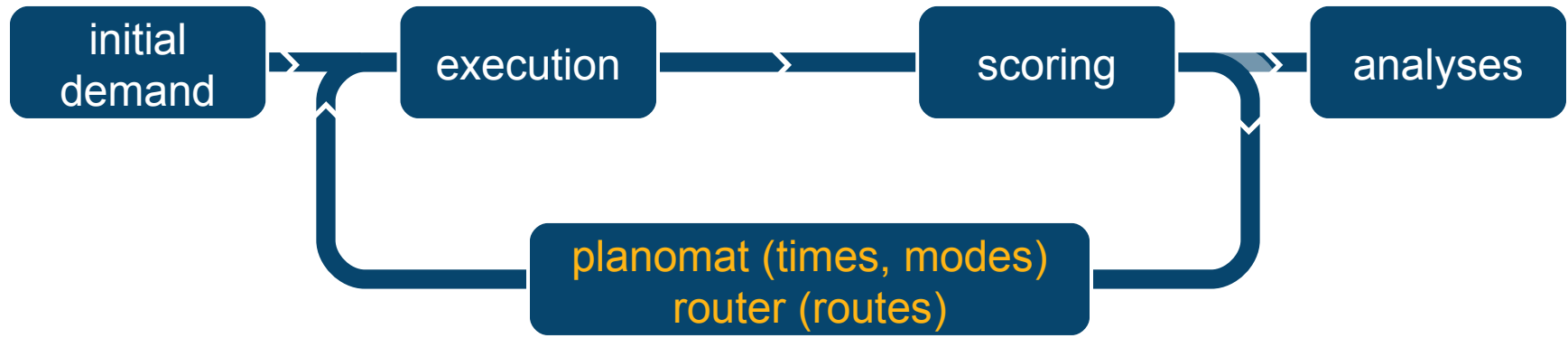


# Quick computation of "triple convergence" in MATSim - method and preliminary results

# Research objective

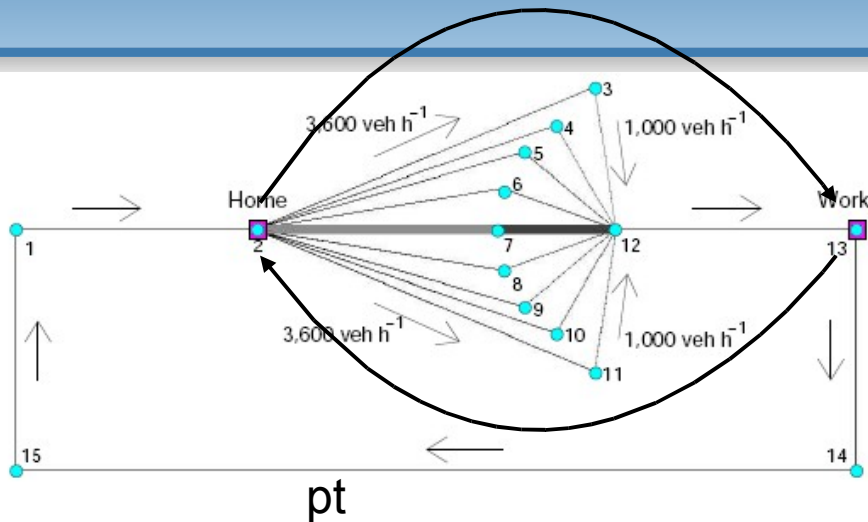
- Triple Convergence (Downs, 2004)
  - routes
  - times
  - modes
  - (activity pattern)
  - (locations with short-term flexibility)
- Quick computation:
  - Find stationary state with as few as possible iterations, while maintaining stability.
  - Have best-response modules for each of the travel-behavior aspects the agents may adapt.
  - That is, couple agent-based demand optimization to iterative demand simulation.

# MATSim Setup

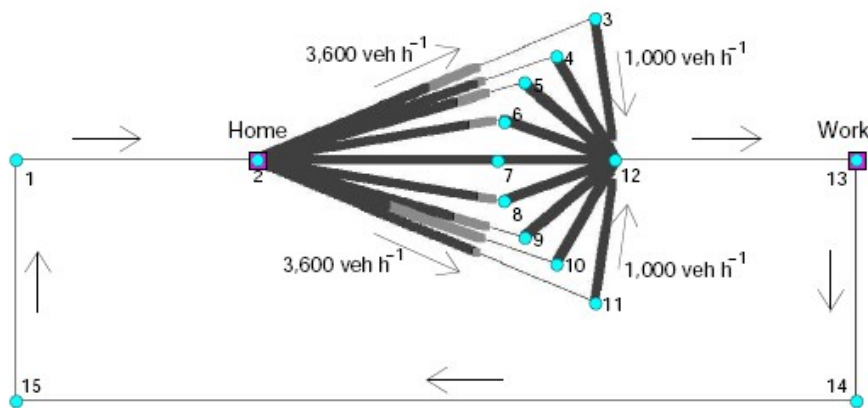


- *planomat*: simultaneous optimization of activity times and modes, while keeping routes fixed
  - times: start time of activity plan, duration of each activity
  - modes: subtour-based, alternatives: “car” and “pt”
- *router*: find time-of-day dependent shortest path, while keeping departure times fixed

# Scenario: equil-test, with “pt” alternative



(a) Before replanning.



(b) After replanning.

“pt” alternative:  $t_{\text{travel,pt}} = x * t_{\text{travel,car}}$

Expectations:

- There will be car as well as “pt” plans with similar scores.
- Agents will choose car mode for the h-w-h tour despite longer trip travel time in the morning.
- Strategies:
  - 80% SelectExpBeta
  - 10% planomat
  - 10% ReRoute

# Planomat implementation (1)

```
package org.matsim.planomat;

public class PlanOptimizeTimes implements PlanAlgorithm {

    private LegTravelTimeEstimator legTravelTimeEstimator = null;

    public PlanOptimizeTimes(final LegTravelTimeEstimator legTravelTimeEstimator) {

        this.legTravelTimeEstimator = legTravelTimeEstimator;

    }

    public void run(final Plan plan) {

        ...

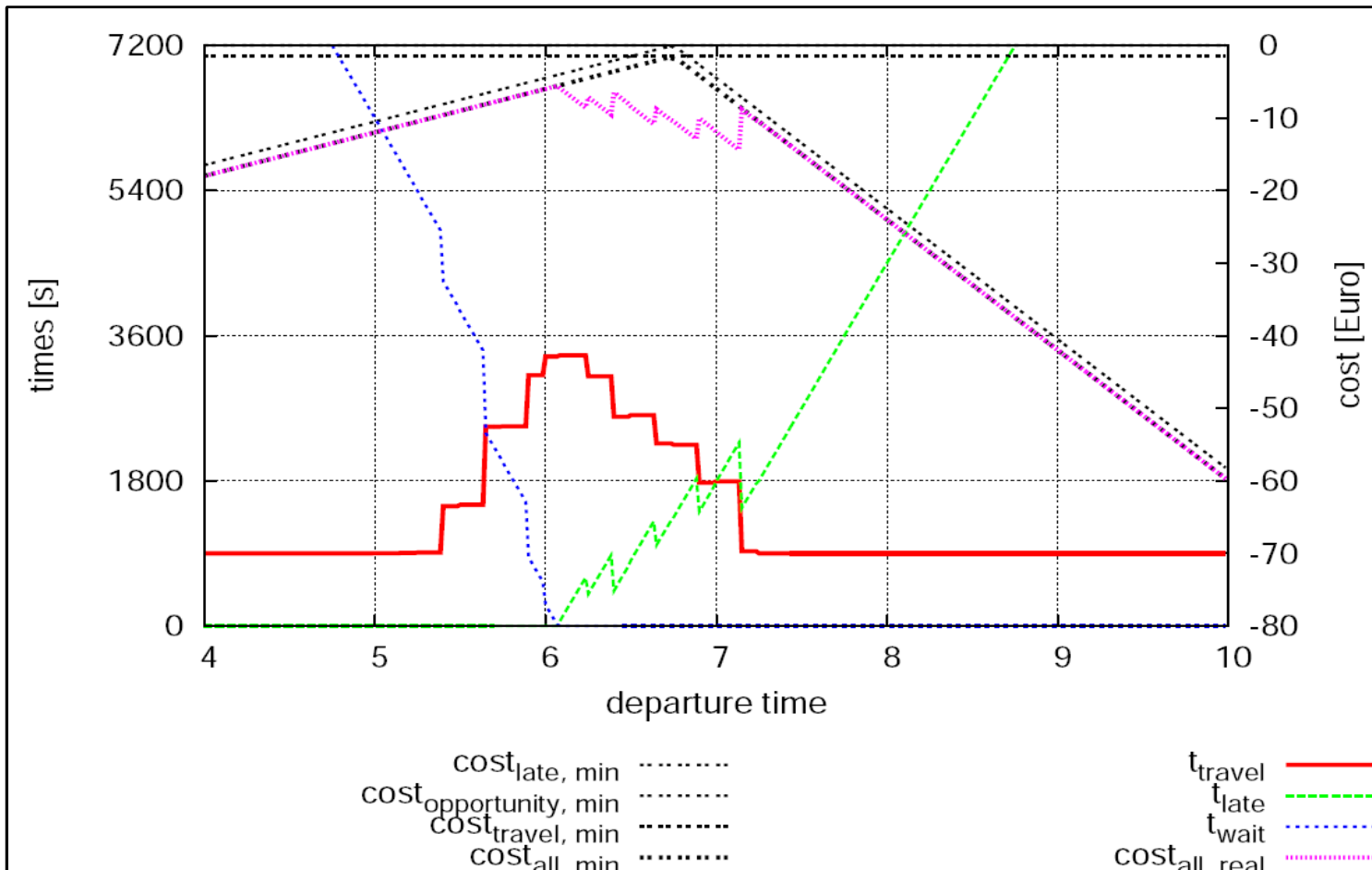
    }

}
```

# Planomat implementation (1) - leg travel time estimation

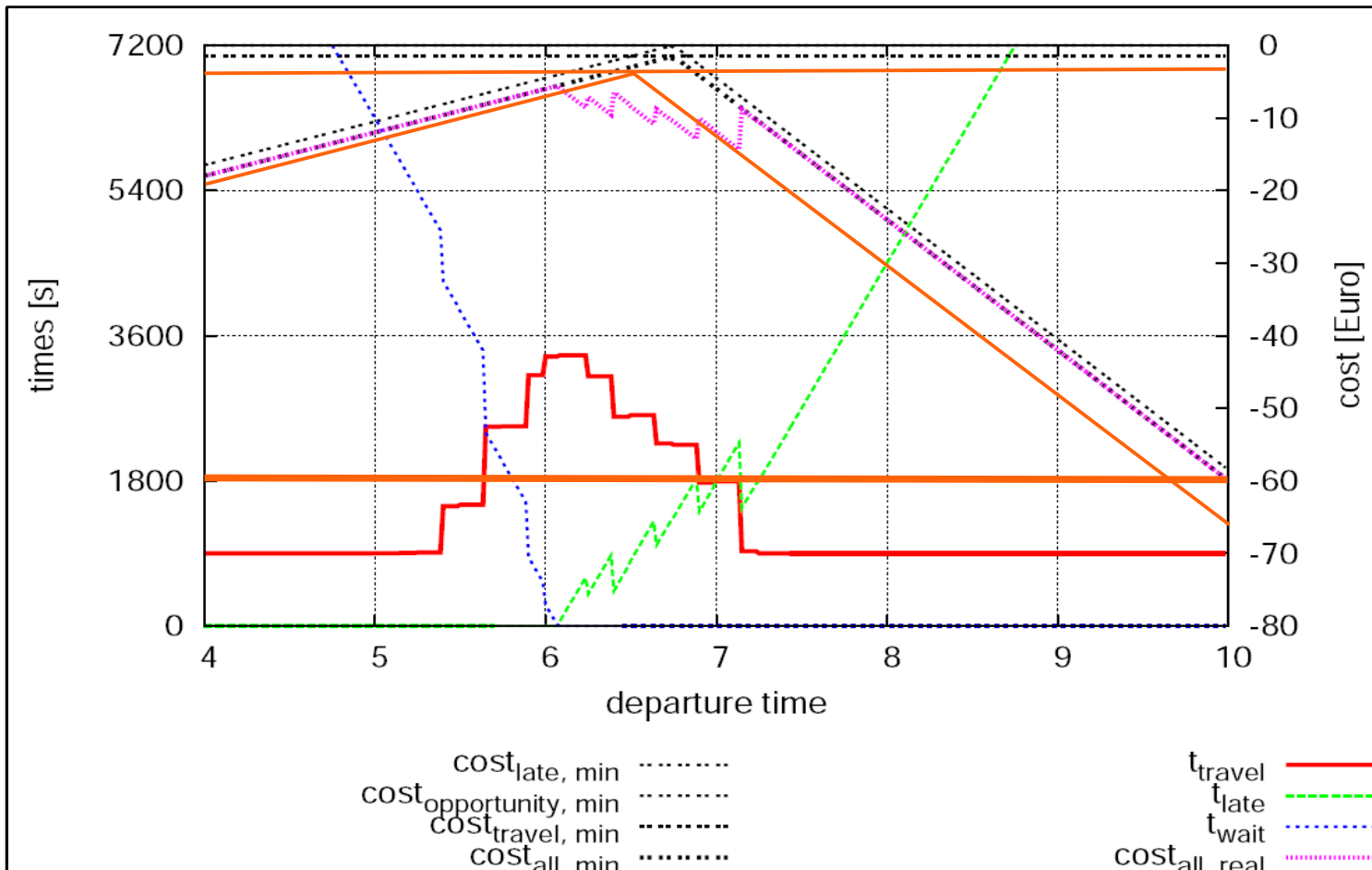
```
package org.matsim.planomat;  
  
public class PlanOptimizeTimes implements PlanAlgorithm {  
  
    private LegTravelTimeEstimator legTravelTimeEstimator = null;  

```



# Planomat implementation (1) - leg travel time estimation

```
package org.matsim.planomat;  
  
public class PlanOptimizeTimes implements PlanAlgorithm {  
  
    private LegTravelTimeEstimator legTravelTimeEstimator = null;
```



# Planomat implementation (2)

```

public class PlanOptimizeTimes implements PlanAlgorithm {

    ...

    public void run(final Plan plan) {

        String optiToolboxName = Gbl.getConfig().planomat().getOptimizationToolbox();
        if (optiToolboxName.equals(PlanomatConfigGroup.OPTIMIZATION_TOOLBOX_JGAP)) {

            PlanAnalyzeSubtours planAnalyzeSubtours = new PlanAnalyzeSubtours();
            planAnalyzeSubtours.run(plan);

            org.jgap.Configuration jgapConfiguration = this.initJGAPConfiguration();

            IChromosome sampleChromosome =
                this.initSampleChromosome(planAnalyzeSubtours, jgapConfiguration);
            jgapConfiguration.setSampleChromosome(sampleChromosome);

            ScoringFunction sf =
                Gbl.getConfig().planomat().getScoringFunctionFactory().getNewScoringFunction(plan);

            PlanomatFitnessFunctionWrapper fitnessFunction = new PlanomatFitnessFunctionWrapper(
                sf,
                plan,
                this.legTravelTimeEstimator,
                planAnalyzeSubtours );

            Genotype population = null;
            jgapConfiguration.setFitnessFunction( fitnessFunction );
            population = Genotype.randomInitialGenotype( jgapConfiguration );
            population.evolve( Gbl.getConfig().planomat().getJgapMaxGenerations() );
            IChromosome fittest = population.getFittestChromosome();
            this.writeChromosome2Plan(fittest, plan, planAnalyzeSubtours );

        }
    }
}

```

# Planomat implementation (2) - subtour identification, GA individual encoding

```

public class PlanOptimizeTimes implements PlanAlgorithm {
    ...

    public void run(final Plan plan) {
        h-w-h h-w-h-s-h h-l-w-s-w-w-w-h-l-h
        if (optiToolboxName.equals(PlanomatConfigGroup.OPTIMIZATION_TOOLBOX_JGAP)) {

            PlanAnalyzeSubtours planAnalyzeSubtours = new PlanAnalyzeSubtours();
            analyzeSubtours = planAnalyzeSubtours.analyzeSubtours(
                gap.Configuration, chromosome sampler, this.initSampler(
                    configuration, chromosome sampler, scoringFunction,
                    Gbl.getConfig().getScoringFunctionWrapper(
                        PlanomatFitnessFunction, sf, plan, this.logger,
                        planAnalyzeSubtours));

            Genotype population = jgapConfiguration.getInitialGenotype();
            population.evolve(jgapConfiguration.getEvolutionParameters());
            IChromosome fittest = population.getFittestChromosome();
            this.writeChromosome2Plan(fittest, plan, planAnalyzeSubtours);
        }
    }
}

```

h-w-h

$t_{start,plan}$
$t_{dur,w}$
mode <sub>0</sub>

h-w-h-s-h

$t_{start,plan}$
$t_{dur,w}$
$t_{dur,h}$
$t_{dur,s}$
mode <sub>0</sub>
mode <sub>1</sub>

h-l-w-s-w-w-w-h-l-h

$t_{start,plan}$	$t_{dur,w}$	mode <sub>2</sub>
$t_{dur,l}$	$t_{dur,h}$	mode <sub>3</sub>
$t_{dur,w}$	$t_{dur,l}$	mode <sub>4</sub>
$t_{dur,s}$	$t_{dur,s}$	
$t_{dur,w}$	mode <sub>0</sub>	
$t_{dur,w}$	mode <sub>1</sub>	

# Planomat implementation (2)

```

public class PlanOptimizeTimes implements PlanAlgorithm {

    ...

    public void run(final Plan plan) {

        String optiToolboxName = Gbl.getConfig().planomat().getOptimizationToolbox();
        if (optiToolboxName.equals(PlanomatConfigGroup.OPTIMIZATION_TOOLBOX_JGAP)) {

            PlanAnalyzeSubtours planAnalyzeSubtours = new PlanAnalyzeSubtours();
            planAnalyzeSubtours.run(plan);

            org.jgap.Configuration jgapConfiguration = this.initJGAPConfiguration();

            IChromosome sampleChromosome =
                this.initSampleChromosome(planAnalyzeSubtours, jgapConfiguration);
            jgapConfiguration.setSampleChromosome(sampleChromosome);

            ScoringFunction sf =
                Gbl.getConfig().planomat().getScoringFunctionFactory().getNewScoringFunction(plan);

            PlanomatFitnessFunctionWrapper fitnessFunction = new PlanomatFitnessFunctionWrapper(
                sf,
                plan,
                this.legTravelTimeEstimator,
                planAnalyzeSubtours );

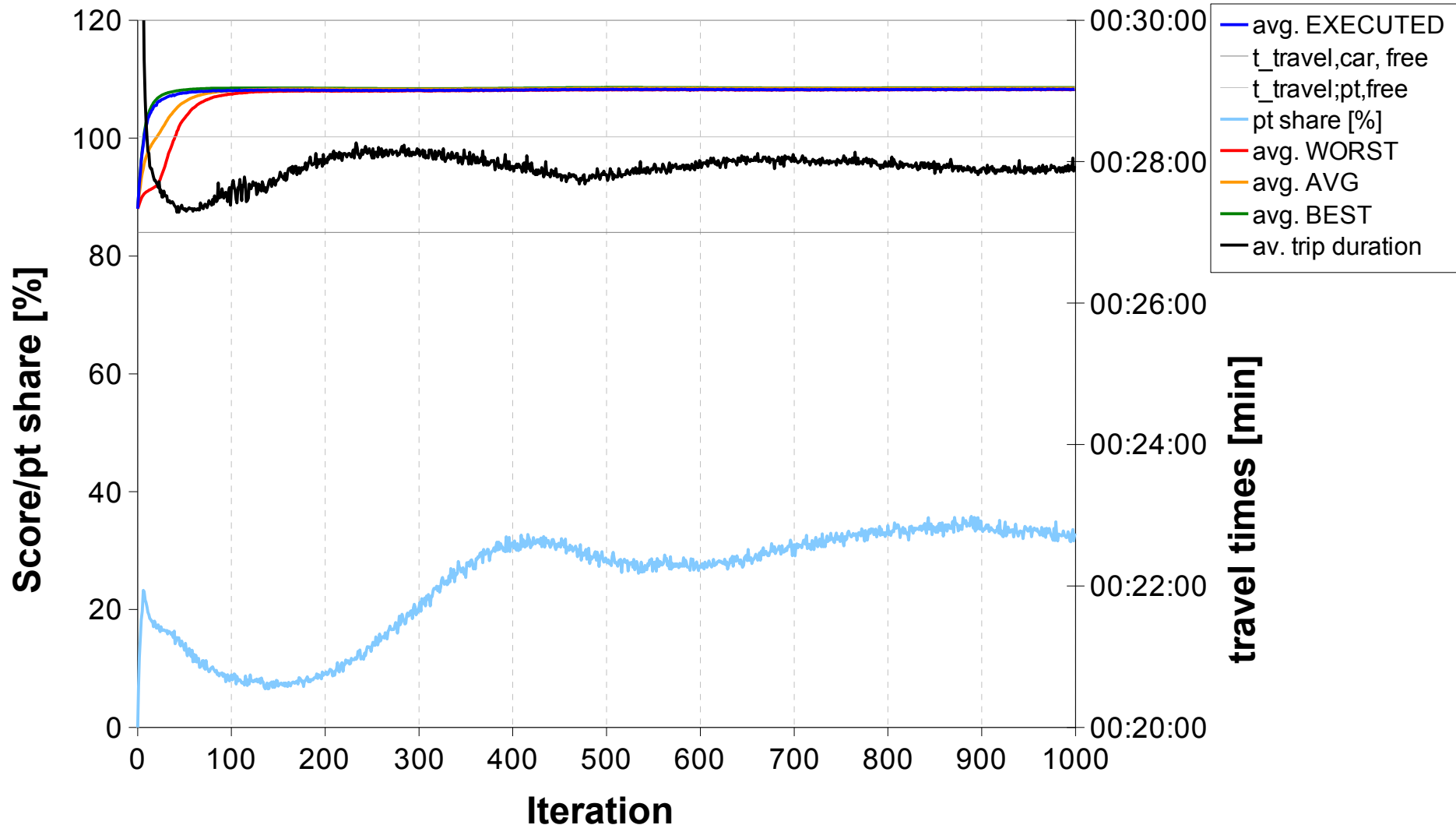
            Genotype population = null;
            jgapConfiguration.setFitnessFunction( fitnessFunction );
            population = Genotype.randomInitialGenotype( jgapConfiguration );
            population.evolve( Gbl.getConfig().planomat().getJgapMaxGenerations() );
            IChromosome fittest = population.getFittestChromosome();
            this.writeChromosome2Plan(fittest, plan, planAnalyzeSubtours );

        }
    }
}

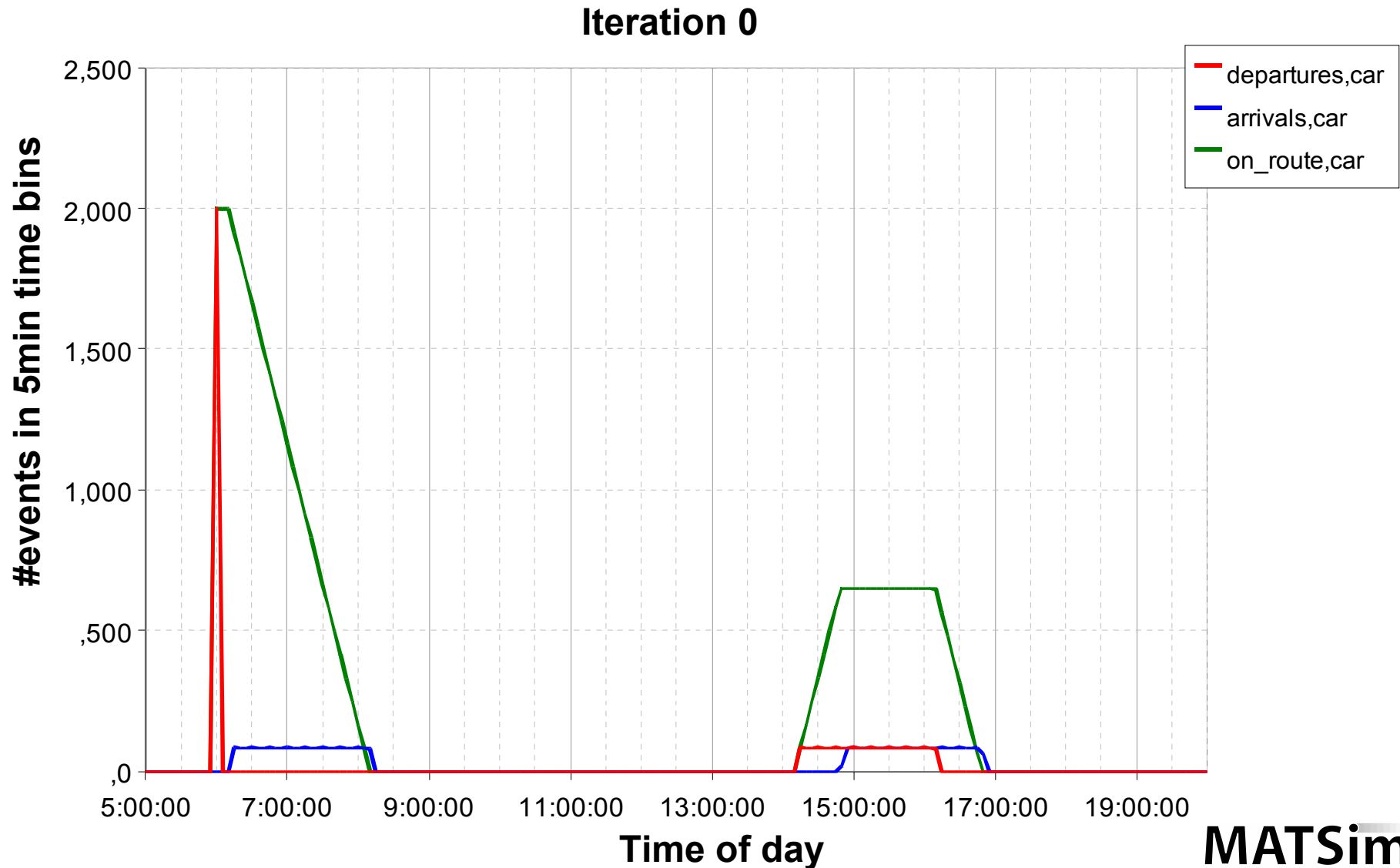
```

# Preliminary results

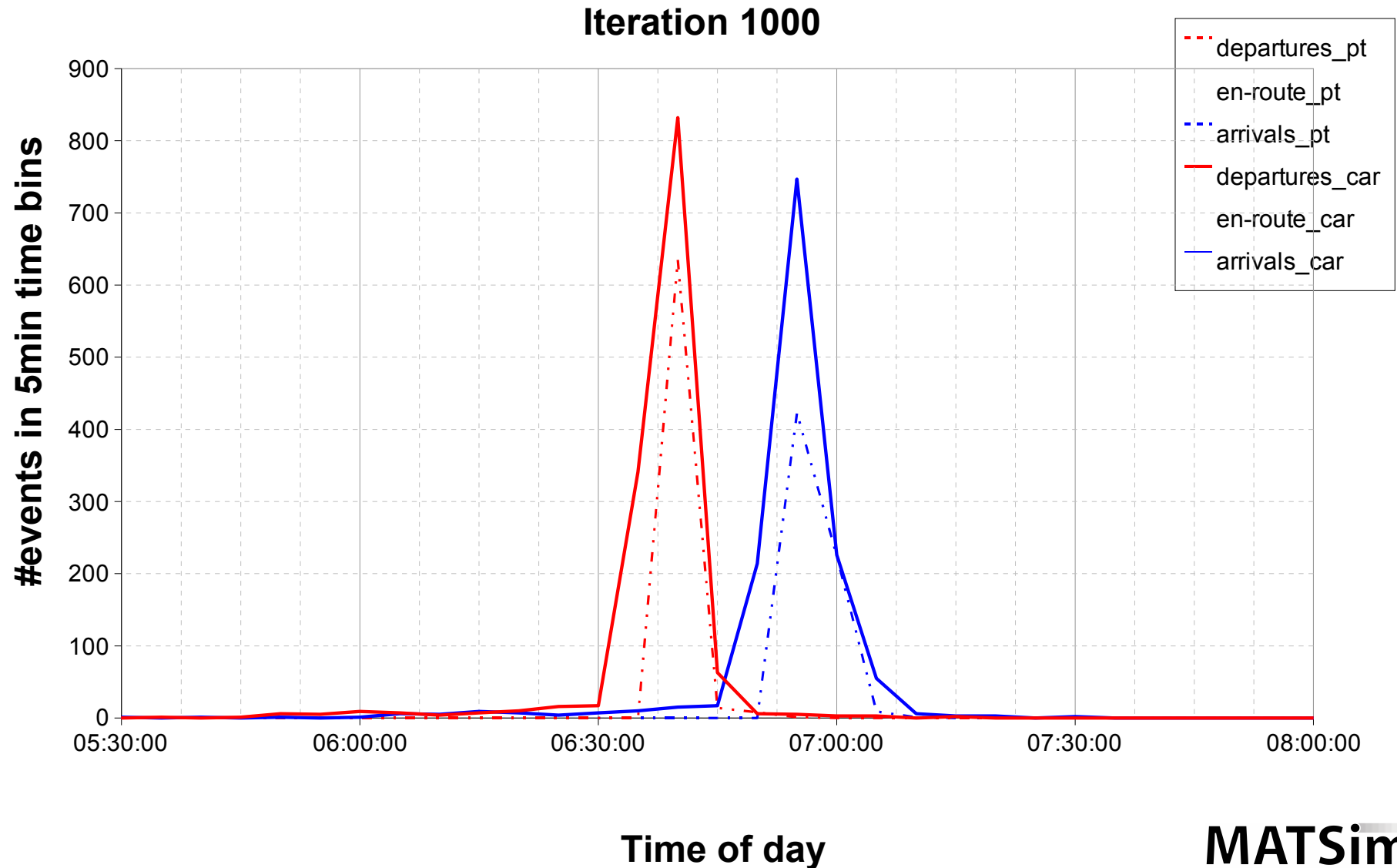
planomat and routes, ptFreespeedFactor = 1.05



planomat and routes, ptFreespeedFactor = 1.05

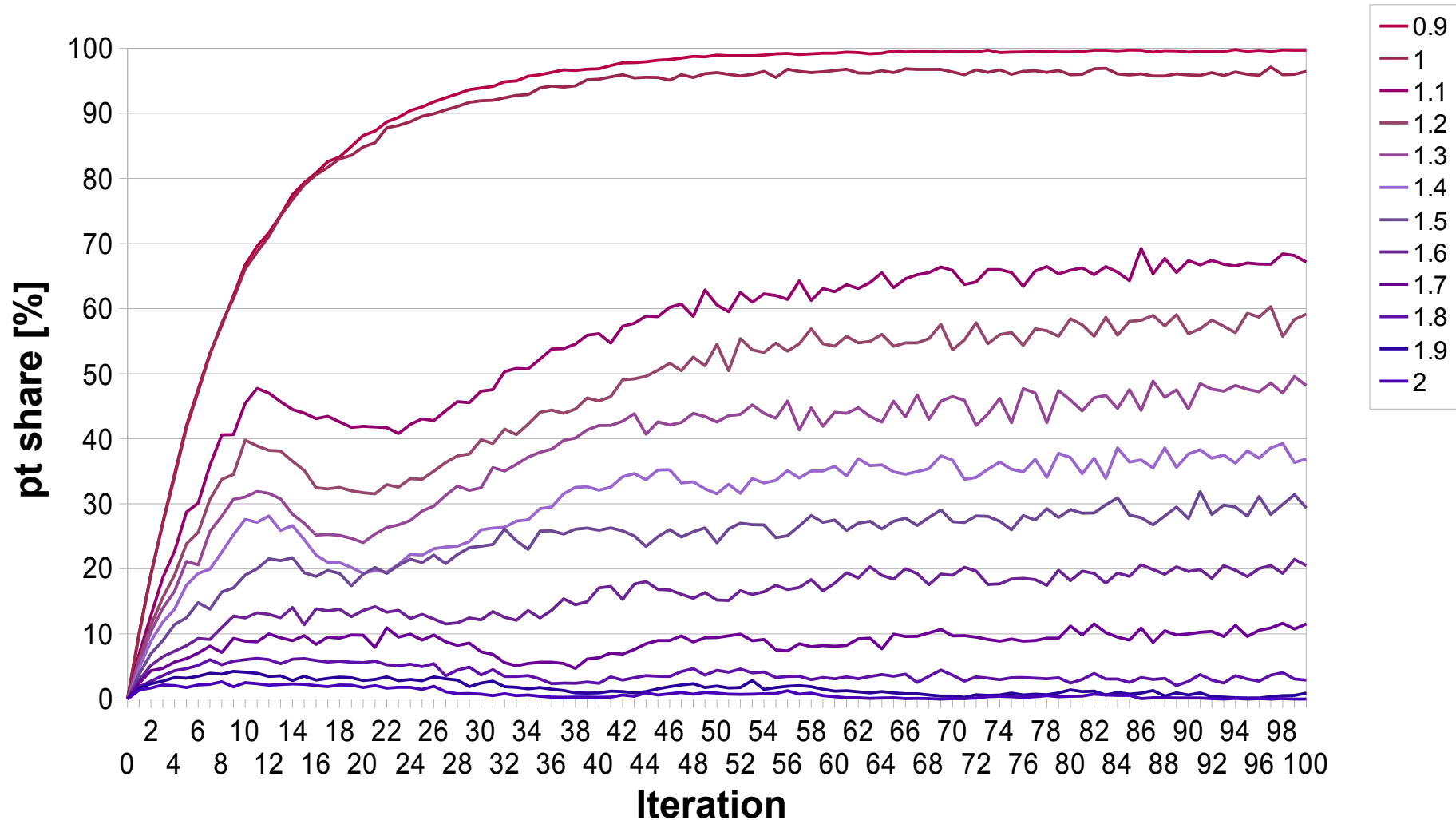


planomat and routes, ptFreespeedFactor = 1.05



# Preliminary results

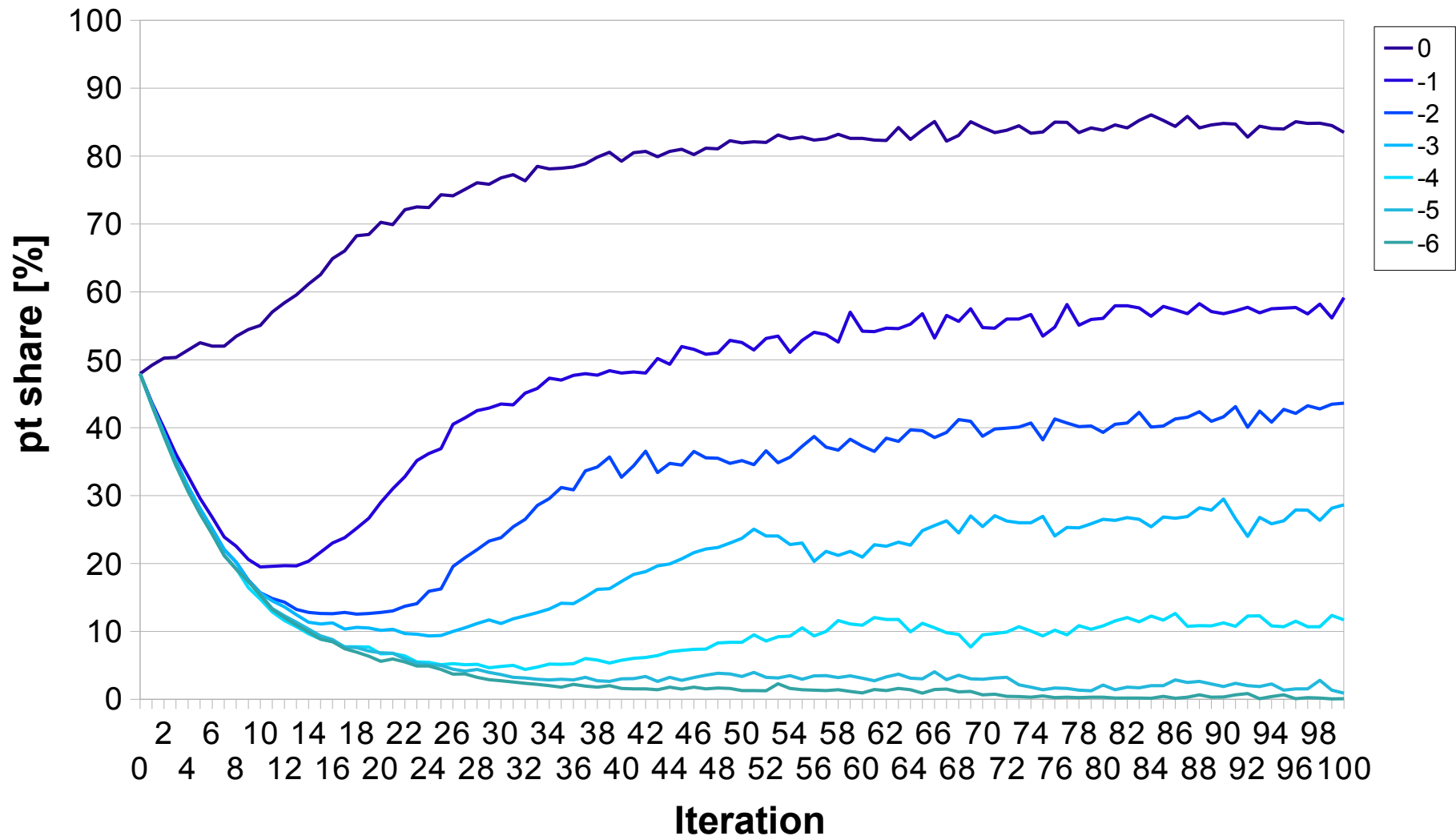
planomat, NO routes, varying ptFreespeedFactor



Initial demand: All car, all agents depart at 6:00 AM

# Preliminary results

planomat, NO routes, varying travelingPt



Initial demand: Random times and mode choice

# Thanks for your attention!

## Questions? Comments?

**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

Technische Universität Berlin



**MATSim**  
Multi-Agent Transport Simulation

# References

Downs, A. (2004) *Still Stuck in Traffic: Coping with Peak-Hour Traffic Congestion*, Brookings Institution Press, Washington, D.C.