Optimizing Number, Sequence and Type of Activities in Agents' Schedules

Matthias Feil

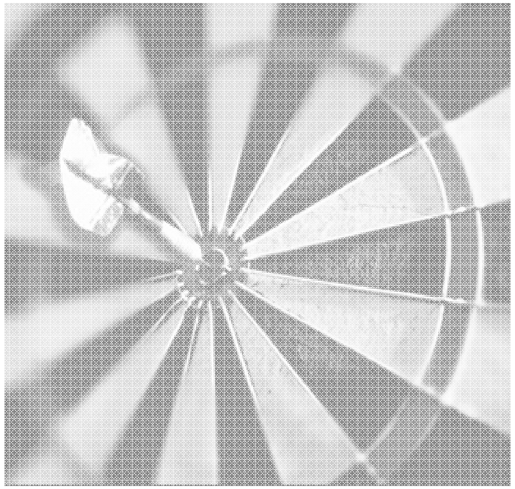
MATSim User Meeting
Berlin

April 2009

IVT Institut für Verkehrsplanung und Transportsysteme
Institute for Transport Planning and Systems

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Objectives of the presentation



- Short background on how the presentation fits into the broader MATSim environment
- Explanation of basic principles of the implemented algorithms
- Clarification of the use of the parameters

Agenda

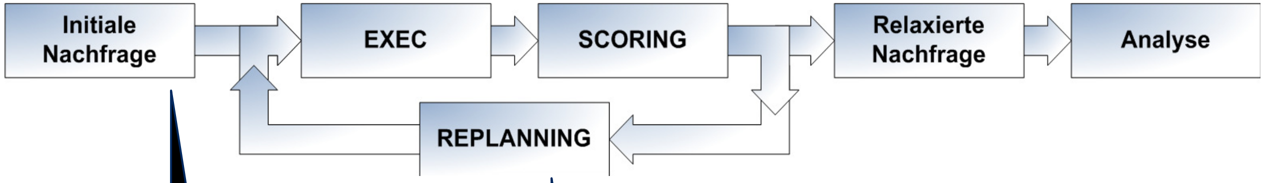
- Challenge and objective
- Optimization of agents' schedules
  - PlanomatX
  - TimeModeChoicer
  - Schedule Recycling
- Modification of the utility function
- "How do I use it?" and outlook

Agenda

- **Challenge and objective**
- Optimization of agents’ schedules
  - PlanomatX
  - TimeModeChoicer
  - Schedule Recycling
- Modification of the utility function
- “How do I use it?” and outlook

The objective is to establish a REPLANNING module that optimizes the number, type and order of the activities of a schedule

MATSim structure



Current situation

- The structure of agents‘ schedules is determined in the generation of the initial demand, based upon socio-economic data (e.g., census, microcensus)

- The REPLANNING step includes
  - Optimization of timings (Planomat)
  - Mode Choice (Planomat)
  - Secondary Location Choice
  - Route Choice
- However, the structure of agents‘ schedules (number, type and order of activities) is kept fixed throughout the evolutionary learning process. This is an artificial constraint

Objective

- Establish a REPLANNING module that optimizes the structure of agents‘ schedules, given the generalized cost of travelling (and activity participation) determined in the previous EXEC step

Agenda

- Challenge and objective

- **Optimization of agents' schedules**
  - **PlanomatX**
  - TimeModeChoicer
  - Schedule Recycling

- Modification of the utility function

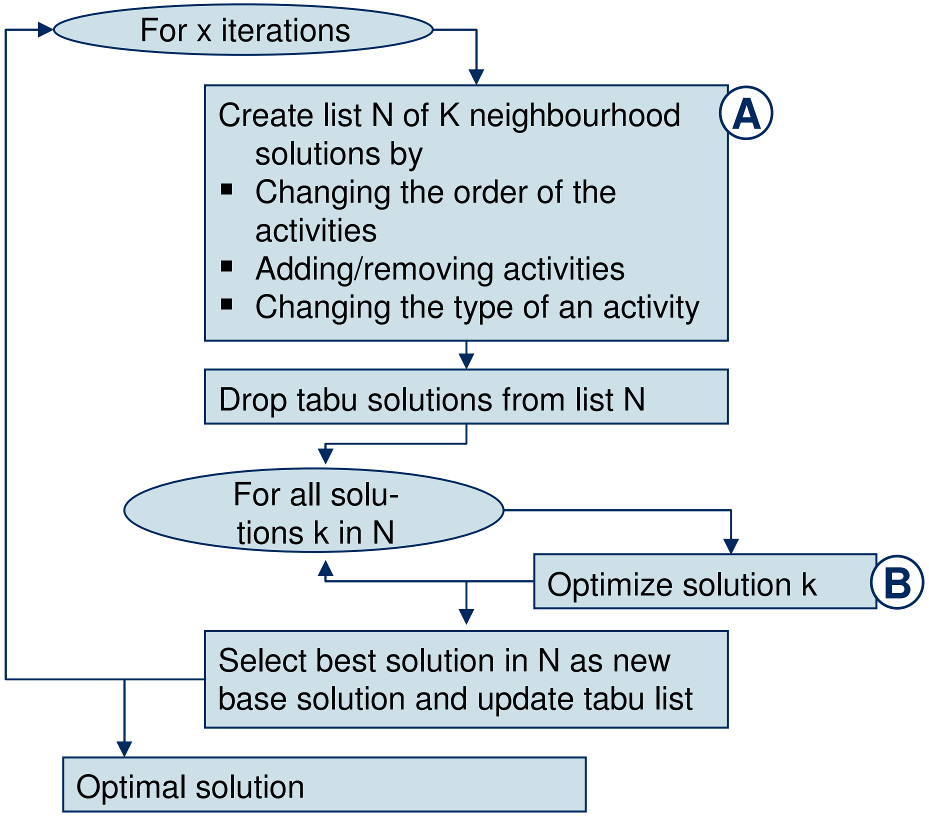
- "How do I use it?" and outlook

PlanomatX implements a Tabu Search heuristic and optimizes the number, type and order of the activities of a schedule

Rationale for employment of Tabu Search heuristic

- GAs able to reliably find (nearly) global optimum but known as rather inefficient*
- Global optimum no ultimate objective since people do not globally optimize either
- Tabu Search expected to bring gains in computational performance: it quickly relaxes to an „ok“ solution from which it (slowly) directs towards global optimum. „Ok“ solution may suffice for MATSim application

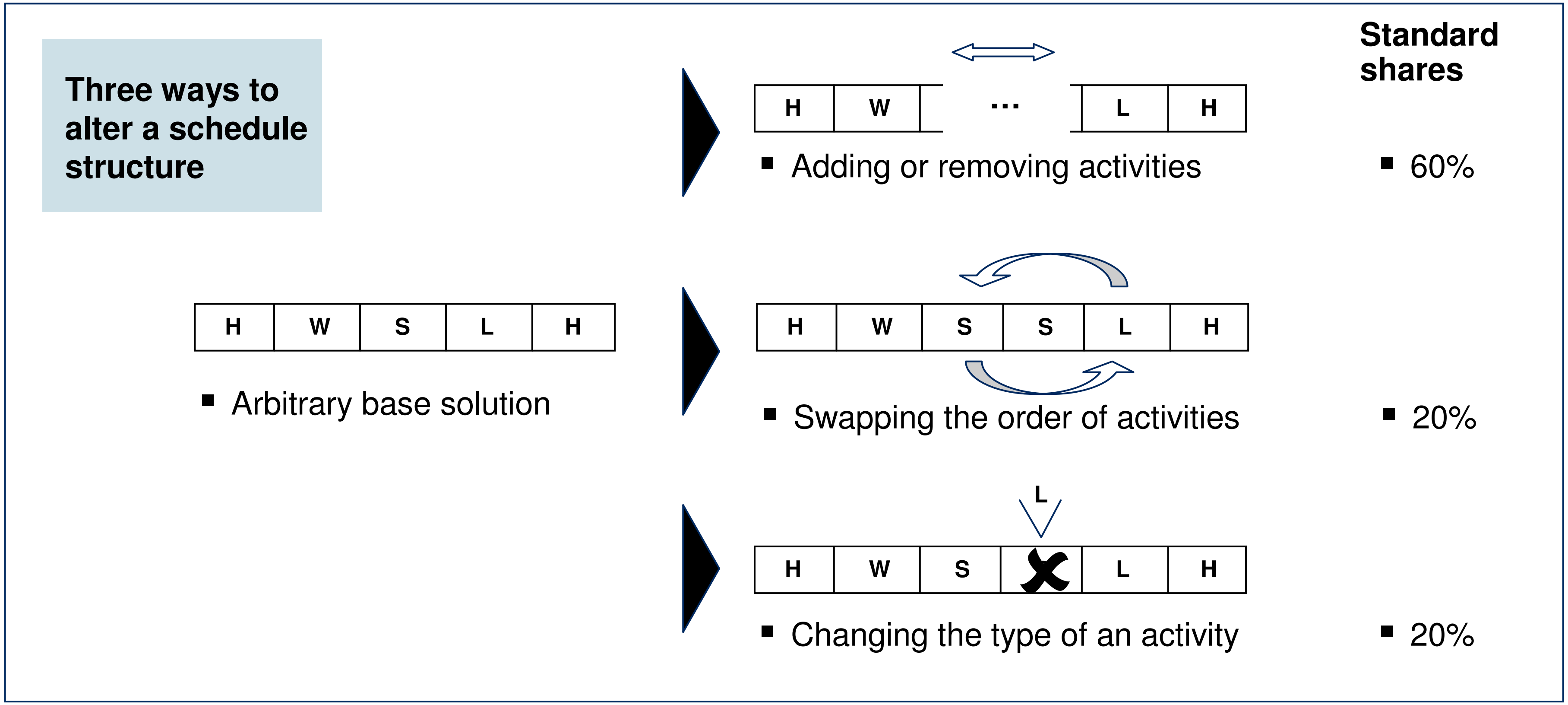
High-level PlanomatX process flow


For x iterations
Create list N of K neighbourhood solutions by
▪ Changing the order of the activities
▪ Adding/removing activities
▪ Changing the type of an activity
A
Drop tabu solutions from list N
For all solu-tions k in N
Optimize solution k
B
Select best solution in N as new base solution and update tabu list
Optimal solution

* See e.g.,
Charypar, D. and K. Nagel (2005) Generating complete all-day activity plans with genetic algorithms, *Transportation*, **32** (4) 369-397
Meister, K., M. Frick and K.W. Axhausen (2005b) A GA-based household scheduler, *Transportation*, **32** (5) 473 – 494.

The neighbourhood creation is the most critical step in Tabu Search – PlanomatX implements three types of „moves“

PlanomatX neighbourhood creation



Three ways to alter a schedule structure

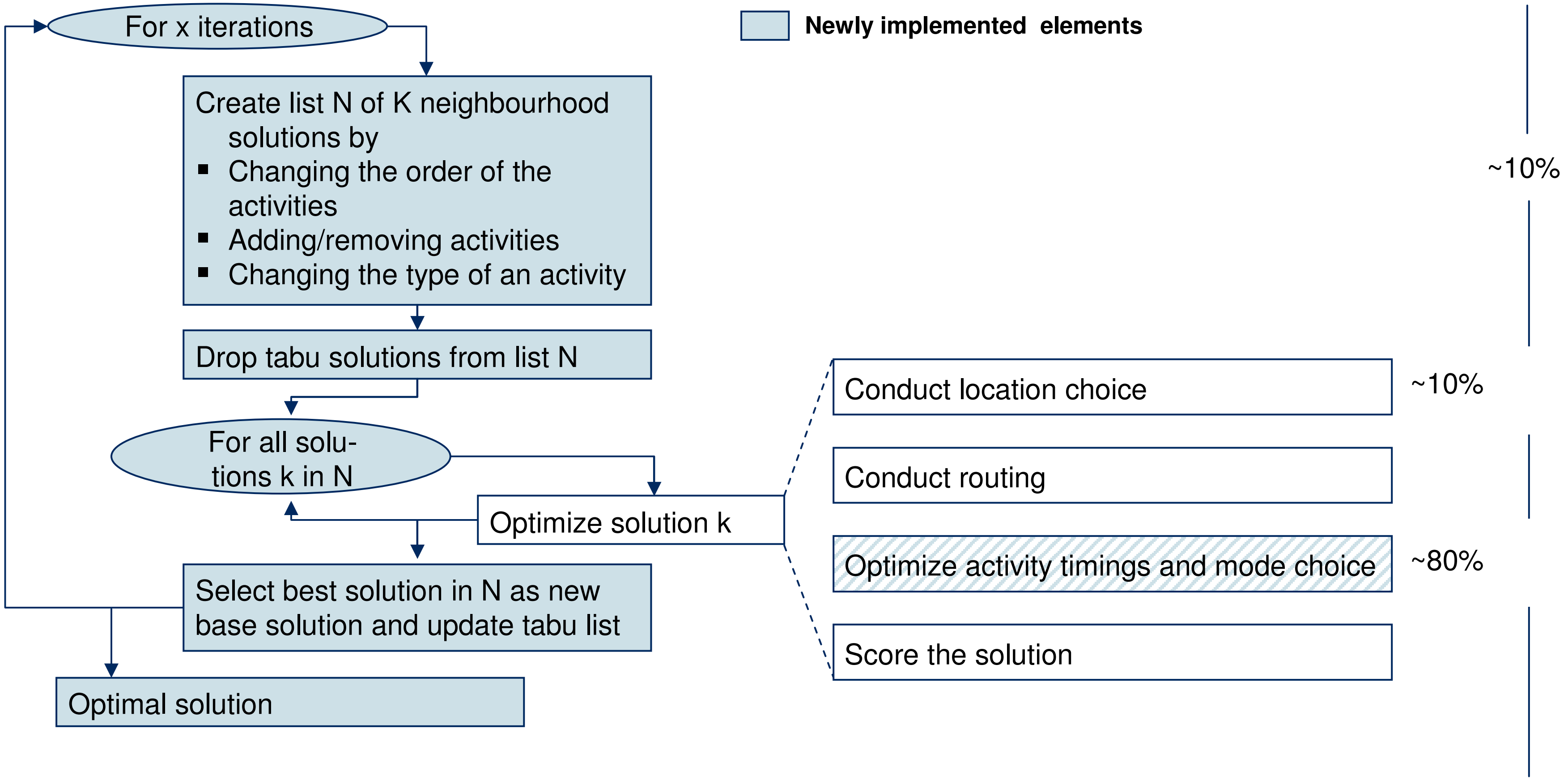
Standard shares

- Arbitrary base solution
- Adding or removing activities — 60%
- Swapping the order of activities — 20%
- Changing the type of an activity — 20%

- Trade-off of neighbourhood creation
  - Create as few „waste“ solutions as possible, but
  - Do not constrain the neighbourhood search such that good moves cannot be reached

B For each neighbourhood solution, activity timings as well as location, route and mode choices are optimized

PlanomatX process flow **Runtime shares**

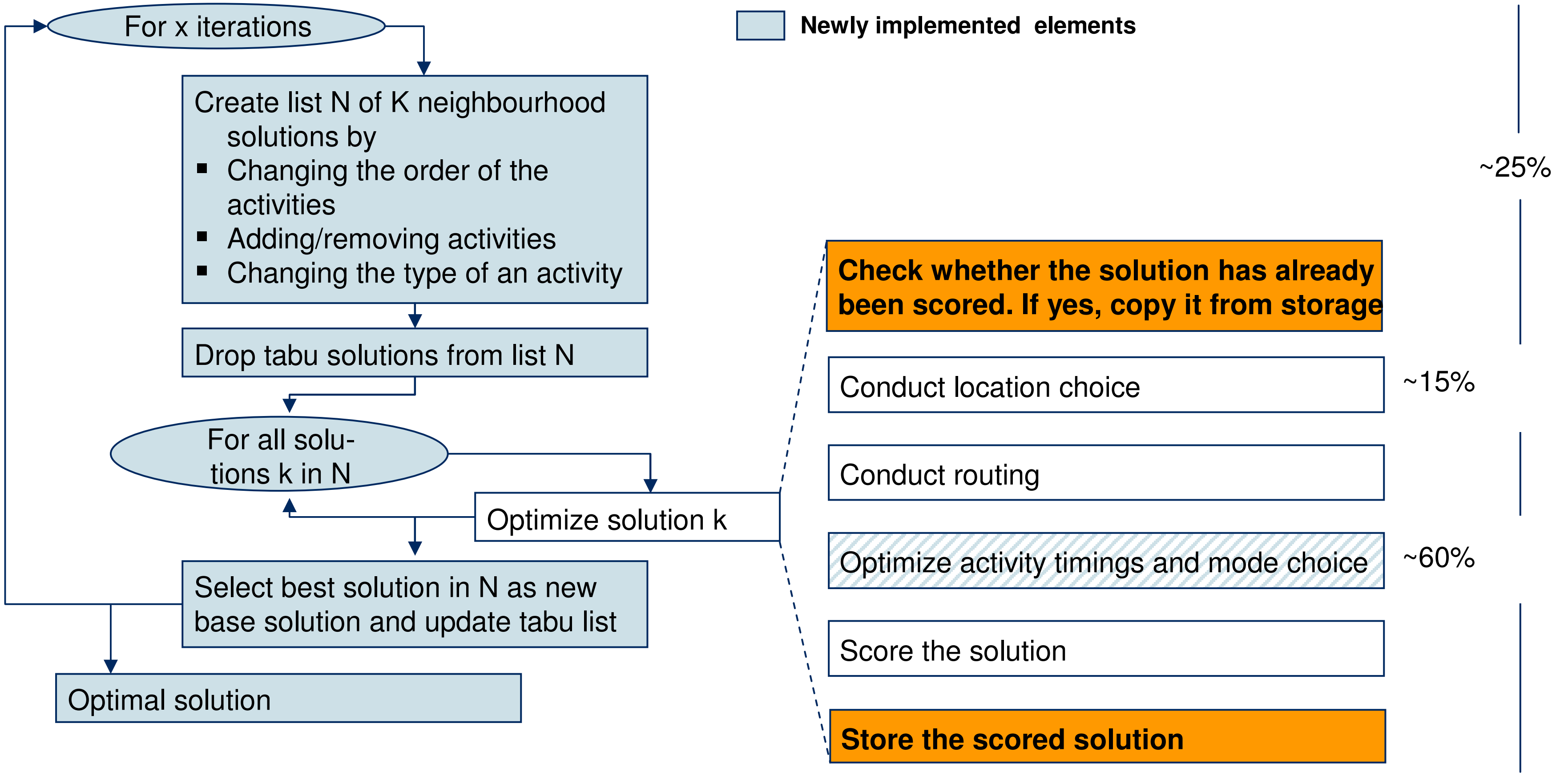


Newly implemented elements
For x iterations
Create list N of K neighbourhood solutions by
Changing the order of the activities
Adding/removing activities
Changing the type of an activity
Drop tabu solutions from list N
For all solutions k in N
Optimize solution k
Select best solution in N as new base solution and update tabu list
Optimal solution
Conduct location choice
Conduct routing
Optimize activity timings and mode choice
Score the solution
~10%
~10%
~80%

- **Each neighbourhood solution (= schedule structure) is optimized in itself**
- **This implies that, if a schedule structure‘s score is lower than that of another structure it can be entirely dropped**

Ⓑ The optimization of activity timings and mode choice is time-consuming, therefore a workaround has been implemented

PlanomatX process flow **Runtime shares**



🟦 **Newly implemented elements**

For x iterations

Create list N of K neighbourhood solutions by
- Changing the order of the activities
- Adding/removing activities
- Changing the type of an activity

Drop tabu solutions from list N

For all solutions k in N

Optimize solution k

Select best solution in N as new base solution and update tabu list

Optimal solution

Check whether the solution has already been scored. If yes, copy it from storage

Conduct location choice

Conduct routing

Optimize activity timings and mode choice

Score the solution

Store the scored solution

~25%

~15%

~60%

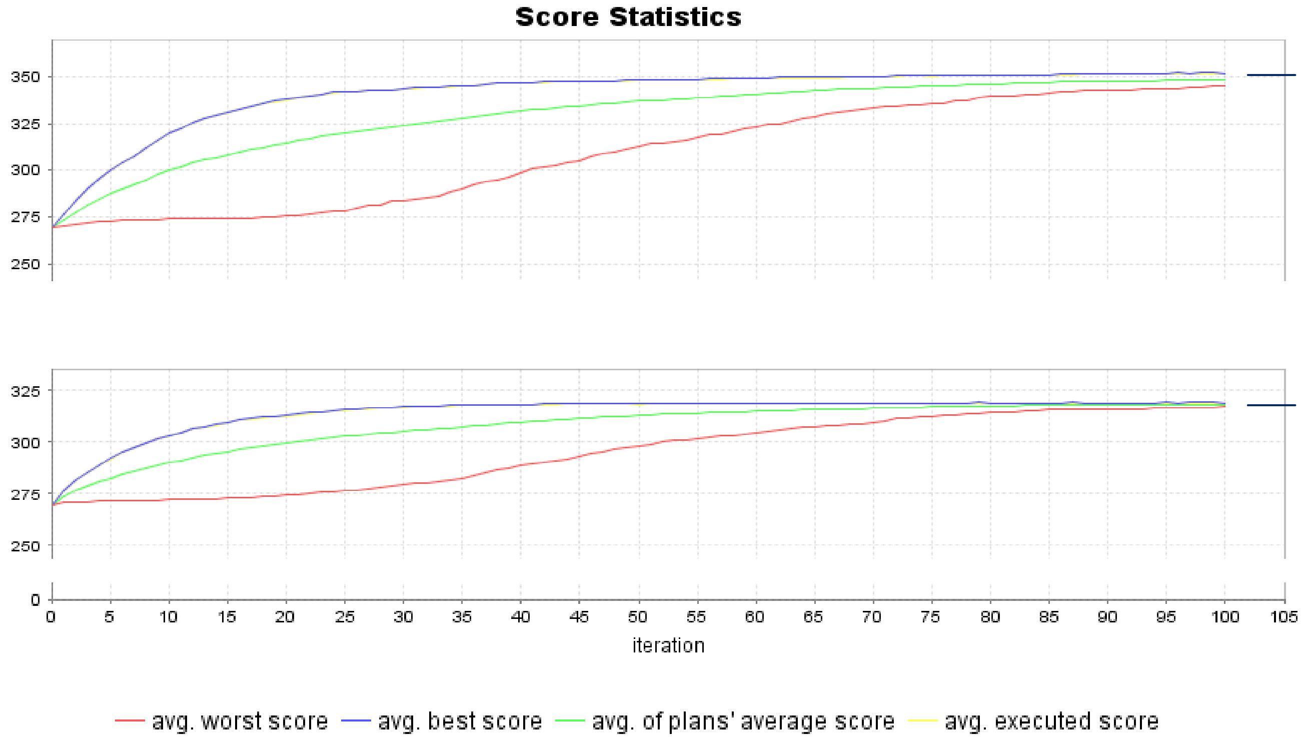
Σ -30%

- **About 35% of solutions can be retrieved from the storage**
- **The storage and retrieval of solutions reduces the overall runtime by about 30%**

Relaxation on a simple chessboard-like network with about 300 agents – PlanomatX yields a higher average score than pure time optimization…

Optimization of activity planning
(PlanomatX)

Time optimization only
(Time-Optimizer)



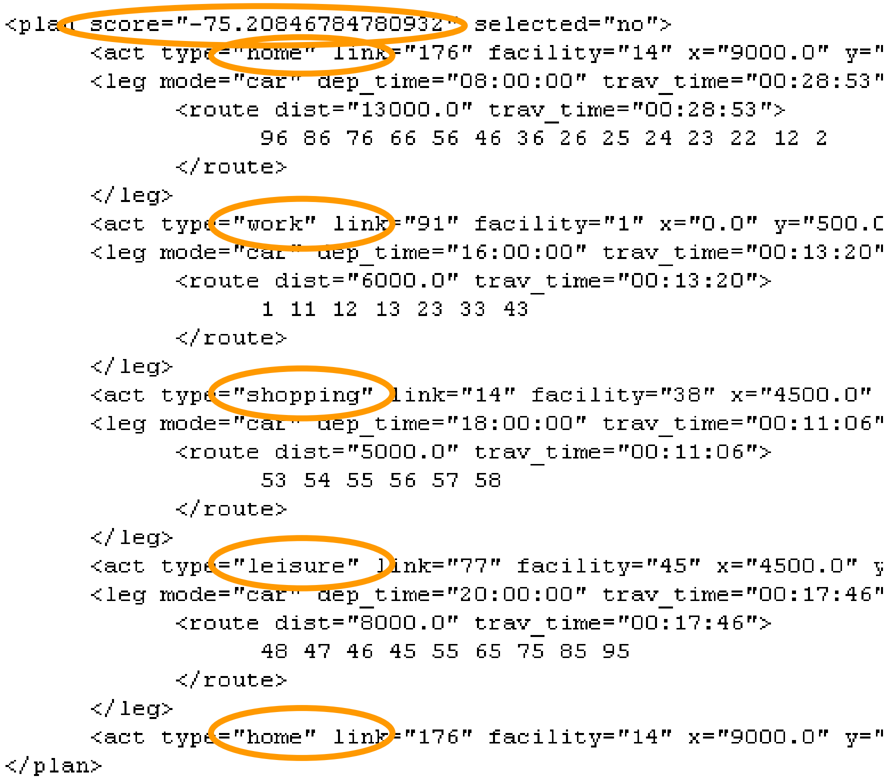
~350

~320

- **Optimization with flexible activity chain yields higher score than just optimizing the times of a fixed activity chain**

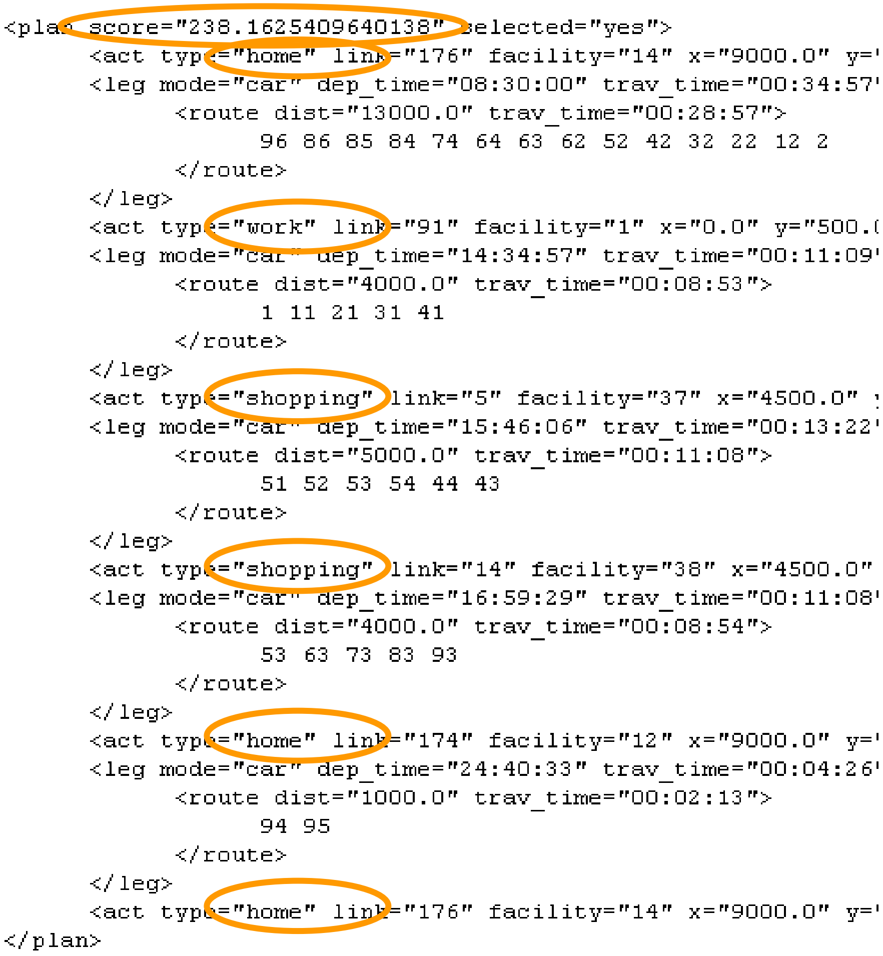
… and the flexible number of activities can obviously be observed in agents‘ schedules

Example initial plan



```
<plan score="-75.20846784780932" selected="no">
      <act type="home" link="176" facility="14" x="9000.0" y="
      <leg mode="car" dep_time="08:00:00" trav_time="00:28:53"
            <route dist="13000.0" trav_time="00:28:53">
                  96 86 76 66 56 46 36 26 25 24 23 22 12 2
            </route>
      </leg>
      <act type="work" link="91" facility="1" x="0.0" y="500.
      <leg mode="car" dep_time="16:00:00" trav_time="00:13:20"
            <route dist="6000.0" trav_time="00:13:20">
                  1 11 12 13 23 33 43
            </route>
      </leg>
      <act type="shopping" link="14" facility="38" x="4500.0"
      <leg mode="car" dep_time="18:00:00" trav_time="00:11:06"
            <route dist="5000.0" trav_time="00:11:06">
                  53 54 55 56 57 58
            </route>
      </leg>
      <act type="leisure" link="77" facility="45" x="4500.0" y
      <leg mode="car" dep_time="20:00:00" trav_time="00:17:46"
            <route dist="8000.0" trav_time="00:17:46">
                  48 47 46 45 55 65 75 85 95
            </route>
      </leg>
      <act type="home" link="176" facility="14" x="9000.0" y="
</plan>
```

Corresponding optimized plan



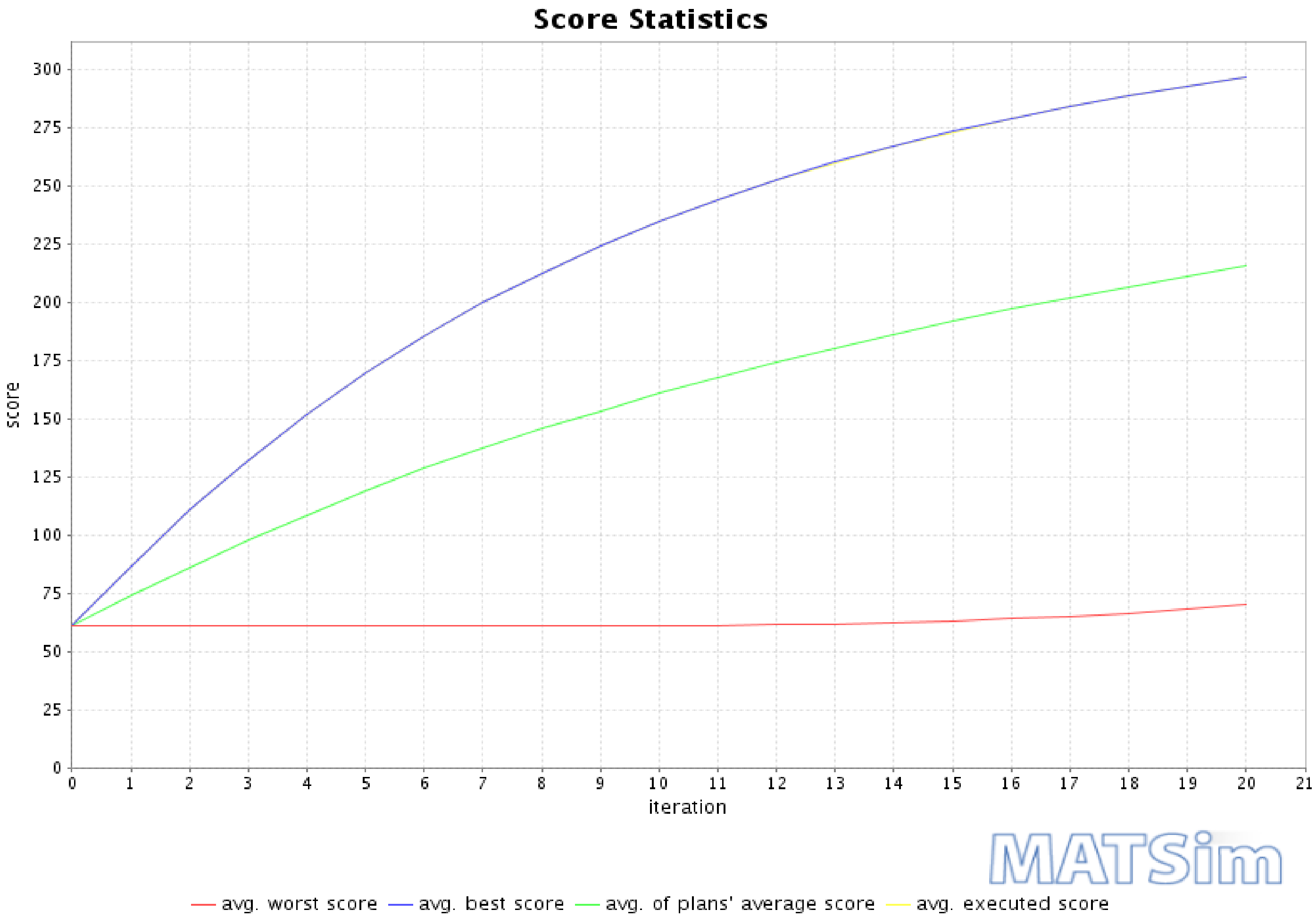
```
<plan score="238.1625409640138" selected="yes">
      <act type="home" link="176" facility="14" x="9000.0" y="
      <leg mode="car" dep_time="08:30:00" trav_time="00:34:57"
            <route dist="13000.0" trav_time="00:28:57">
                  96 86 85 84 74 64 63 62 52 42 32 22 12 2
            </route>
      </leg>
      <act type="work" link="91" facility="1" x="0.0" y="500.
      <leg mode="car" dep_time="14:34:57" trav_time="00:11:09"
            <route dist="4000.0" trav_time="00:08:53">
                  1 11 21 31 41
            </route>
      </leg>
      <act type="shopping" link="5" facility="37" x="4500.0" y
      <leg mode="car" dep_time="15:46:06" trav_time="00:13:22"
            <route dist="5000.0" trav_time="00:11:08">
                  51 52 53 54 44 43
            </route>
      </leg>
      <act type="shopping" link="14" facility="38" x="4500.0"
      <leg mode="car" dep_time="16:59:29" trav_time="00:11:08"
            <route dist="4000.0" trav_time="00:08:54">
                  53 63 73 83 93
            </route>
      </leg>
      <act type="home" link="174" facility="12" x="9000.0" y="
      <leg mode="car" dep_time="24:40:33" trav_time="00:04:26"
            <route dist="1000.0" trav_time="00:02:13">
                  94 95
            </route>
      </leg>
      <act type="home" link="176" facility="14" x="9000.0" y="
</plan>
```

What works on the small scenario works on the large-scale Zurich 10% scenario alike but is pretty time-consuming

PlanomatX
20 iter

Time ModeChoicer
5 iter

Runtime
05h 09min / iter*
4.3 sec / agent



* Without traffic assignment, 4 cores

Agenda

- Challenge and objective
- Optimization of agents' schedules
  - PlanomatX
  - **TimeModeChoicer**
  - Schedule Recycling
- Modification of the utility function
- "How do I use it?" and outlook

The TimeModeChoicer helps reduce the runtime of the PlanomatX

Current situation

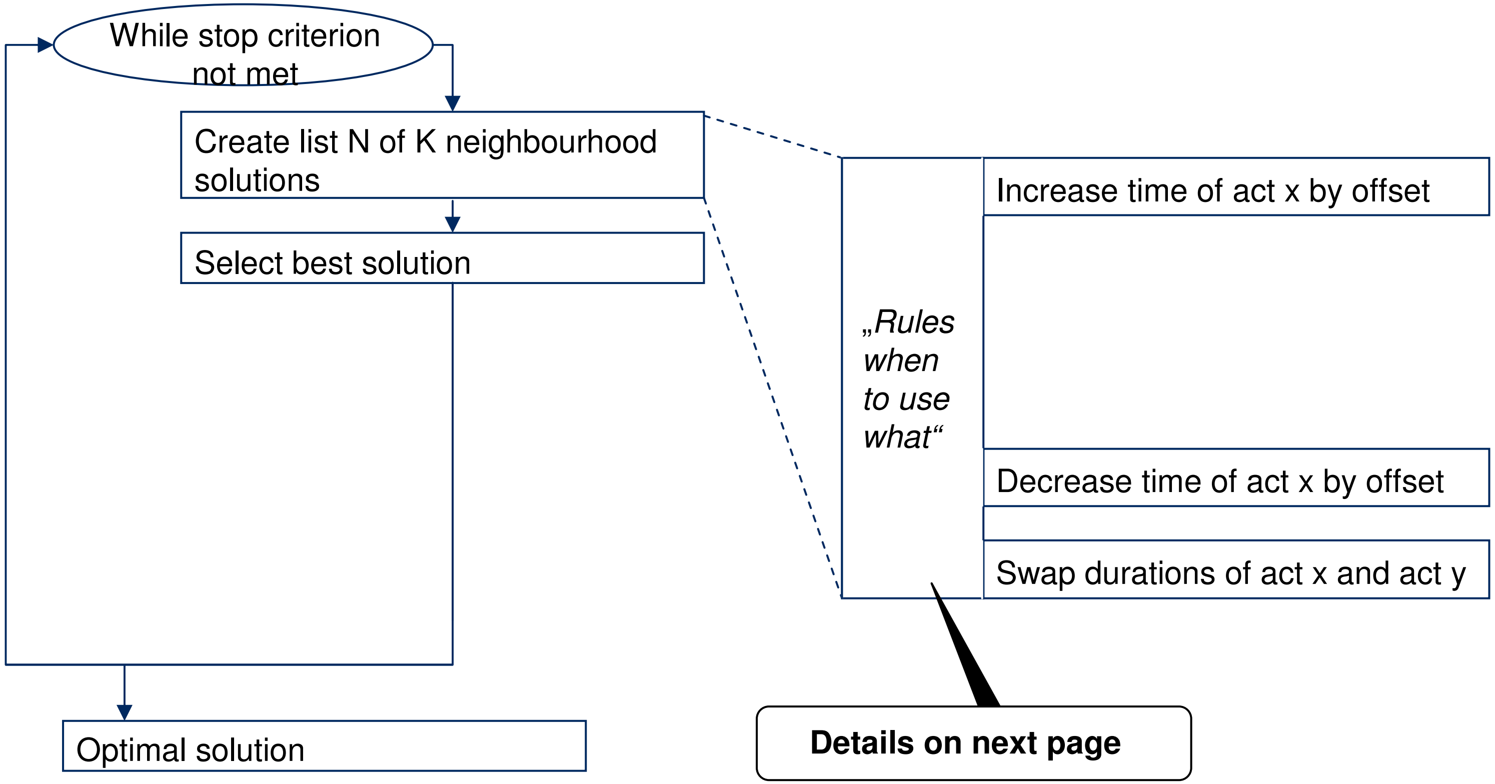
- Optimization of activity timings and mode choice accounts for significant part of overall PlanomatX runtime
- Planomat module is based upon a GA known as rather inefficient (see before)

Objective

- Optimize activity timings and mode choice more efficiently
- Test whether Tabu Search can reduce runtime

The basic process flow of the TimeModeChoicer resembles the PlanomatX flow but is simpler

Basic process flow of the TimeModeChoicer



While stop criterion not met

Create list N of K neighbourhood solutions

Select best solution

Optimal solution

„Rules when to use what“

Increase time of act x by offset

Decrease time of act x by offset

Swap durations of act x and act y

Details on next page

- **Standard settings are:**
  - **Neighbourhood size = 10**
  - **Maximum number of iterations = 30; stopp criterion = „no improvement over last 5 iterations“**
  - **Offset = 30min**

The TimeModeChoicer drops the tabu check (to save runtime) and rather employs intelligent rules to prevent from cycling

Rules of neighbourhood creation

Iteration

1st

Conduct a **„complete“ neighbourhood search** increasing the duration of every activity by a given offset time while decreasing the duration of each other activitiy by that offset time (number of neighbourhood solutions = (n-1)+(n-2)+…+2+1, where n is the length of the activity chain)

2nd onwards

Directed neighbourhood search

1/3 Further increase act duration that was increased in last move

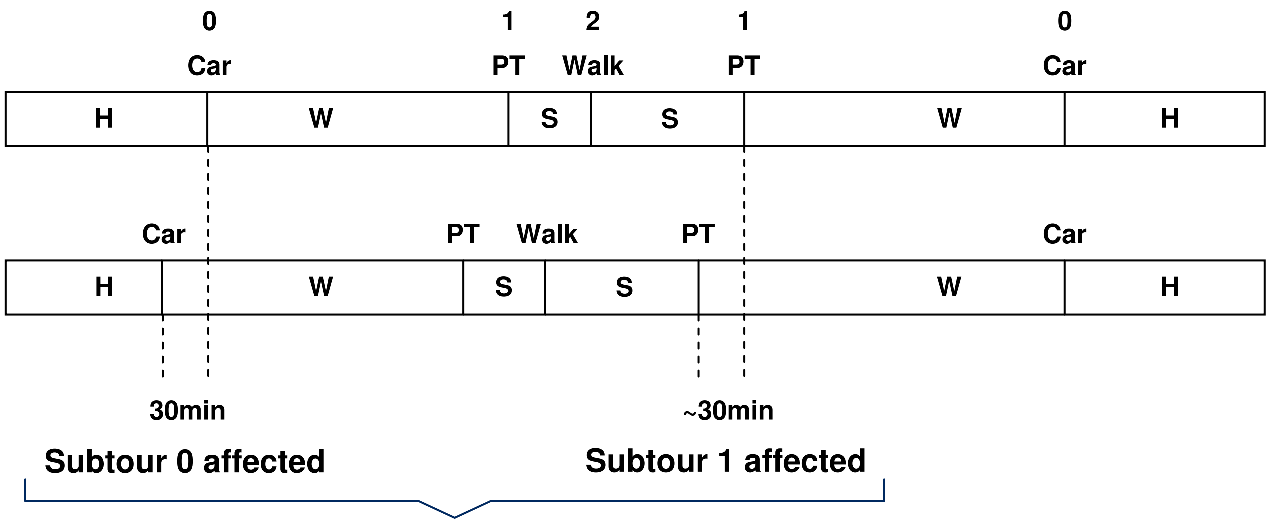
1/3 Further decrease act duration that was decreased in last move

1/3 Increase/decrease other act durations

- In addition to directed neighbourhood search, prevent algorithm from getting stuck in local optima by swapping act durations when the act duration to be decreased would fall below minimum time

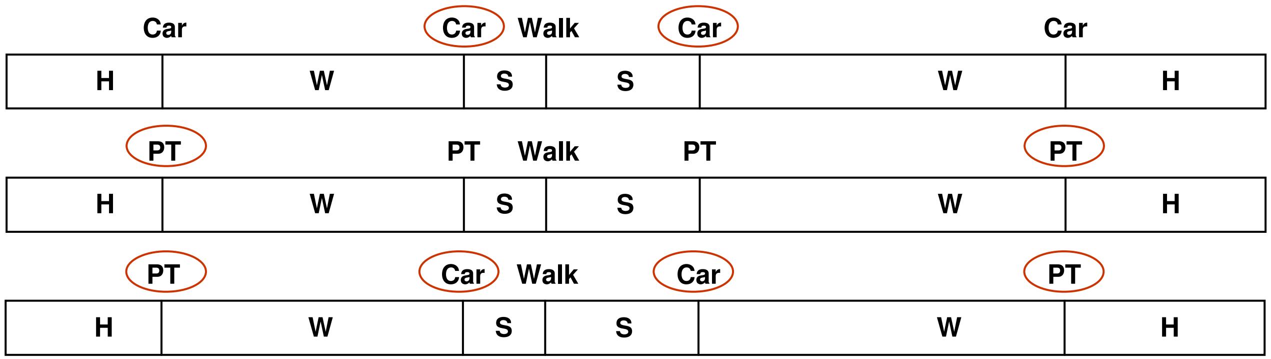
Introducing mode choice considerably enlarges the solution space compared with a simple time optimization – Example

Time optimization only



Time optimization and mode choice

- With e.g., 3 modes (car, pt, walk), there would be 3^2 = 9 possibilities
- However, walk is only tested for subtours of less than x meters of distance. Assuming that both subtours exceed the walk distance limit, the additional solution space would look like:

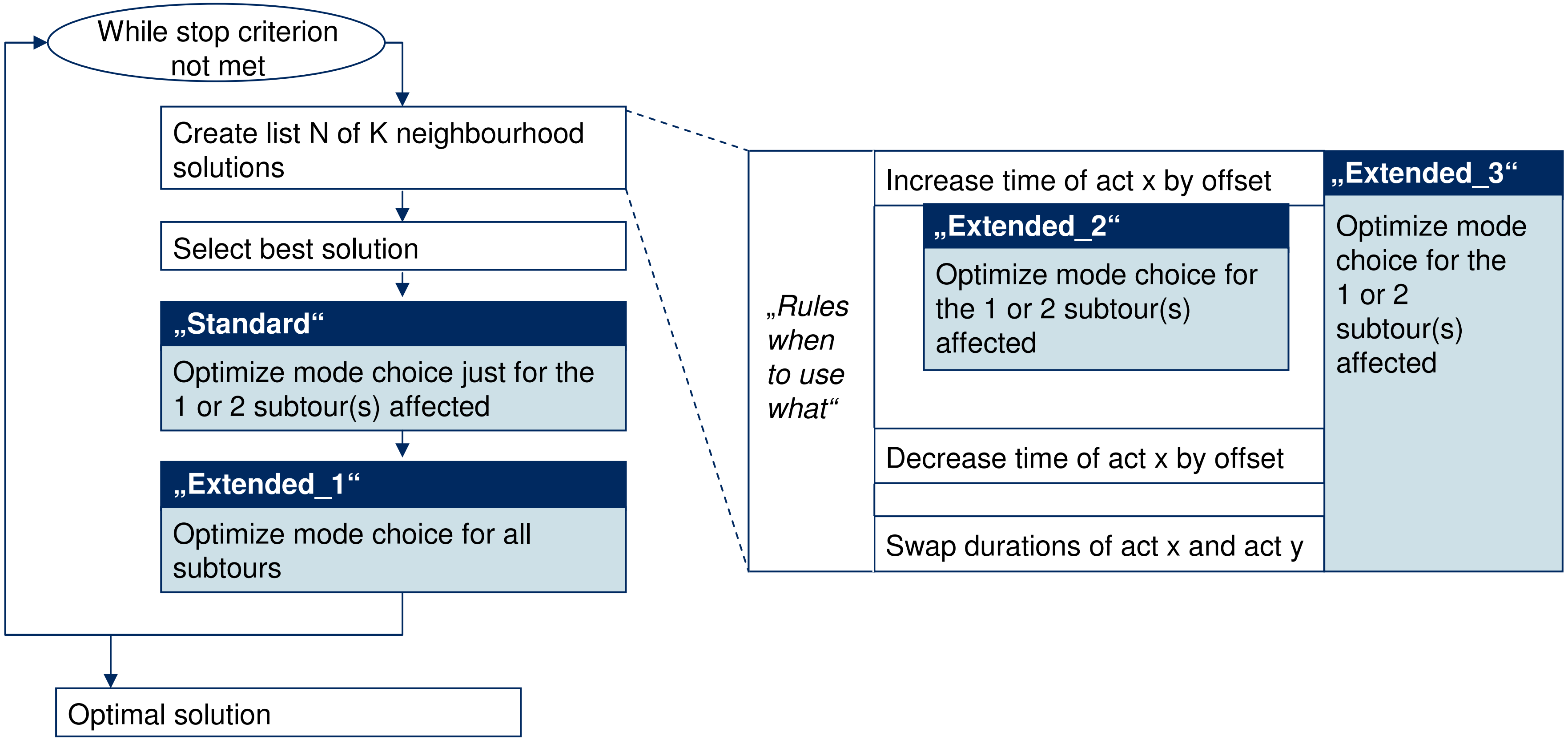


- Test all additional mode alternatives and select mode combination with highest score

The mode choice can be chosen to be included at four different process steps…

Basic process flow of the TimeModeChoicer

🟦 Newly implemented elements



While stop criterion not met

Create list N of K neighbourhood solutions

Select best solution

„Standard“

Optimize mode choice just for the 1 or 2 subtour(s) affected

„Extended_1“

Optimize mode choice for all subtours

Optimal solution

„Rules when to use what“

Increase time of act x by offset

„Extended_2“

Optimize mode choice for the 1 or 2 subtour(s) affected

Decrease time of act x by offset

Swap durations of act x and act y

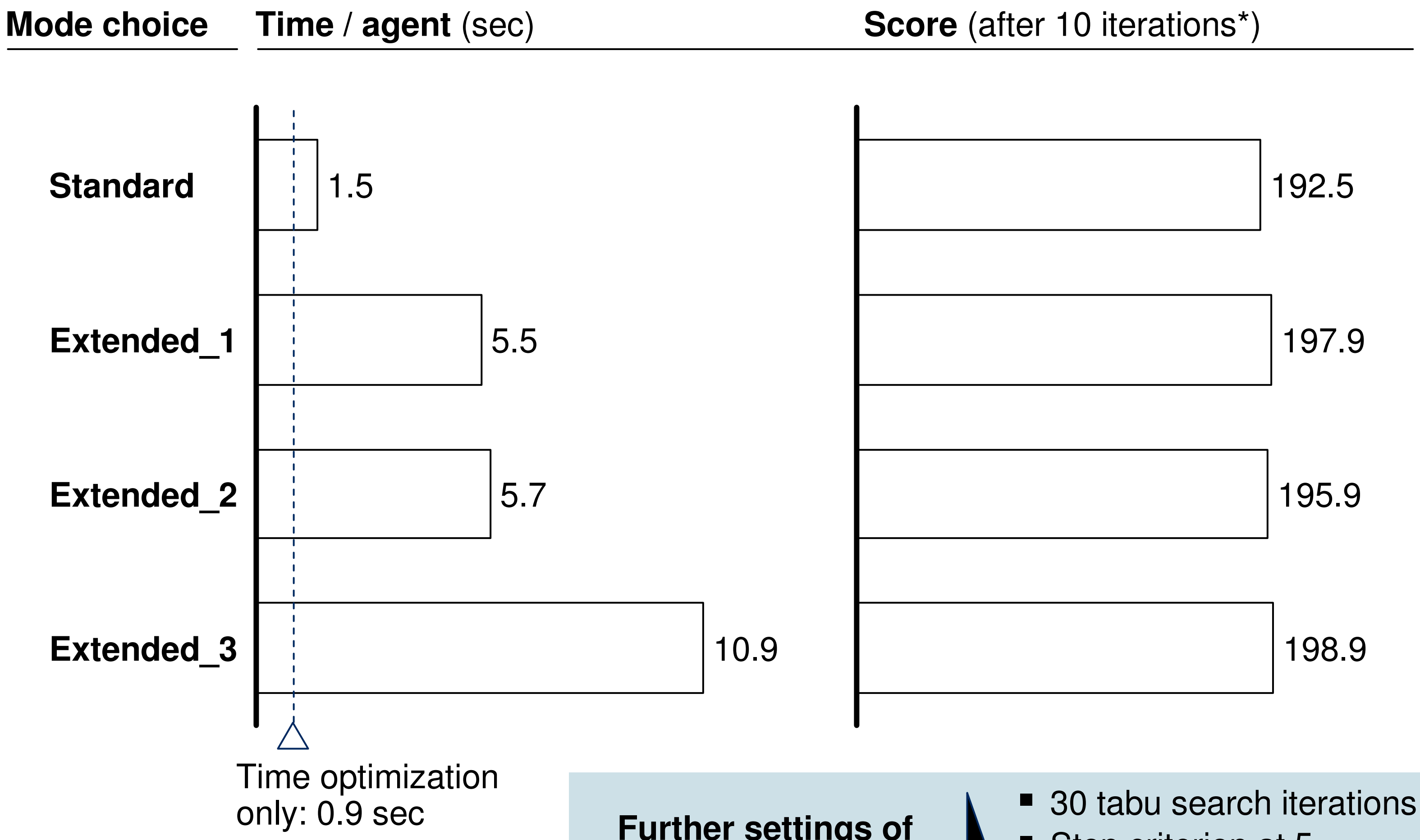
„Extended_3“

Optimize mode choice for the 1 or 2 subtour(s) affected

- **Mode choice can be optimized at various levels leading to different run times**
- **However, almost independently from the level, results are always the same**

See next page

… what has little effect on the results quality but heavy impact on the runtimes (small test scenario with about 300 agents)



Further settings of TimeModeChoicer

- 30 tabu search iterations
- Stop criterion at 5
- Neighbourhood size of 10

* Individual optimization of agents with PlanomatX18

Agenda

- Challenge and objective
- Optimization of agents' schedules
  - PlanomatX
  - TimeModeChoicer
  - **Schedule Recycling**
- Modification of the utility function
- "How do I use it?" and outlook

Schedule recycling takes opportunity of many agents aiming for equal schedule structures

Situation

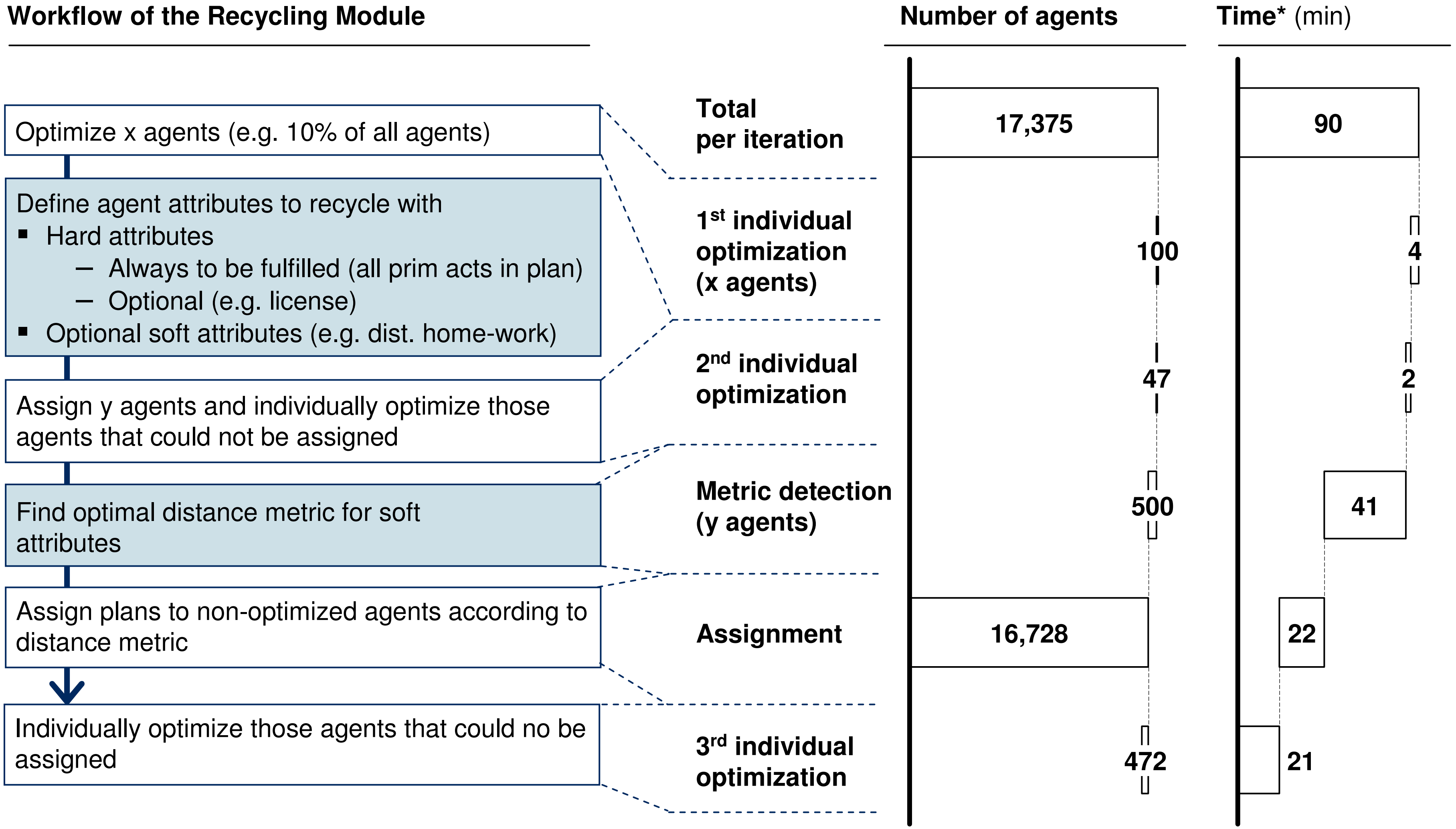
- PlanomatX takes way too long to be applied to large scale scenarios (4.3 sec / agent;
  >5h per iteration)
- Many agents features equal schedule structures after the optimization

Objective

- Further reduce the runtime
- Re-use, or „recycle“, optimized schedules of agents for other similar agents whose schedules have not been optimized yet

Schedule recycling implies to find correct spatial and socio-economic relationships of individually optimized agents with non-optimized agents

🟦 First MATSim iteration only



Workflow of the Recycling Module

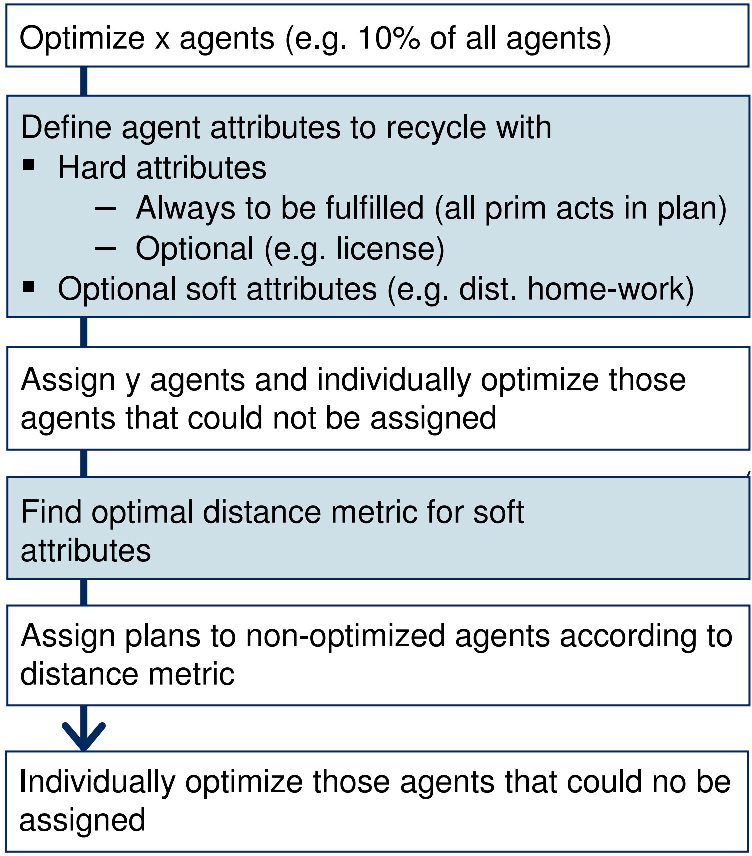
- Optimize x agents (e.g. 10% of all agents)
- Define agent attributes to recycle with
  - Hard attributes
    - Always to be fulfilled (all prim acts in plan)
    - Optional (e.g. license)
  - Optional soft attributes (e.g. dist. home-work)
- Assign y agents and individually optimize those agents that could not be assigned
- Find optimal distance metric for soft attributes
- Assign plans to non-optimized agents according to distance metric
- Individually optimize those agents that could no be assigned

	Number of agents	Time* (min)
Total per iteration	17,375	90
1st individual optimization (x agents)	100	4
2nd individual optimization	47	2
Metric detection (y agents)	500	41
Assignment	16,728	22
3rd individual optimization	472	21

* Zurich 10% scenario on satawal, 8 CPUs

The determination of the distance metric is at the core of the schedule recycling

Workflow of the Recycling Module



Optimize x agents (e.g. 10% of all agents)

Define agent attributes to recycle with
- Hard attributes
  - Always to be fulfilled (all prim acts in plan)
  - Optional (e.g. license)
- Optional soft attributes (e.g. dist. home-work)

Assign y agents and individually optimize those agents that could not be assigned

Find optimal distance metric for soft attributes

Assign plans to non-optimized agents according to distance metric

Individually optimize those agents that could no be assigned

Definition of the distance metric

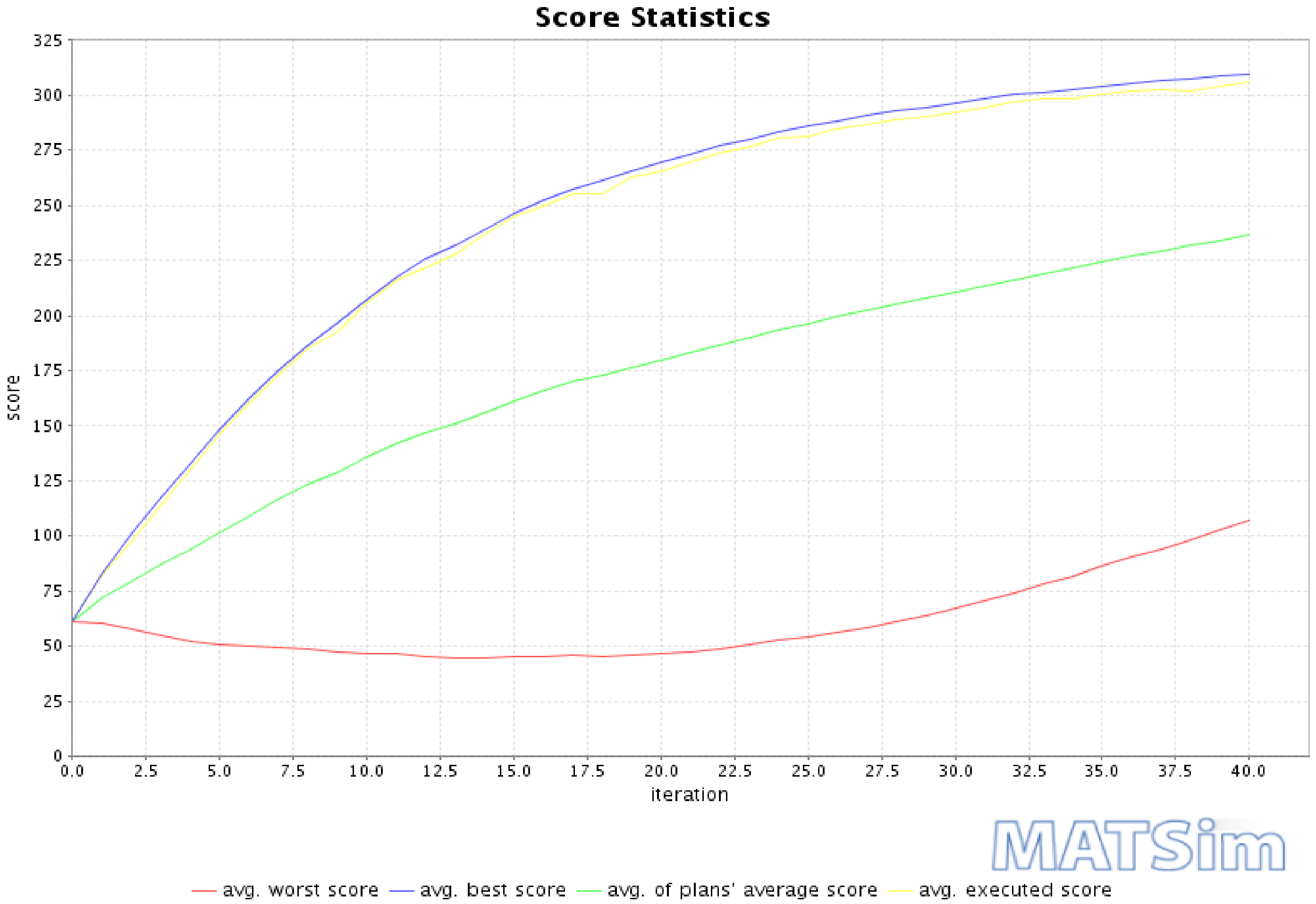
- **„Reverse clustering“**
  1. Define a default metric vector (e.g. {1,1} for a set of two attributes)
  2. Take a subset of the non-optimized agents, assign plans to these agents according to the metric vector and calculate their score
  3. Repeat step 2 for n iterations with different metric vectors
  4. Choose the metric vector for which the score becomes maximum

Like PlanomatX, schedule recycling produces nice optimization graphs (Zurich 10% scenario)…

PlanomatX
20 iter

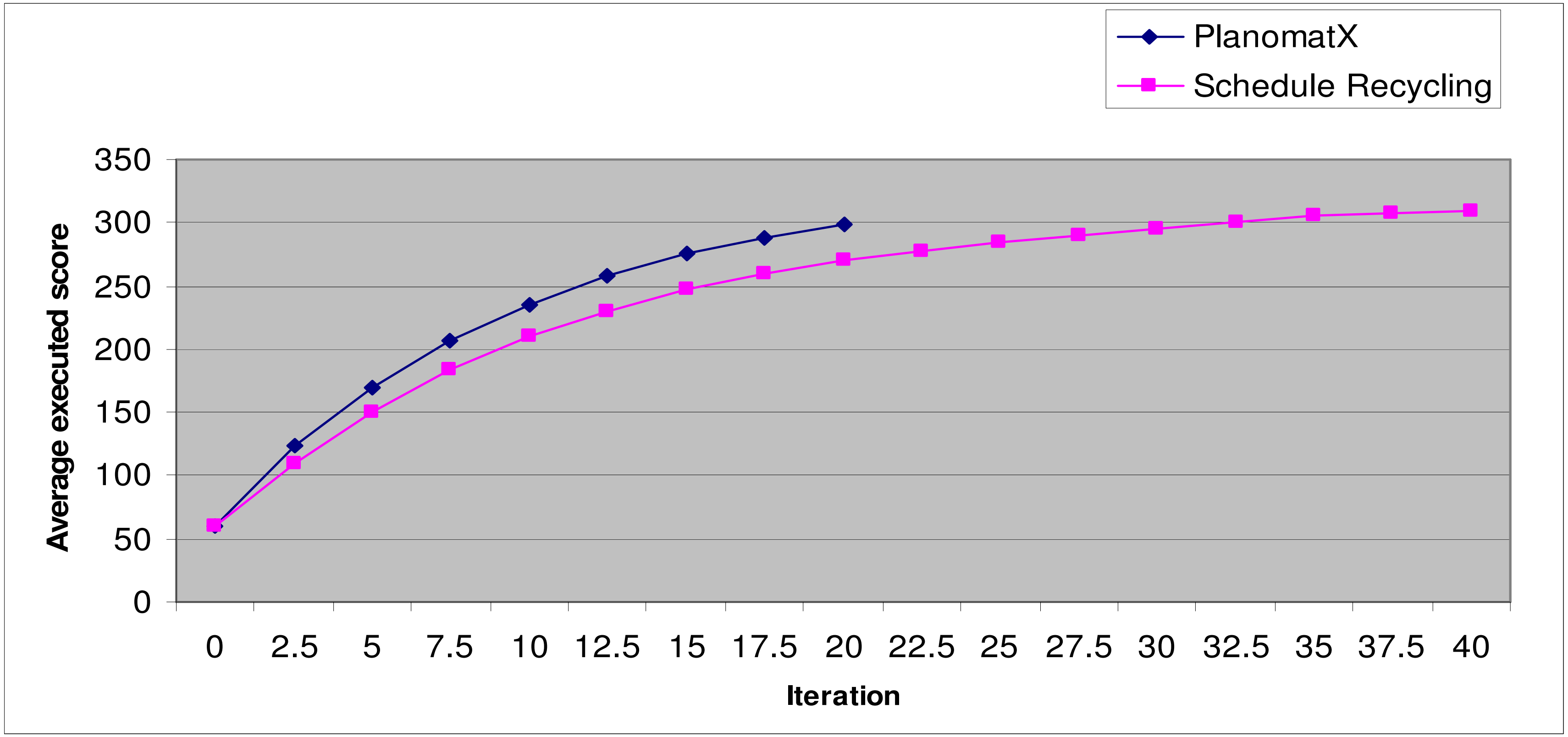
Time ModeChoicer
30 iter

Runtime
01h 58min / iter*
1.6 sec / agent



* Without traffic assignment, 4 cores

… which require more iterations to relax but at less runtime



Overall runtime

	20 iterations	40 iterations
PlanomatX	~112h	
Schedule Recycling	~51h	~100h

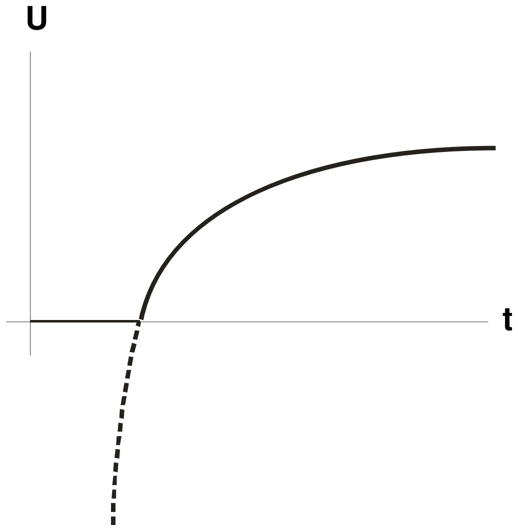
Agenda

- Challenge and objective
- Optimization of agents' schedules
  - PlanomatX
  - TimeModeChoicer
  - Schedule Recycling
- **Modification of the utility function**
- "How do I use it?" and outlook

MATSim‘s current utility function does not support the number of activities in agents‘ schedules being a dimension of the learning process

Situation

- Current utility function features a log form for the duration of activities



Complication

- The log form leads to unrealistic results when we allow for changes in the number of activities in the schedule
  - When the number of activities in a schedule is a dimension of the learning process the log form leads to a lot of very short acitivities due to the decreasing marginal utility of the log-form
  - Example: A schedule of two 30 minutes activities of a certain type is always better than a schedule of once 60 minutes of the same activity.

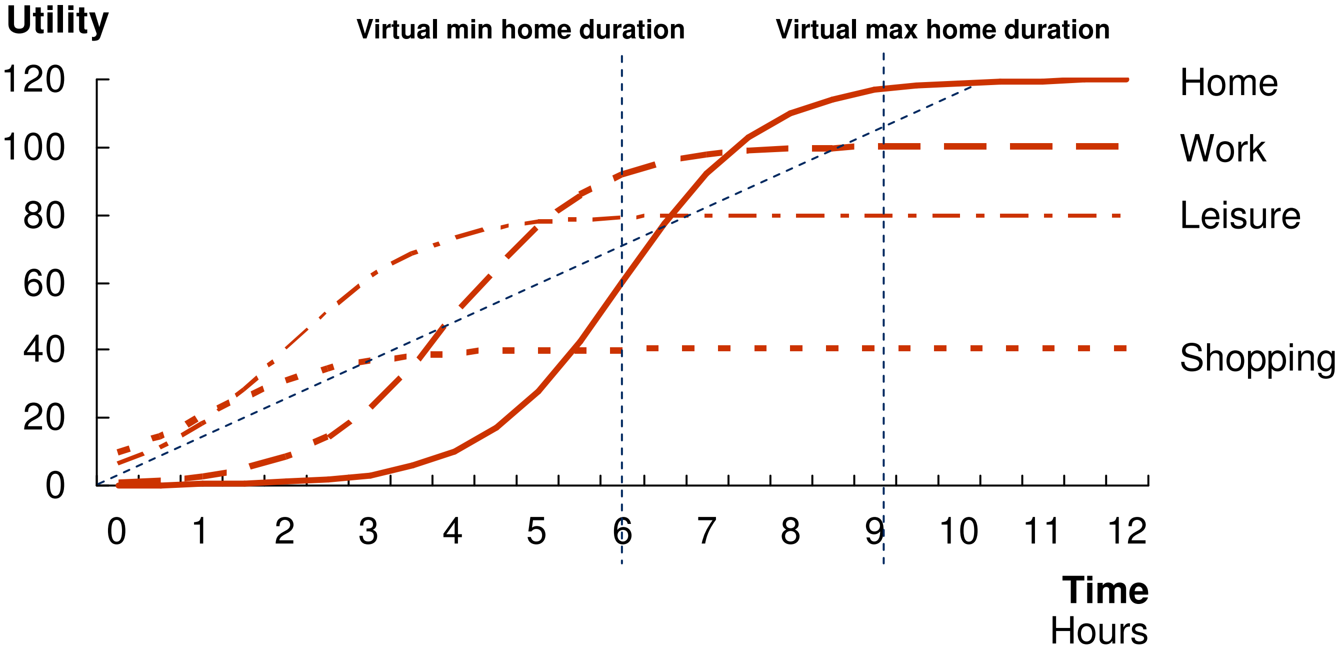
Objective

- Establish a utility function that is able to cope with the number of activities in a schedule being a dimension of the learning process

A first draft of a new scoring function has been established following Joh* (Aurora Project)

Scoring function: first test settings

$$U_a = U_a^{\min} + \frac{U_a^{\max} - U_a^{\min}}{\left(1 + \gamma_a \exp[\beta_a(\alpha_a - v_a)]\right)^{1/\gamma_a}}$$



- Allows for control of number and type of activities in optimal daily plan
- Currently minimal duration of 1hour for all activity types**, however no activity-specific minimal durations required any longer (hence no „too short“-penalty necessary)

* Chang-Hyeon Joh (2004) Measuring and Predicting Adaptation in Multidimensional Activity-Travel Patterns, *Dissertation*, Technical University of Eindhoven, Eindhoven

** Can also be set individually for each activity

Agenda

- Challenge and objective
- Optimization of agents’ schedules
  - PlanomatX
  - TimeModeChoicer
  - Schedule Recycling
- Modification of the utility function
- **“How do I use it?” and outlook**

PlanomatX config parameters

🟧 **Primary parameter**

Parameter	Standard setting	Description
🟧 ***neighbourhood_size***	10	Number of candidate solutions that are created per iteration: „*keep it rather low*“
🟧 ***max_iterations***	20	Number of iterations: „*increase this if really good results required*“
weight_change_order*	0.2	Share of neighbourhood solutions that change the order of activities of the base solution
weight_change_number*	0.6	Share of neighbourhood solutions that change the number of activities of the base solution
weight_inc_number**	0.5	Share of neighbourhood solutions for which the number of activities is increased
timer	TimeModeChoicer	Module used to optimize activity timings and mode choice: „*Planomat or TimeModeChoicer*“
final_timer	none	Module used to refine activity timings at the end of PlanomatX optimization: „*normally not required but if so take TimeOptimizerWIGIC*“
LC_mode	reducedLC	Way of location choice: „*reducedLC takes first feasible solution, fullLC optimizes*“
LC_set_size	2	Number of location choice alternatives if fullLC chosen: „*2 is probably enough, do not exceed 4*“

* weight_change_type = 1 – weight_change_order – weight_change_number

** weight_dec_number = 1 – weight_inc_number

TimeModeChoicer config parameters

🟧 **Primary parameter**

Parameter	**Standard setting**	**Description**
🟧 ***neighbourhood_size***	▪ 10	▪ Number of candidate solutions that are created per iteration: „*keep it rather low*“
🟧 ***max_iterations***	▪ 30	▪ Number of maximum iterations: „*increase this if really good results required*“
🟧 ***stop_criterion***	▪ 5	▪ Stop of optimization if no improvement over last <stop_criterion> iterations: „*increase this if really good results required*“
🟧 ***offset***	▪ 1800 seconds	▪ Duration by which an activity is increased/decreased during a move: „*chose something between 900 and 3600*“
minimum_time	▪ 3600 seconds	▪ Minimum time of an activity: „*Should be more than 900 seconds*“
possible_modes	▪ car, pt, walk	▪ Available modes
maximum_walking_distance	▪ 2000 meters	▪ Limits the solution space of the mode choice to speed up the calculation
mode_choice	▪ standard	▪ Defines the level of mode choice (see page 19)

Schedule recycling config parameters

🟧 **Primary parameter**

Parameter	Standard setting	Description
iterations	20	Size of choice set from which the parameter setting is selected that generates the best score: *„20 seems very reasonable, no need to change“*
noOfTestAgents	–	Number of agents that are optimized individually at the beginning of each iteration: *„set to about 20% of all agents but not more than 100“*
noOfAgents	–	Number of agents that are assigned with one of the plans of the test agents to evaluate the distance metric: *„set to about 30% of all agents but not more than 500“*
Soft coefficients		
primActsDistance	yes	Distance between the agent‘s primary activities
homeLocationDistance	no	Currently, these parameters have neither an effect on a schedule‘s utility nor do they limit the choice of a schedule structure. In the long run, this may though change and, therefore, these parameters are ready to be considered.
sex	no	
age	no	
license	no	
carAvailability	no	
employed	no	

The presented algorithms have not been committed to the core yet but will so soon

- Outlook:
  - Full completion of algorithms and commitment to the MATSim core
  - Enhancement and empirical estimation of the utility function
- Contact: mfeil@student.ethz.ch

BACKUP

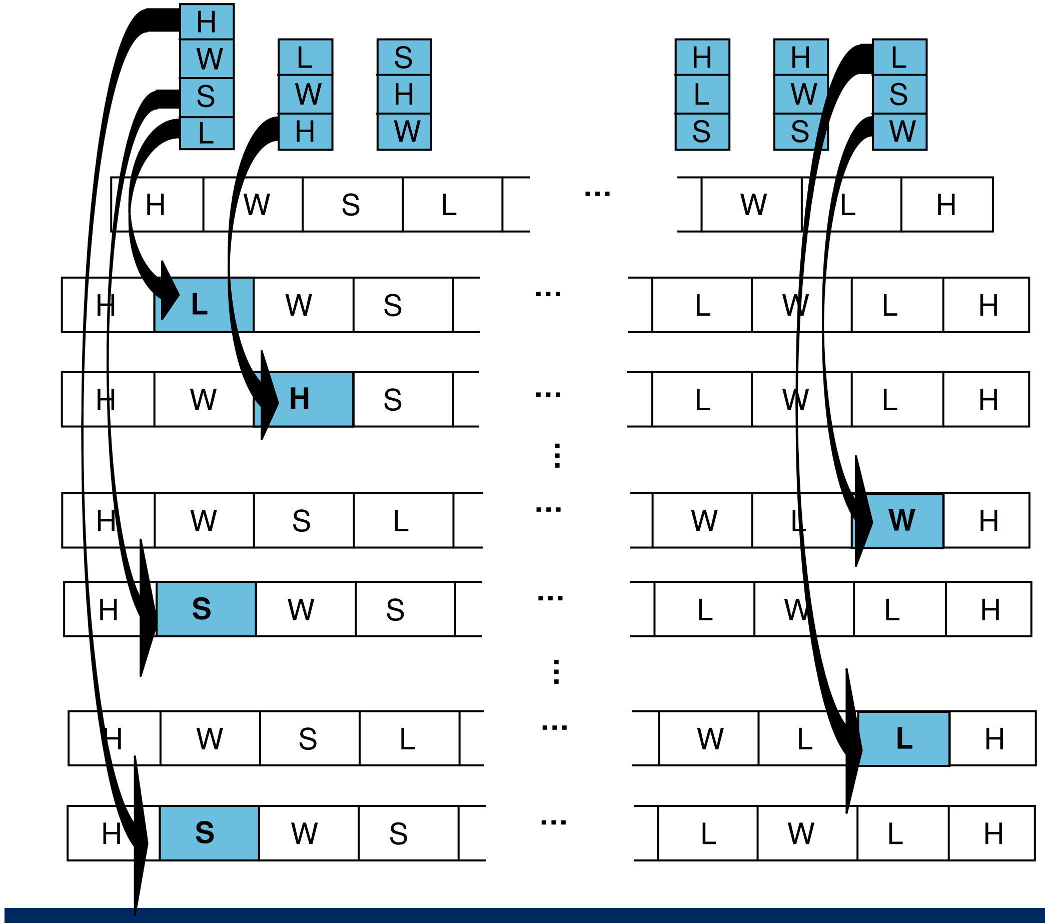
Details of „change number“ sub-algorithm – increasing the number of acts

□□□□□□□□□ Schedule

▨ Acts inserted

BACKUP

Initial random allocation of act types to cycling schedule positions



H W S L | L W H | S H W | … | H L S | H W S | L S W

H W S L … W L H

H **L** W S … L W L H

H W **H** S … L W L H

⋮

H W S L … W L **W** H

H **S** W S … L W L H

⋮

H W S L … W L **L** H

H **S** W S … L W L H

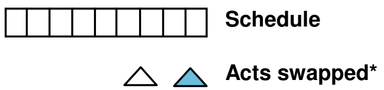
Key features

- At initial step, each „gap“ is provided with a list of randomly ordered act types
  - First gap is provided with all existing act types (read from config file)
  - All other gap lists reduced by act type of act „behind“ the gap (avoids creating equal new plans)
- Acts are inserted „cycling“ through the schedule
- Maximum number of insertions is s_{max} = t + (t-1) * (n-2) , where t is the number of act types existing and n the number of acts of the plan
- If the number of allocated neighbourhood fields is higher than the number of possible insertions s_{max}, the algorithm fills the remaining fields with the default plan from the previous iteration

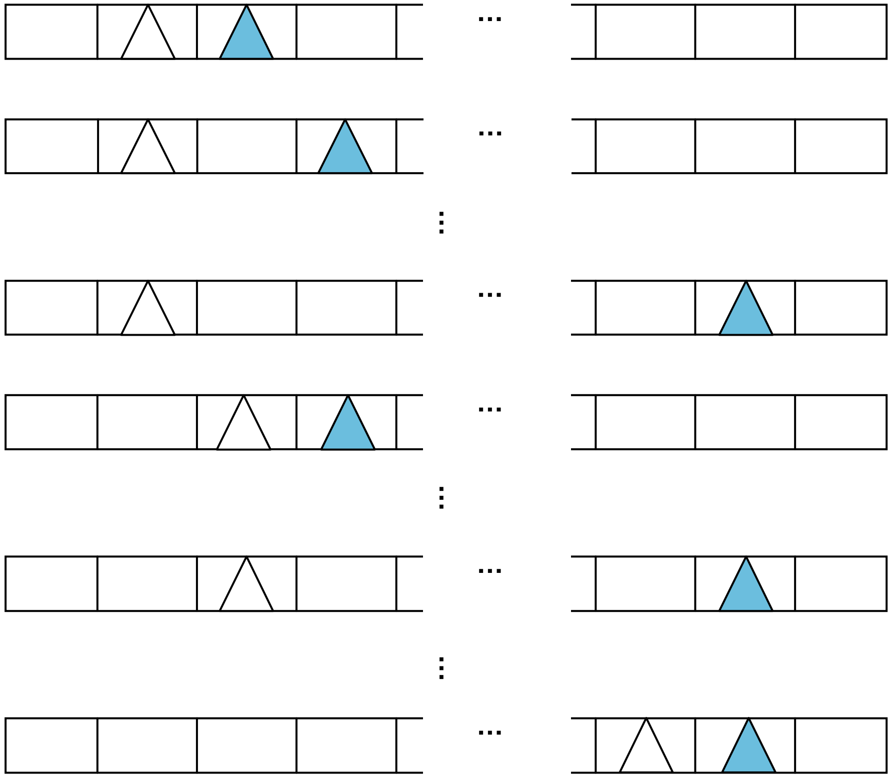
Details of „change order“ sub-algorithm

A

BACKUP



Two nested loops to select acts to be swapped



Key features

- Two nested loops to select acts to be swapped
  - First and last act remain unchanged
  - Acts are swapped only if they do not have the same type as the swap would lead to the same plan, otherwise
- Maximum number of swaps is s_{max} = (n-3)+(n-4)+ … + (n-(n-1)), where n is the number of acts of the plan
- If the number of allocated neighbourhood fields is higher than the number of possible swaps s_{max}, the algorithm fills the remaining fields with the default plan from the previous iteration

* If they do not have the same type