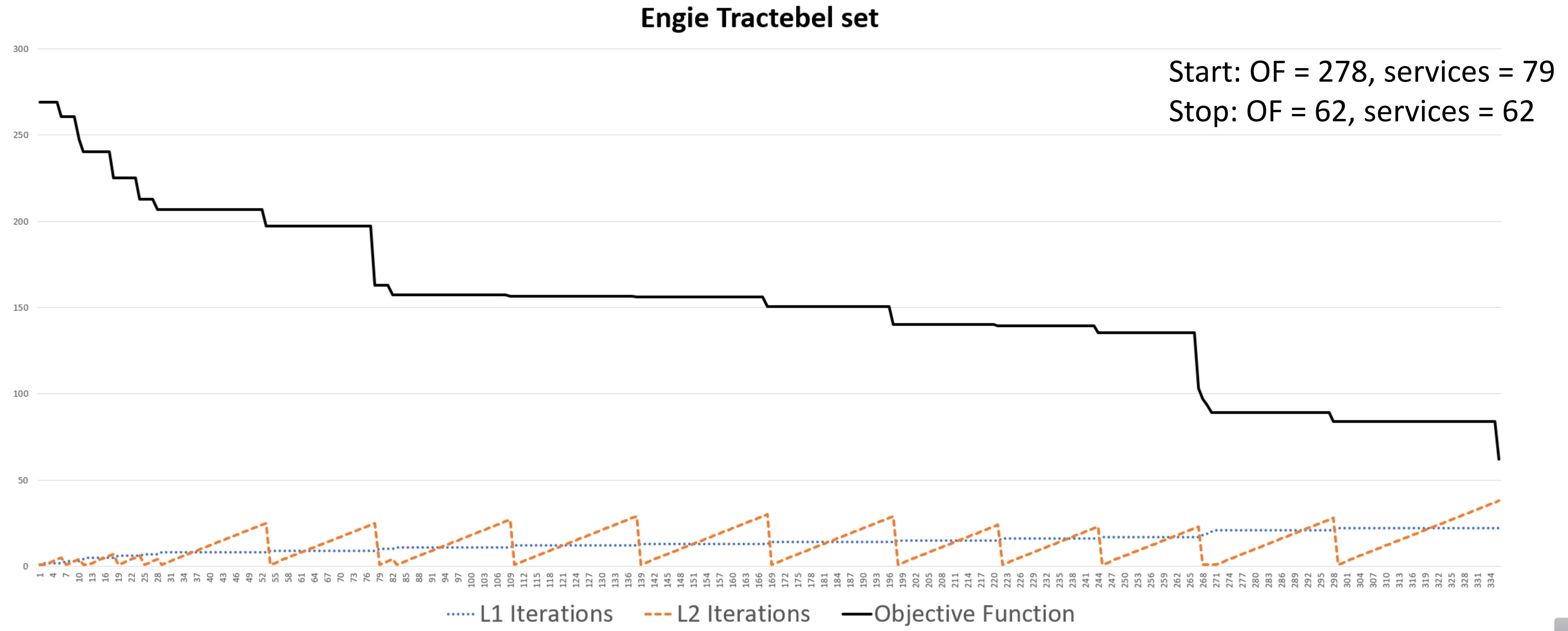


Start: OF = 110, services = 28

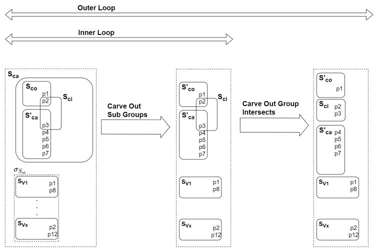
Stop: OF = 34, services = 34

Demonstration – Tractebel set



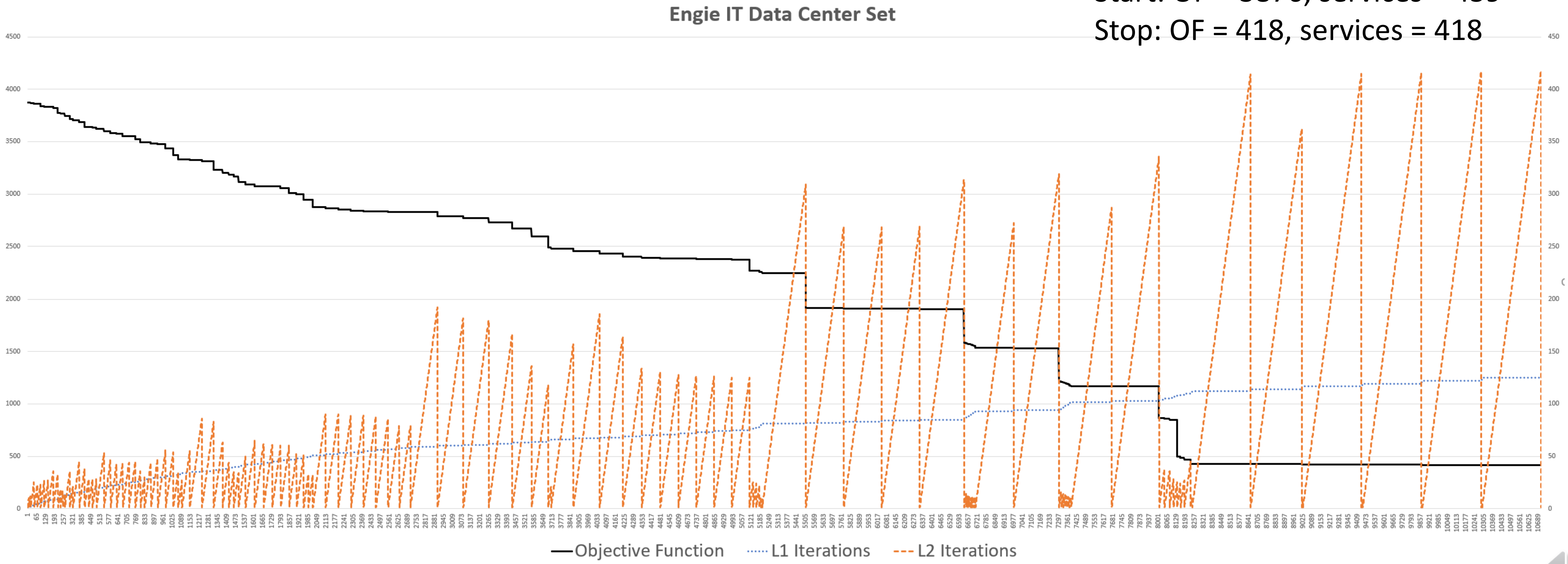


Demonstration – Engie IT set



Start: OF = 3876, services = 459

Stop: OF = 418, services = 418



5

Evaluation

Evaluation

- Big O = n^3
- Splitting services = impacting rules → how much extra rules?
- Essential building block for evolvable rule base creator
  - Destination splitting to be developed.
- Potential Improvement
  - Memory, performance optimizations.
- Global Optimum?

6

Conclusion

Conclusion

- Splitting Services = applying SoC.
- Resulting in fine grained rule base – fine grained modular structure with low coupling.

- The algorithm works

- The algorithm needs extension:
  - adjust rules → already done – rule base increases with an order of magnitude
  - Redefine destinations → to be done

THANK YOU

geert.haerens@engie.com

engie.com