	Instituut voor Mobiliteit (IMOB) Universiteit Hasselt Wetenschapspark 5 bus 6 3590 Diepenbeek
Subject	Notes on Collaboration - Carpooling Model
Authors	Luk Knapen
File	$dataSim/pre/WP5/design/carpoolingNotes
Date	`$Date: 2013-03-14 22:15:06 +0100 (Thu, 14 Mar 2013) $`
Id	`$Id: carpoolingNotes.tex 228 2013-03-14 21:15:06Z LukKnapen $`

Contents

Chapter 1

Intro

1.1 Context - Document purpose

1. This is an IMOB-internal report: it is aimed to support the design of an Agent Based Model for Carpooling. It describes concepts, data structures and algorithms that can be used to solve subProblems in the model.

2. Companion documents: [1], [2], [3],

1.2 Glossary

activityPattern	: sequence of activityTypes corresponding to a single day
activityType	: enumeration *home, work, dailyShopping, socialVisit, ...*
cell	: element of a partition, **syn.** *part*, *block*
demographSig	: demographic signature for an individual (including *homeLocation*)
distance matrix	: square matrix giving the minimal distance to travel between origin and destination locations (centroids of *TAZ*). This matrix in general is not symmetric
homeLocation	: location where individual lives
homeWorkDist	: distance between *homeLocation* and work location
impedance matrix	: square matrix giving the time to travel between origin and destination locations (centroids of *TAZ*). This matrix in general is not symmetric. Impedance matrices hold for specific periods: *OFFpeak*, *AMpeak*, *PMpeak*
mode	: enumeration *walk, bike, car, carPassenger, roadBoundPubTr, railBoundPubTr, ...*
part	: element of a partition, **syn.** *cell*, *block*
modeSwitch	: changing mode when concatenating *tripComponents* (in a single *trip*)
pool	:
popDens(p)	: population density for the smallest area containing point p
socioEconSig	: socio-economic signature of an individual (array of classified values): education level, income category, profession type, ...
TAZ	: Traffic Analysis Zone (in general coincides with a *homogenous area*)
timeBlock	: finite period of time
transferium	: location in the transportation network where people can change transportation mode.
trip	: movement between two locations both of which host an activity
tripComponent	: movement between two locations at least one of which hosts a *modeSwitch* or a modification of the participants set. A tripComponent is defined by a tuple `(org,dst,ts0,mode,ps)` where `org` is the origin location, `dst` is the destination location, `ts0` is the time at which travel starts, `mode` is the transportation mode used and `ps` designates the participants set.

Part I

Supporting Concepts - Helpers

Chapter 2

Limited Detour Network

2.1 Definition and Algorithm

1. start from 2 locations: A, B (PPA Possible Path Area ?)
2. determine shortest path (from both A and B) smaller than given value to each node
3. keep euclidian distances for each point as lower bound estimates: what to do when cost function is used ?
4. use Dijkstra algorithm
5. remove node N from the network if distance from A to B via N is larger than specified limit; this is done
   (a) using shortest distances, if they are known
   (b) a shortest distance and a euclidian distance estimate, if one of the distances is still unknown
6. for each node (in digraph) following data are kept:
   (a) shortest from to A
   (b) shortest distance to A
   (c) shortest from to B
   (d) shortest distance to B
   (e) euclidian distance to A
   (f) euclidian distance to B

2.2 Characteristics

1. for each node N the sum of the distances to both A and B is the length of the shortest path from A to B via N
2. for a node N, the shortest path N.SP() to/from A (B) can be found by looking for the neighbour N1 for which N1.SP() +/- dist(N,N1) == N.SP(). So it is easy to reconstruct the shortest via-paths

2.3 K-shortest Paths

1. sort the nodes using the shortest via-distance from A to B in increasing order. Reconstruct the via-paths for the first K nodes in the sorted list: this is a K-shortest path set.
2. there is no node Ne *not* belonging to the LDN that is on a path shorter than the *limit length* used to construct the LDN

2.4 Application to simple carpooling case

1. Consider two people p_1 and p_2 planning to move from *origin*s O_1, O_2 to *destination* D. The maximal distances accepted by p_1 and p_2 respectively are $\overline{C_1}$ and $\overline{C_2}$
2. determine $ldn_1 = LDN(\overline{C_1}, O_1, D)$, $ldn_2 = LDN(\overline{C_2}, O_2, D)$
3. the set of carpooling parkings is CPP
4. the best carpool parking to use is

$$p = \begin{cases} none & if(CPP \cap nLdns_1 \cap nLdn_2 = \emptyset \\ \arg\min\limits_{p \in CPP}(dist(O_1, p) + dist(O_2, p) + dist(p, D)) & otherwise \end{cases} \tag{2.1}$$

which can readily be determined from ldn_1 and ldn_2.

Part II

Agents and Interactions

Chapter 3

Collaboration on Activity Execution

3.1 Schedule - Agenda

3.1.1 Schedule structure

1. sequences, precedence relation: TODO:LK
2. actor (set) operation sequence (see note 19 in manuscript): operations graph
3. partial order relation: TODO:LK

3.1.2 Collaboration

1. Actor relation and actor set partitioning

3.1.2.1 Collaboration Graphs

1. influence relationship, influence graph (see note 19 in manuscript)
2. transition merging: shared activities: TODO:LK

3.1.2.2 Negotiation to share operation

1. See [4]

3.2 A priori feasibility

1. initial agenda
2. collaboration (co-action, cooperation) on activity or tripComponent

3.3 Base case: pre-specified shared activities

3.4 Extension 1: Actors decide about collaboration

3.5 Extension 2: Resources owned by actors

3.6 Extension 3: Negotiatable actor attributes

3.7 Extension 4: Collaboration on split actions

3.8 Trips

3.8.1 Trip - TripComponent splitting

3.8.2 Trip Construction - Trip Network - Trip Graph

3.8.3 Consistency requirements on Trip Graphs

Chapter 4

Carpooling route determination

4.1 Symbols used

Table 4.1: Symbols Used

Symbol	Meaning
B_k	k-th Bell number (see [7])
$c_{\mathcal{A}}$	cost to travel an hyperArc
D	Set of destination locations
DRN	Detailed Road Network
$dist(a, b)$	Distance from a to b
E	Set of hyperedges (hypergraph edges)
$fts(p_i)$	Set of feasible transferia for participant p_i.
$fps(t_i)$	Set of participants for whom t_i is a feasible transferium.
$\mathcal{G}, \mathcal{H}, \ldots$	Hypergraphs
L	Set of links in graph
L_R	Set of links in reduced graph
LDN	Limited Detour Network
M	Set of transportation modes
N	Set of nodes in graph
N_R	Set of nodes in reduced graph
$z_{p,o}$	Number of participants starting at origin $o \in O$
O	Set of origin locations (in transportation network)
P	Set of people
$\mathcal{P}$	Set of all partitions of of P
P_C	Set of participant candidates
RN	Reduced transportation Network
$\overline{T}$	Set of available transferia (see 1.2 for *transferium* definition)

4.2 Problem statement

4.2.1 Context - Definitions - Terminology

1. Let P be the set of all persons and $P_C \subseteq P$ be the subset of carpooling participants under consideration. P_C is assumed to be given a priori. The algorithm described here is not aimed at finding the optimal set of participants: that task is accomplished by the *agent based simulator* that will make use of the procedures described here.

2. Consider a set O of *origin* nodes, a set D of *destination* nodes and a set T of transferia (transfer nodes) in a connected network. $O \cap D = \emptyset$ is required but $(T \cap O \neq \emptyset) \vee (T \cap D \neq \emptyset)$ is allowed.

3. Consider a set of people $P \ni p_i$ and their associated trip $(o_i, d_i) \rightleftharpoons p_i$ where $o_i \in O \wedge d_i \in D$.

4. The number of people leaving origin O_i is given by $z_{p,o_i} = |\{p_j \in P | p_j \rightleftharpoons (o_j, d_j) \wedge o_j = o_i\}|$.

5. Paths followed by people can *join* or *fork* at *transferia*. Mode switches essentially occur at transferia.

4.2.2 Reduced network

1. The carpooling application makes use of a *reduced network* derived from the detailed road network $DRN = \langle N, L \rangle$ with N the set of nodes and $L \subseteq N \times N$ the set of links. The reduced network RN corresponds to a *directed loop free graph*. It contains nodes contained in O, D and T only: $RN = \langle N_R, L_R. \rangle$ where $N_R = O \cup D \cup T$ and $L_R \subseteq N_R \times N_R$.

2. The set of links is given by the complete graph on nodes $N = O \cup T \cup D$.

3. The application makes use of a predefined set of carpool parking lots. Hence, T is given in advance and the reduced network can be calculated in advance. For the problem of finding an optimal *hyperroute*, the set of agents P is given, hence O and D are given.

4. For the forward trip (from origin to destination) only $L_{O,T}$, $L_{T,D}$ and $L_{O,D}$ are used: the network in this case reduces to

$$L = L_O \cup L_D \cup L_T \cup L_{O,T} \cup L_{T,D} \cup L_{O,D} \tag{4.1}$$
$$L_O = O \times O \setminus \{(o_j, o_j) | o_j \in O\} \tag{4.2}$$
$$L_D = D \times D \setminus \{(d_j, d_j) | d_j \in D\} \tag{4.3}$$
$$L_T = T \times T \setminus \{(t_j, t_j) | t_j \in T\} \tag{4.4}$$
$$L_{O,T} = O \times (T \setminus O) \tag{4.5}$$
$$L_{T,D} = (T \setminus D) \times D \tag{4.6}$$
$$L_{O,D} = O \times D \tag{4.7}$$
$$L_{D,O} = D \times O \tag{4.8}$$

   The subgraphs $\langle O, L_O \rangle$, $\langle D, L_D \rangle$ and $\langle T, L_T \rangle$ are *complete* graphs.

5. The number of nodes in the reduced network for the forward problem, is given by equation 4.12; the number of edges is given by equation 4.13.

$$z_O = |O| \tag{4.9}$$
$$z_D = |D| \tag{4.10}$$
$$z_T = |T \setminus (O \cup D)| \tag{4.11}$$
$$z_{nodes} = z_O + z_D + z_T \tag{4.12}$$
$$z_{edges} = z_O \cdot (z_O - 1) + z_D \cdot (z_D - 1) + z_T \cdot (z_T - 1) + z_O \cdot z_T + z_T \cdot z_D + z_O \cdot z_D \tag{4.13}$$

6. $L_{D,T}$, $L_{T,O}$ and $L_{D,O}$ are used for the return trip (from destination to origin) only.

7. The major part of the links (L_T) is to be determined only once since it does not depend on any particular carpooling activity.

4.2.3 Cost associated with links - Transportation modes

1. Reduced network link travel costs are assumed to be time independent (at least for the first version of the software). They are calculated from cheapest path costs in the original network (lowest cost to travel between two locations). Since they are constant, they can be calculated *just-in-time* and be cached for reuse.

2. A set M of transportation modes is assumed to be given a priori. The cost to travel each link in the reduced network is calculated for each applicable mode $m \in M$. Note on applicability: many carpool parkings are not reachable by train.

4.3 Problem formulation using hyperpaths

4.3.1 Shared route hyperPath

1. See [5] for an introduction to *directed hyperGraphs*,

2. TripComponents joining at a location $t_0 \in T$ correspond to hyperArcs (directed hyperEdges) in the directed hyperGraph $\mathcal{H} = \langle N_R, E \rangle$ where $E = \langle E_T, E_H \rangle \wedge E_H \cap E_T = \emptyset \wedge E_H \subset 2^{N_R} \wedge E_T \subset 2^{N_R}$. Here E_T is the *tail* of the hyperArc and $E_H = \{t_0\}$ is the *head*. Since everyone can leave a location l_0 to join with some other people at location l_1 (at some specific moment in time) only once, the hyperArc is a *backward star*. In the same way, hyperArcs starting from a *fork* location are *forward stars*.

4.3.2 Constraints - Assumptions

1. In this application, we assume that for each carpooling trip, there is at least one network link on which every participant is in the car. Everyone is picked up and drives together over a non-zero distance before the first participant is dropped off. The hyperPath corresponding to the carpooling trip can be split into two parts at the node corresponding to any location where all participants have boarded and no-one has left: the first part consists of *B-arc*s only (and thus is a (special case of a) *B-path*), the second part consists of *F-arc*s only.

2. The capacity of the car ($CarCap$) used for the trip is sufficient: $\sum_{o \in O} z_{p,o} \leq CarCap$. This is easily guaranteed by the software using the algorithms described here.

4.4 Problem characteristics

4.4.1 Route structure

1. Consider two nodes n_A and n_B on a candidate hyperGraph connecting origins to destinations so that between n_A and n_B each participant is on board of the vehicle, and so that before reaching n_A and after leaving n_B not everyone is on board. n_A is the last *join* transferium and n_B is the first *fork* transferium on each possible hyperPath in the hyperGraph: see figure 4.1

2. The hyperGraph $\mathcal{H}_{O,n_A}$ linking all nodes in O to n_A is a hyperTree since each hyperEdge (hyperArc) $E \in \mathcal{H}_{O,n_A}$ is a *B-arc* (backward hyperArc). The hyperGraph $\mathcal{H}_{n_B,D}$ linking n_B to all nodes in D is a hyperTree for analogous (*F-arc*) reasons.

3. Each hyperArc a in $\mathcal{H}_{O,n_A}$ corresponds to the ride-share of a set of people from a set of locations $tail(a)$ to location $head(a)$ arriving in a specific car. At each transferium in $\mathcal{H}_{O,n_A}$, all but one cars are left and the ride continues using the remaining car. Note that there shall be at least one car-based hyperArc arriving at each transferium: from the available cars, one of the cars having sufficient capacity to travel from n_A to n_B is chosen to continue the ride.

4.4.2 Costs

1. The *link* cost $c_{\mathcal{A},h,m}$ to travel a hyperArc h using mode m, is the sum of the *link travel* costs involved:

$$c_{\mathcal{A},h,m} = \sum_{l \in head(h) \times tail(h)} c_{l,m} \tag{4.14}$$

   where $c_{l,m}$ is the cost to travel a single link using mode m. The cost to travel a (sub)hyperGraph is the sum of the costs to travel the hyperArcs constituting the (sub)hyperGraph.

2. The *transfer* cost $c_T(h_0, ps)$ associated with a hyperTree h_0 where people $p \in ps$ join, is the total cost for mode changes in the hyperArc. It is calculated recursively as follows. Let

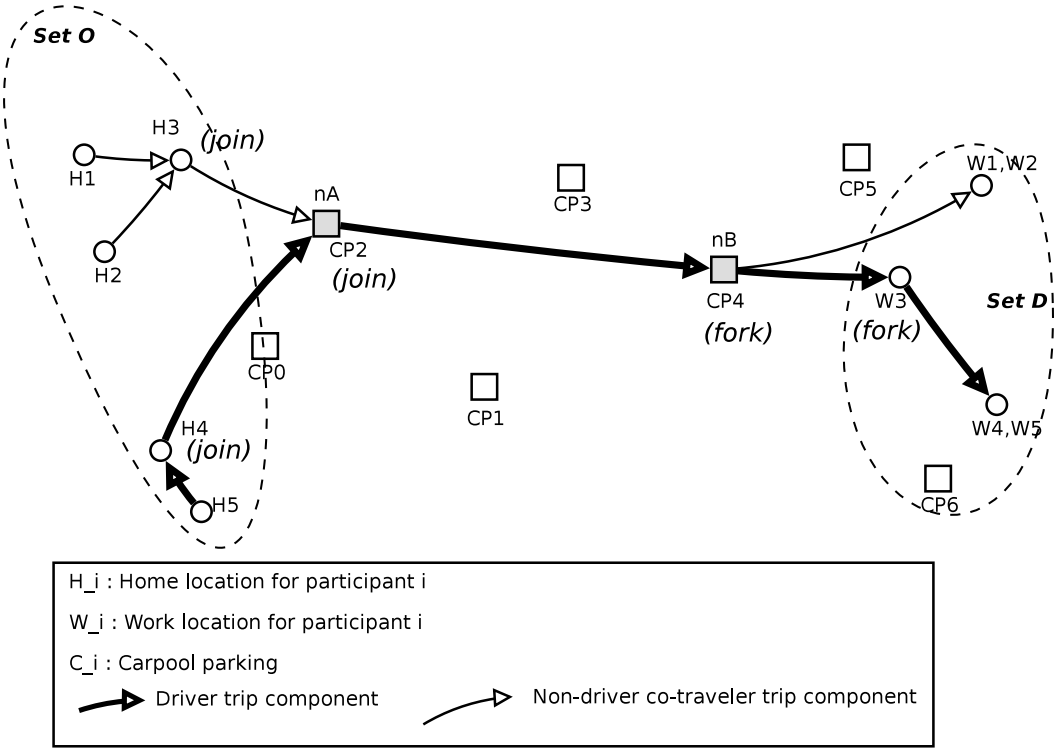

Figure 4.1: Trips driven by carpooling people numbered from 1 to 5. H_i are *home* locations. W_i are work locations. CP_i are carpool parkings. Person P_5 is the driver. P_1 and P_2 leave the participants set at N_B where they continue to their work location using a different mode (e.g. subway). P_3 is dropped at its destination. P_4 and P_5 work at the trip endpoint. N_A is the head of the backward hyperArc, N_B is the tail of the forward hyperArc.

$t \in T_0 \subseteq T$	be the set of transferia used
$ps_t \subseteq P$	be the set of participants joining at t
$c_A(t, ps)$	be the cost for participants ps to enter transferium t (A stands for *arrival*)
$c_D(t, ps)$	be the cost for participants ps to leave transferium t (D stands for *departure*)
$c_W(t, ps)$	be the cost for participants ps to wait at transferium t (W stands for *wait*)
VOT_p	be the value of time fort participant p
$\Delta\tau_{D,i}$	be the time required for departure from transferium i
$\Delta\tau_{W,0,i}$	be wait time at transferium t_0 for participants coming from location i

then TODO:LK SOMETHING WRONG HERE ?

$$c_T(h_0, ps_0) = c_A(t_0, \bigcup_{i \in tail(h_0)} ps_i) + \sum_{i \in tail(h_0)} c_T(t_i, ps_i) + c_W(t_i, ps_i) + c_D(t_i, ps_i) \tag{4.15}$$

$$= \sum_{i \in tail(h_0)} c_T(t_i, ps_i) + c_W(t_i, ps_i) + c_D(t_i, ps_i) + c_A(t_0, ps_i) \tag{4.16}$$

with

$$c_A(t_0, ps_i) = k_{A,0} \cdot \sum_{j \in ps_i} VOT_j \tag{4.17}$$

$$c_D(t_i, ps_i) = \Delta\tau_{D,i} \cdot \sum_{j \in ps_i} VOT_j \tag{4.18}$$

$$c_W(t_i, ps_i) = \Delta\tau_{W,0,i} \cdot \sum_{j \in ps_i} VOT_j \tag{4.19}$$

hence

$$c_T(h_0, ps_0) = \sum_{i \in tail(h_0)} \left(c_T(h_i, ps_i) + \sum_{j \in ps_i} VOT_j \cdot (\Delta\tau_{W,0,i} + \Delta\tau_{D,i} + k_{A,0}) \right) \tag{4.20}$$

$B_1 = 1$	$B_7 = 877$
$B_2 = 2$	$B_8 = 4140$
$B_3 = 5$	$B_9 = 21147$
$B_4 = 15$	$B_{10} = 115975$
$B_5 = 52$	$B_{11} = 678570$
$B_6 = 203$	$B_{12} = 4213597$

Table 4.2: Bell numbers (taken from http://oeis.org/A000110 (*The On-Line Encyclopedia of Integer Sequences*))

3. The cost to bring all participants $p \in ps_0$ from their respective origins to $n_A = head(h_0)$ is given by

$$c_{n_A} = c_T(h_0, ps_0) + \sum_{l \in links(h_0)} c_{\mathcal{A},h,m_l} \tag{4.21}$$

where m_l is the mode used to travel link l.

4. The cost to travel $\mathcal{H}_{O,n_A}$ ($\mathcal{H}_{n_B,D}$) is the sum of the hyperArc travel costs. Note that hyperArcs in $\mathcal{H}_{O,n_A}$ ($\mathcal{H}_{n_B,D}$) can have different *mode*s (subject to some constraints). See also section 4.8.3. The cost for the carpooling equals

$$cost(\mathcal{H}_{O,n_A}) + cost(\mathcal{H}_{n_A,n_B}) + cost(\mathcal{H}_{n_B,D}) \tag{4.22}$$

where $\mathcal{H}_{n_A,n_B}$ is the trivial hyperGraph (simple path) linking n_A to n_B.

5. Each feasible combination (n_A, n_B) will be investigated. For each such combination, every possible $\mathcal{H}_{O,n_A}$ and $\mathcal{H}_{n_B,D}$ need to be constucted and evaluated.

4.5 *Join* hypertree - Problem search space size

4.5.1 Building the *join* hyperTree

1. In order to get all participants in P joined at n_A, several transferia can be used. At each transferium, two or more subsets of P can be merged (refer to figure 4.1): this can be done in multiple stages so that in the end at n_A all $p \in P$ belong to the same set. A selected series of *merge* operations corresponds to a hyperTree.

2. *Part clustering*: at each transferium, two or more parts are merged to result in a new partition. The *part clustering* step generates a partition $\mathcal{P}_1$ from a partition $\mathcal{P}_0$ by merging a set of parts from $\mathcal{P}_0$: this done in every *join* node. *Part clustering* continues until the resulting partition consists of a *single part* only (the one that contains all participants). This one is connected directly to the root n_A. Note that

   (a) at transferium n_A the resulting partition consists of a single *part* only (containing all participants).

   (b) each origin is the source of exactly one *tripComponent* having its dedicated mode: for some modes a vehicle is required.

   (c) a *transferium* is to be selected in each *clustering* step.

3. This process is repeated for every possible hyperTree that connects the set O to n_A (hence all candidate participants $p \in P$). Every possible partition of set P is to be considered: the number of partitions for z_P for $P|$ is given by the Bell number $B_{n_O} = \sum_{k=0}^{n_O-1} \binom{n_O-1}{k} B_k$ (see [7]). The first Bell numbers are given in table 4.2. For additional info on Bell numbers, consult
   http://mathworld.wolfram.com/BellNumber.html
   http://en.wikipedia.org/wiki/Bell_number)
   http://oeis.org/A000110

4. The number of hyperTrees to investigate thus grows rapidly with the number of participants.

4.5.2 Structuring all partitions of a set

1. Consider two partitions p_0 and p_1 of a set S. If and only if each element in p_0 is a subset of an element in p_1, then p_0 is said to *refine* p_1. This defines the *refinement* relation R on the set of all partitions $\mathcal{P}(S)$ of a given set S. The *refinement* relation is a partial order relation over $\mathcal{P}(S)$. The *refinement* relation is a *complete lattice* (i.e. all elements share the same *infimum* $\bigvee S$ and the same *supremum* $\bigwedge S$).

2. Now consider the *transitive reduction* of the *refinement* relation. Since the refinement relation is a lattice, the transitive reduction is unique (the lattice corresponds to a finite directed acyclic graph).

3. The transitive reduction of the refinement is graphically represented by a *Hasse diagram*. Consider a partition of $\mathcal{P}(S)$ defined by the equivalence relation *containsSameNumberOfParts*.

$$p_0, p_1 \in \mathcal{P}(S) : (p_0, p_1) \in containsSameNumberOfParts \Leftrightarrow |p_0| = |p_1| \tag{4.23}$$

   Then consider the representation of the *transitive reduction* that emphasizes the *rank structure* as shown in figure 4.2. Each layer in the diagram contains partitions of S with identical cardinality. Let P_{csnop} denote a partition of $\mathcal{P}(S)$: it is induced by relation *containsSameNumberOfParts*.

$$\forall l \in P_{csnop}, \forall p_0, p_1 \in l : |p_0| = |p_1| \tag{4.24}$$

   The equivalence classes of P_{csnop} (Hasse diagram layers) are identified by l_i so that $i = |p| \wedge p \in l_i$. The index i denotes the size of the *part*s (*cell*s) in l_i.

4. Consider an ordered set L of Hasse diagram layers l_i, ordered by index i. The number of *layer*s equals the number of elements in the set S. Let $P_{i,j}$ denote the j-th part in *layer* l_i.

4.5.3 Mapping Hasse diagram paths to transferia sets

1. Apply the structuring idea from section 4.5.2 to the set of participants P. Moving from layer i to layer $i-1$ corresponds to joining two sets of participants into a single one.

2. Each *layer* in the Hasse diagram is associated with a transferium at which the *join* occurs. Let n_P be the number of participants (and thus the number of *layer*s).

$$\mathcal{T} : \mathbb{N}_1^{n_P} \Rightarrow T : \Theta(i) \mapsto t \tag{4.25}$$

   This mapping is subject to specific constraints (see sections 4.6 and 4.7).

3. This structure is interpreted/used as follows: consider a path from $\bigvee S$ to $\bigwedge S$ in the diagram. The link from P_{i+1}, j to P_i, k designates the join operation performed at transferium $\Theta(i)$ that merges the participants of exactly two *part*s. Refer to figure 4.3

4. A specific *join hyperTree* defines a partial order relation PO over the *join* events at the respective *transferia*.

   (a) *Join events* in different subtrees are mutually independent because one such *join event* combines two disjoint participant subsets for which it is not relevant by which sequence of join events they have been constructed.

   (b) Hence, each total order relation that embeds PO can be used and they are all equivalent: considering one of them is sufficient to investigate all different solutions (because of the mentioned independency).

   (c) If $\mathcal{H}_a$ is a subHyperTree of $\mathcal{H}_b$ then all join events for $\mathcal{H}_a$ precede the join event associated with the root of $\mathcal{H}_b$.

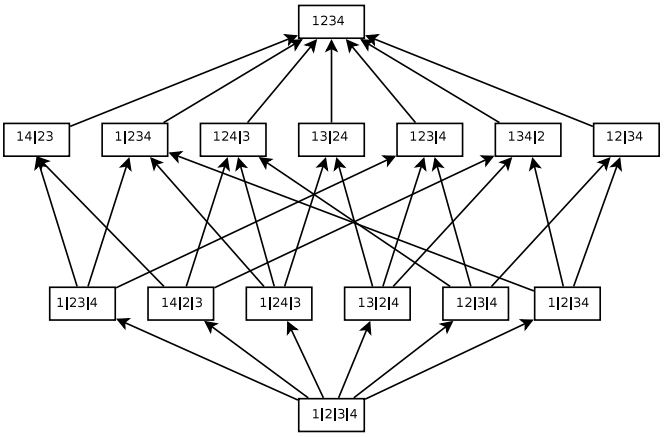

Figure 4.2: Hasse diagram for the transitive reduction of the refinement relation for a 4-element set partitioning

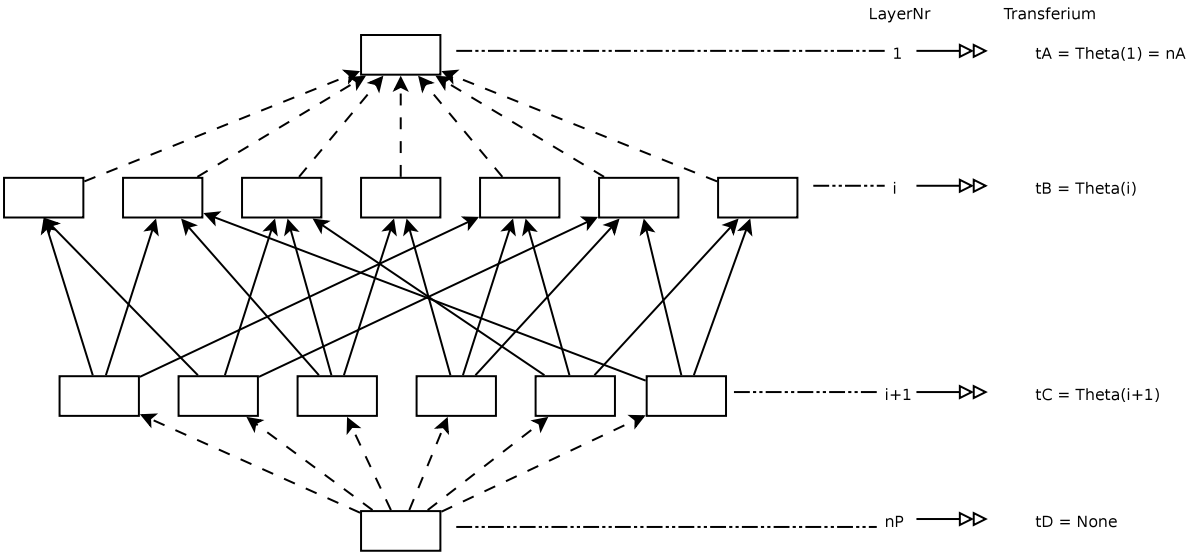

Figure 4.3: Mapping of Hasse diagram layers onto set of transferia. The interpretation of the thick line edge is: merge parts A and B at transferium $\Theta(i) = t$

(d) Let $ns_a = nodes(\mathcal{H}_a) \setminus root(\mathcal{H}_a))$ (and similar definition for ns_b). Then, if $ns_a \cap ns_b = \emptyset$, the subtrees are mutually independent. Let $e_{a,i}$ be a *join* event associated with a specific node i in $\mathcal{H}_a$ (and $e_{b,j}$ defined similarly) then

$$(ns_a \cap ns_b = \emptyset) \Rightarrow (\forall i, j \in \mathbb{N} : (e_{a,i}, e_{b,j}) \notin PO) \tag{4.26}$$

(e) We use the total order where all *join events* in a particular transferium are consecutive elements and thus are not interspersed with unrelated join events.

5. Every instance of the mapping $\mathcal{T}$ specified in 4.28 then can be determined as follows. Let Q denote a partition of the ordered set of layers $\mathcal{L}$ and let $q_j \in Q$ denote a *part* (*cell*). Every *layer* in a *part* q_j is mapped to the same transferium t: see figure 4.4

$$\overline{\mathcal{T}} : Q \Rightarrow T : \overline{\Theta}(q_j) \mapsto t \tag{4.27}$$

$$\mathcal{T} : \mathcal{L} \Rightarrow T : \Theta(l_i) \mapsto \overline{\Theta}(q_j \ni l_i) \tag{4.28}$$

4.5.4 Problem size

1. Sections 4.5.2 and 4.5.3 define the search space associated with the *coRouting* problem. Solving the problem by brute force consists of generating all possible paths from $\bigvee S$ to $\bigwedge S$

4.5.4.1 Number of partitions in a layer

1. For a set S of $N = |S|$ elements, each partition at layer l is constructed by $N - l$ join operations. Each such partition thus corresponds with a sum expression for N that contains $N - l$ plus $(+)$ operators

$$N = \sum_{i=1}^{i=l-1} n_i \tag{4.29}$$

   with $n_i \in \mathbb{N}$.

2. Let M_k be the number of ways to write N as a sum of k natural numbers with $1 < k < N$ and let $f_m(n_i)$ denote the frequency of n_i in the m-th sum: the number of *partitions* in the *layer* then is given by

$$N_l = \sum_{m=1}^{m=M_k} \frac{N!}{\prod_{i=l-1}^{i=1} f_m(n_i)! \cdot (n_i!)^{f_m(n_i)}} \tag{4.30}$$

   because neither the order of *part*s of equal size nor the order of elements within a *part* are relevant.

4.5.4.2 Number of paths through the Hasse diagram

The number of edges pointing from *layer* i to layer $i - 1$ equals $\binom{i}{2}$. Hence, the number of paths $z_p(N)$ to investigate for a set of size N is given by

$$z_p(N) = \prod_{i=1}^{i=N} \binom{i}{2} = \frac{(N-1)! \cdot N!}{2^{N-1}} \tag{4.31}$$

$$\frac{z_p(N)}{z_p(N-1)} = \frac{N \cdot (N-1)}{2} \tag{4.32}$$

4.5.4.3 Number of transferium assignments

1. The size of the ordered set $\mathcal{L}$ of Hasse diagram layer is given by $|\mathcal{L}| = n_P$ (the number of participants). The number of partitions of $\mathcal{L}$ is 2^{n_P-1} because partitions of an ordered set can be generated by *installing separators between elements*. There are $n_P - 1$ locations for separators. The number of partitions having exactly m parts is given by

$$z_{\mathcal{L},m} = \binom{n_P - 1}{m - 1} \tag{4.33}$$

2. Let z_{cpp} the number of carpool parkings. Then the number of different $\mathcal{T}$ functions as specified by 4.28 is given by

$$z_\Theta = \sum_{i=1}^{i=n_P} (z_{\mathcal{L},i} \cdot \binom{z_{cpp}}{i}) = \sum_{i=1}^{i=n_P} (\binom{n_P - 1}{i - 1} \cdot \binom{z_{cpp}}{i}) \tag{4.34}$$

where $\binom{k}{0} = 1$.

4.5.4.4 Search space size

1. The search space size is given the product of 4.31 and 4.34

$$z_p(n_p) \cdot z_\Theta = \frac{(n_p - 1)! \cdot n_p!}{2^{n_p-1}} \cdot \sum_{i=1}^{i=n_P} (\binom{n_P - 1}{i - 1} \cdot \binom{z_{cpp}}{i}) \tag{4.35}$$

2. This number grows rapidly with z_{cpp} and n_p. It is essential to find as many constraints as possible to limit the search space size.

4.5.4.5 Numerical example: number of partitions on each layer for $N = 4$

Layer	Split sum	nPartsPerSplit	nPartsPerSplit	nPartsPerLayer
1	4	$\frac{4!}{(1!\cdot(4!)^1)}$	1	1
2	$1+3$	$\frac{4!}{(1!\cdot(1!)^1)\cdot(1!\cdot(3!)^1)}$	4	7
	$2+2$	$\frac{4!}{(2!\cdot(2!)^2)^1)}$	3	
3	$1+1+2$	$\frac{4!}{(2!\cdot(1!)^2)\cdot(1!\cdot(2!)^1)}$	6	6
4	$1+1+1+1$	$\frac{4!}{(4!\cdot(1!)^4)}$	1	1

4.5.4.6 Numerical example: number of partitions on each layer for $N = 5$

Layer	Split sum	nPartsPerSplit	nPartsPerSplit	nPartsPerLayer
1	5	$\frac{5!}{(1!\cdot(5!)^1)}$	1	1
2	$1+4$	$\frac{5!}{(1!\cdot(1!)^1)\cdot(1!\cdot(4!)^1)}$	5	15
2	$2+3$	$\frac{5!}{(1!\cdot(2!)^1)\cdot(1!\cdot(3!)^1)}$	10	
2	$1+1+3$	$\frac{5!}{(2!\cdot(1!)^2)\cdot(1!\cdot(3!)^1)}$	10	25
	$1+2+2$	$\frac{5!}{(1!\cdot(1!)^1)\cdot(2!\cdot(2!)^2)}$	15	
4	$1+1+1+2$	$\frac{5!}{(3!\cdot(1!)^3)\cdot(1!\cdot(2!)^1)}$	10	10
5	$1+1+1+1+1$	$\frac{5!}{(5!\cdot(1!)^5)}$	1	1

4.5.4.7 Numerical example: number of paths to investigate

$N:$	$z_p(N):$	$\frac{z_p(N)}{z_p(N-1)}$
2 :	1 :	
3 :	3 :	3
4 :	18 :	6
5 :	180 :	10
6 :	2700 :	15
7 :	56700 :	21
8 :	1587600 :	28
9 :	57153600 :	36
10 :	2571912000 :	45

4.5.4.8 Numerical example: number of cases to investigate

With $n_p = 4$ and $z_{cpp} = 10$ the number of cases is given by

$$z = \frac{3! \cdot 4!}{2^3} \cdot (\binom{3}{0} \cdot \binom{10}{0} + \binom{3}{1} \cdot \binom{10}{1} + \binom{3}{2} \cdot \binom{10}{2} + \binom{3}{3} \cdot \binom{10}{3}) \tag{4.36}$$

$$= 18 \cdot (1 \cdot 1 + 3 \cdot 10 + 3 \cdot 45 + 1 \cdot 120) \tag{4.37}$$

$$= 5148 \tag{4.38}$$

Please note that this number applies to *join hyperTree* only. The *fork hyperTree* will result in a similar figure. The product of those numbers gives the size of the problem that is to be investigated for several combinations of n_A and n_B (see section 4.4.2/5).

4.6 Transferium feasibility

4.6.1 Transferia feasible for a participant

1. Each participant $p_i \in P$ is assumed to specify the maximal distance $\overline{td}(p_i)$ that can be spent to travel a specific trip $\langle o(p_i), d(p_i)\rangle$ where $o(p_i) \in O$ and $d(p_i) \in D$. This information is used to calculate a Limited Detour Network (see 2) $LDN(\overline{td}(p_i), o(p_i), d(p_i)) = \langle N_{LDN}, L_{LDN}\rangle$ where $L_{LDN} \subset N_{LDN} \times N_{LDN}$.

2. Each transferium that is not in N_{LDN} for participant p_i, is infeasible for p_i.

$$P_C \subseteq P \tag{4.39}$$

$$T_f = \{t \in T | t \in \bigcap_{p_i \in P_C} N^i_{ldn}\} \tag{4.40}$$

where N^i_{ldn} is the nodeSet of $LDN(\overline{td}(p_i), o(p_i), d(p_i))$.

4.6.2 Transferia set reduction

1. TODO:LK negeer elk T dat niet voor minimaal 2 participants bruikbaar is

2. The set of feasible transferia T_f can be reduced to the intersection of the sets of candidate specific feasible transferia. When assigning transferia to a *layer* l in the Hasse diagram for a path p through the diagram, the participants set is determined by *join* operation associated with the link used from level $l+1$ to level l.

3. Let $fts(p_i)$ denote the set of feasible transferia for participant p_i (feasible transferium set for participant).

4. Let $fps(t_i)$ denote the set of participants for whom t_i is a feasible transferium (feasible participant set for transferium).

4.6.3 Partial order over transferia

4.6.3.1 Relations

1. A *part* at a higher level *layer* in the Hasse diagram cannot contain more participants than a part on a lower level *layer* (by definition).

2. A transferium t_a is said to be more specific ($\prec$) than t_b if an only if

$$\begin{aligned} t_a \prec t_b &\Leftrightarrow fps(t_a) \subset fps(t_b) \\ t_a \preceq t_b &\Leftrightarrow fps(t_a) \subseteq fps(t_b) \end{aligned} \tag{4.41}$$

This defines a partial order over the set of transferia: see figure 4.5 for an example).

3. From section 4.5.2/1 it follows that

$$\forall i,j : p_{i,j_i,k_i} \in \mathcal{P}^{j_i}_i(P), p_{i+1,j_{i+1},k_{i+1}} \in \mathcal{P}^{j_{i+1}}_{i+1}(P) : (p_{i+1,j_{i+1},k_{i+1}} \cap p_{i,j_i,k_i} = \emptyset) \vee (p_{i+1,j_{i+1},k_{i+1}} \subseteq p_{i,j,k}) \tag{4.42}$$

where $\mathcal{P}^j_i(P)$ denotes the j-th partition in the i-th layer and $p_{i,j,k}$ is the k-th *part* in that partition.

4. From equations 4.41 and 4.42 it follows that: if an optimal solution consists of the assignment of a set of transferia, not necessarily all of which need to be feasible for all participants, then the more specific transferia are assigned to higher level *layer*s. The order of assignment of unrelated transferia is irrelevant. Hence we use an arbitrarily chosen total order that embeds the partial order specified by equations 4.41 to construct TOTS (Totally Ordered Transferium Set) (see figure 4.7).

4.6.3.2 Example

1. Example: Table 4.4 corresponds to figure 4.5 and shows transferia not unsuitable for participants. It is not guaranteed that the transferia are suitable because they follow from the LDN: this guarantees suitability if a transferium is used as the only one but does not convey any information about combined use of transferia.

2. Table below shows participant subsets for transferia.

	Transferia	
Participant	Homes	CarPoolParks
P1	H1, H2	CP0, CP1, CP4, CP5, CP6
P2	H2	CP1, CP4, CP6
P3	H3	CP2, CP4, CP5, CP6, CP7
P4	H4, H5	CP3, CP4, CP5, CP6, CP7
P5	H5	CP3, CP7

Table 4.3: Mapping of participants to transferia that *can* be feasible (are not infeasible). See also table 4.4, fig. 4.4, fig. 4.5

Transferium	Participants
H1	P1
H2	P1, P2
H3	P3
H4	P4
H5	P4, P5
CP0	P1
CP1	P1, P2
CP2	P3
CP3	P4, P5
CP4	P1, P2, P3, P4
CP5	P1, P3, P4
CP6	P1, P2, P3, P4
CP7	P3, P4, P5

Table 4.4: Mapping of transferia to sets of participants for whom use of the transferia *can* be feasible (is not infeasible). See also table 4.3, fig. 4.4, fig. 4.5

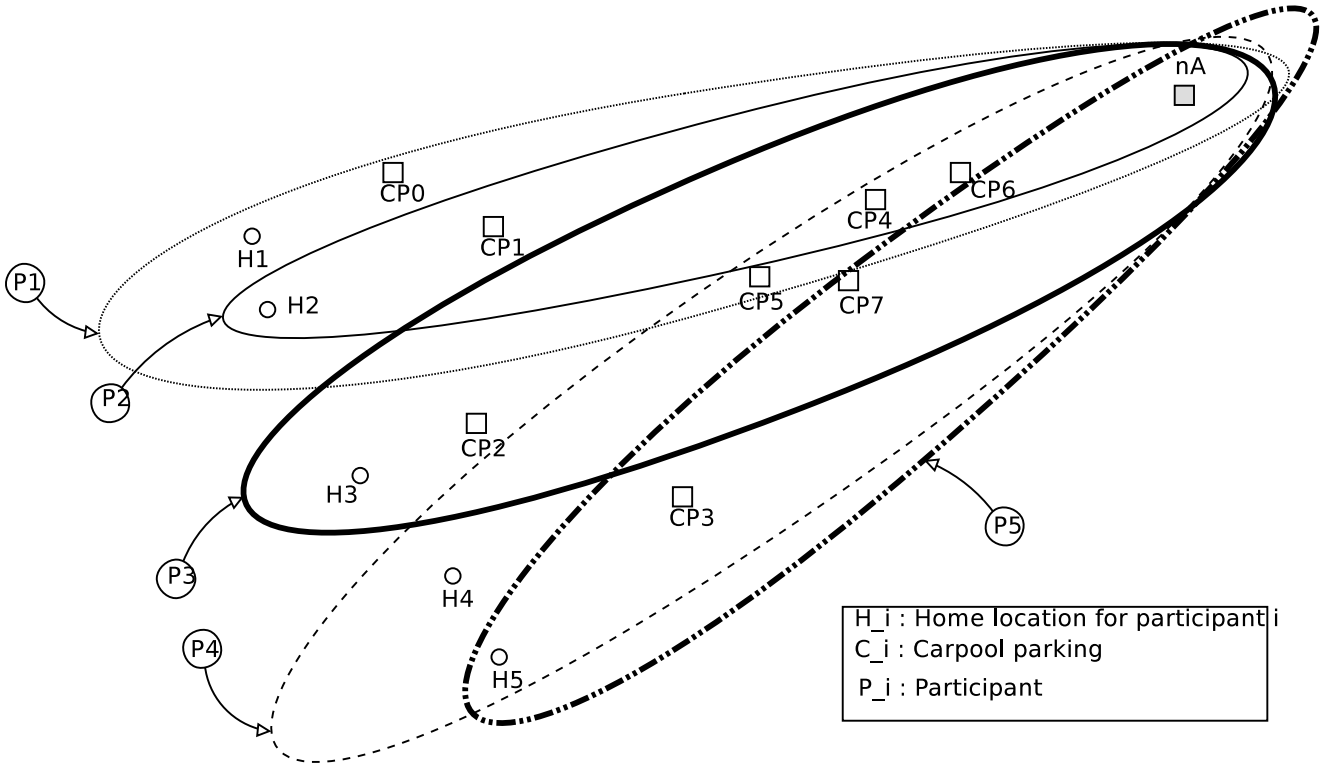

Figure 4.4: Participant specific *Limited Detour Network* perimeter encloses its home location (H_i), the target (n_A) and zero or more transferia (home locations H_j and carPoolParks (CP_k). The *containment* relation defined over the participant sets associated with the carPoolParks, defines a partial order over the carPoolParks. See also table 4.3, table 4.4, fig. 4.5

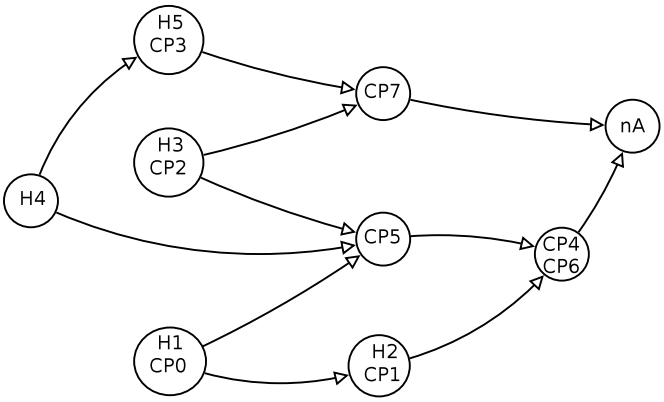

Figure 4.5: Transitive reduction of the partial order on the set of transferia induced by the containment relation between the corresponding participants sets $(ps(H4) \subset ps(CP3) \subset ps(CP7) \subset (ps(n_A) \wedge ps(H4) \subset ps(CP5) \subset ps(CP6) \subset ps(n_A) \wedge ps(CP2) \subset ps(CP7) \wedge ps(CP2) \subset ps(CP5) \wedge ps(CP0) \subset ps(CP5) \wedge ps(CP0) \subset ps(CP1) \subset (ps(CP6))$. See also table 4.3, table 4.4, fig. 4.4

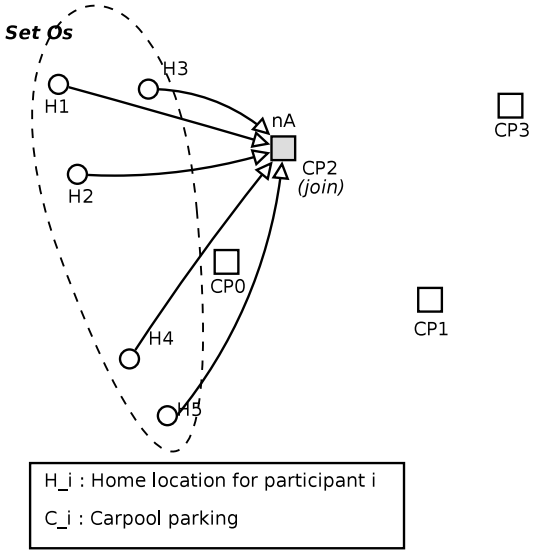

Figure 4.6: Removing a transferium from an optimal solution cannot generate a less expensive solution (Bellman principle of optimality). Compare this figure with figure 4.1: transferium at H_3 and H_4 have been removed from the *join* subTree.

4.7 Observations usable to build a computationally feasible solution

Notes in this section concern the case where a specific path through the Hasse diagram is being evaluated.

4.7.1 Transferium evaluation

1. Consider a node $n_r \in N$ (to be used as the subtree root) and a subset $O_s \subset O$ (see figure 4.6. The cost for the trivial tree consisting of one hyperArc a with $tail(a) = O_s$ and $head(a) = n_r$ is easy to calculate since the cost for each link $(o_{s,i}, n_r)$ with $o_{s,i} \in O_s$ is known (see 4.2.3/1). This sum reflects the case without carpooling from O to n_r and is used as the initial upper bound for the minimal cost associated with the subtree to be constructed.

2. Let $\check{c}(n_r, O_s)$ be the lowest hyperTree cost found at the moment in the search process when one needs to select a transferium to construct a *join*ing hyperArc. Assume $t \in \overline{T} \cup O_s$ is a candidate. The cost for a hyperTree cannot be less than the cost of one of its paths connecting a leaf to the root. Hence, if t would be selected then following holds for the cost for the new hyperTree $\gamma(n_r, O_s)$

$$\gamma(n_r, O_s) \geq \arg \max_{o \in O_s \setminus t} d(o, t, n_r) \tag{4.43}$$

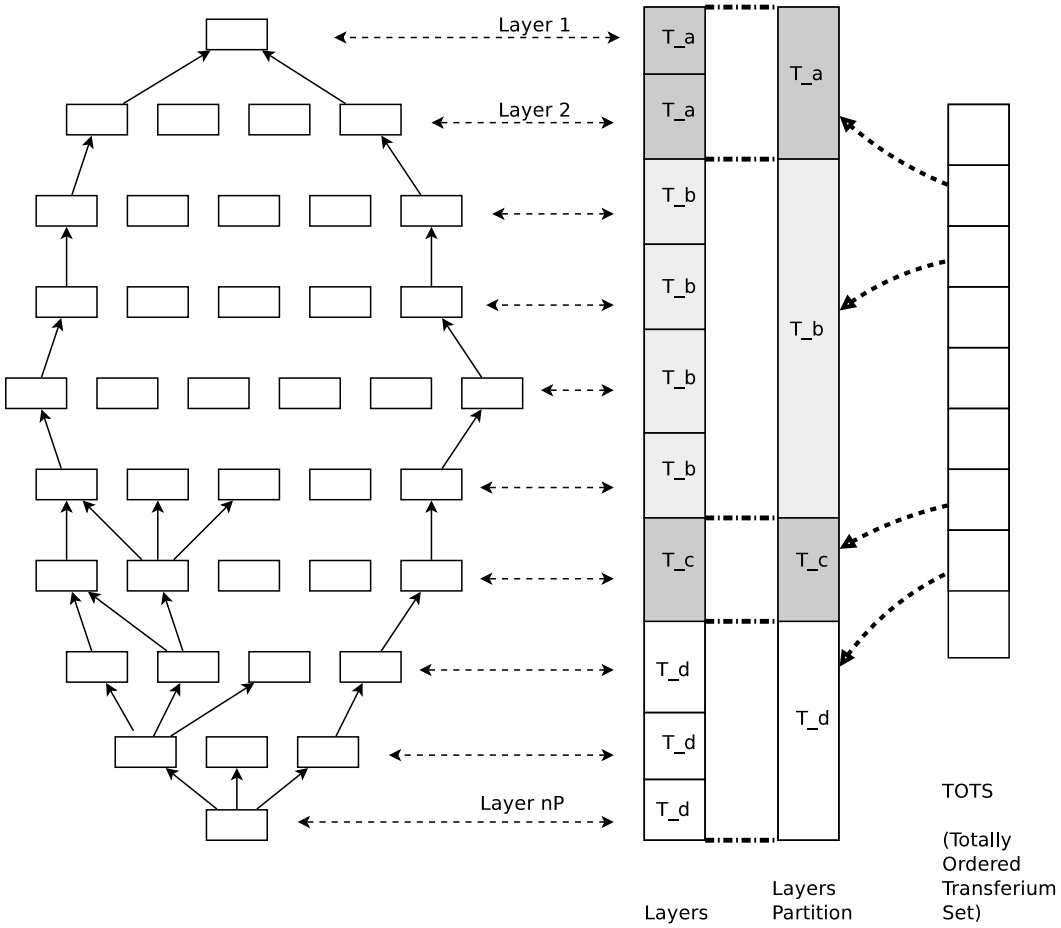

Figure 4.7: The graph on the left side represents the Hasse diagram (most edges not drawn). The *layer* number gives the number of *part*s in each participants set partition. The leftmost of the vertical bars shows the ordered set of *layer-transferium* assignments. The central vertical bar represents a partition of the ordered set: different transferia have been assigned to each *part* (cell). The rightmost vertical bar represents the ordered set of transferia (see section 11.3.2.2).

where $d(o, t, n_r)$ is the cost to move from o via t to n_r. The $d(o, t, n_r)$ values are readily available as simple two-term-sums (see 4.2.3/1). Infeasible transferia thus can easily get rejected when

$$\breve{c}(n_r, O_s) \leq \arg \max_{o \in O_s \setminus t} d(o, t, n_r) \tag{4.44}$$

4.7.2 Transferia assignment

1. Suppose a path from $\bigvee \mathcal{P}(P)$ to $\bigwedge \mathcal{P}(P)$ through the Hasse diagram is being evaluated. A transferium is to be assigned to each layer using the partition described in 4.5.3 and 4.5.4.3.
2. TODO:LK noem linkse kolom V (assignment vector); noem rechtste kolom W; W is een partitie van V; v0, v1 in zelfde wj ==¿ zelfde waarde; i¡¿j ==¿ wi ¡¿ wj
3. TODO:LK noem partiele orde over T TPO
4. TODO:LK optimale solution is (1) path in Hasse en (2) selectie van partitie van V en (3) assignment aan parts/cells in W
5. TODO:LK als t0 en t1 (met t0 ¡ t1 : more specific: zie 'Relations') in optim solution, dan zit t0 op hoger level dan t1 in W (bewijs); dit impliceert niets over het voorkomen van either one
6. TODO:LK als t0 en t1 unrelated in TPO, dan maakt orde in W niets uit (bewijs: zie hoger)
7. TODO:LK kies dus een totale orde waarin TPO ingebed is; voorgangers in deze orde komen op hoger niveau in W; hierdoor vermijdt men verifieren van permutaties in W die equivalent zijn

8. TODO:LK elke mogelijke partitie W moet worden onderzocht; die partities zijn ordered sets; de orde over de parts is bepaald door de orde van de onderliggende set van layers zodat $\forall l_0 \in p_0, \forall l_1 \in p_1, p_0 \neq p_1 : (p_0 \prec p_1) \Leftrightarrow (l_0 < l_1)$

   (a) TODO:LK aan elk part wordt een T toegewezen zodat meer specifieke op hoger level zitten: ord(t0) ¿= ord(t1) ==¿ ord(part(t0)) ¿= ord(part(t1))

   (b) TODO:LK gebruik element v[i] om edges van Hasse[i+1] ==¿ Hasse[i] te labelen met set van bruikbare T; het gebruikte T moet volstaan voor de target participants set van de edge (elke edge voegt juist twee parts samen); het target part moet gelijk of subset zijn van participants set van gebruikte T; edge wordt dus mogelijks onbruikbaar; wordt niet de hele target node onbruikbaar ? ja want die bevat minstens 1 part dat in onderzochte T nooit kan samenkomen; alle ingaande en uitgaande edges van target mogen disabled worden; TODO:LK size van part waaraan toegewezen kan niet groter zijn dan participants set fps(T) die T gebruiken

   (c) ingeval samenvoeging van parts α en β faalt an zijn alle samenvoegingen γ, β met $\alpha \subseteq \gamma$ onmogelijk; vraag is of deze check zin heeft (omwille van resources vereist voor berekening van subset)

4.8 Solution

4.8.1 Selecting root nodes for the *join* and *fork* parts of the hyperpath

1. The root nodes for *join* and *fork* parts respectively are n_a and n_b.

2. Each agent passes in both n_a and n_b. Hence both shall belong to the *LDN* for each participant (necessary condition).

3. For each agent, the distance from n_a to the destination shall not be smaller than the distance from n_b to the destination.

$$\forall p \in P_C : dist(n_a, D(p)) \geq dist(n_b, D(p)) \tag{4.45}$$

   This condition can be used to reject candidates when n_a and n_b are given or to reject (n_a, n_b) tuples when a candidate set is given.

4.8.2 Cars used

4.8.3 Multimodality and Tours

1. **Link costs**: in previous sections, each link has an associated cost for each mode. For *car* mode, the unit distance cost can be car specific. A *mode* that is unavailable on a link is assigned an infinitely large cost.

2. **Mode constraints**:

   (a) in the backward star, each agent can freely select a mode in its origin location (up to the first transferium). Everyone leaves each transferium using *car* mode (because transferia to non-*car* modes are not considered).

   (b) in the forward star, each agent can freely select a mode to move from the last transferium on its trip to the destination but not for other transferia (because transferia to non-*car* modes are not considered).

   (c) transferia $t \in T$ are either home locations or carPool parkings. At least one agent (the carPool trip driver) needs to have its car available at each location where trips join or fork.

   (d) the problem is essentially multimodal (because at least *car* and *carPassenger* modes are used. The solution is subject to the *real life mode switch constraints* that shall hold for each participant (bring own car back home *iff* it was used to leave at home).

```
<tour>   := <cTour> | <ncTour>
<cTour>  := <car> <tour> <car>       # car tour
<ncTour> := <nCar> <ncTour> <nCar>   # non-car tour
```

4.8.4 Algorithm

1. TODO:LK genereer partities van z_P set
2. TODO:LK maak Hasse diagramp voor participants
3. TODO:LK leg lijst van partities aan voor toewijzing van T aan layers: LayerParts
4. TODO:LK bepaal voor elke participant LDN dn set van T (relatie P X T)
5. TODO:LK bepaal voor elke T de set van participants (relatie P X T)
6. TODO:LK reduceer T tot T feasible voor minstens 2 participants
7. TODO:LK stel TPO op
8. TODO:LK find embedding order TTO
9. TODO:LK forall LP in LayerParts voor elke mogelijke subset SS0 uit TTO met size cardinaliteit van W wijs SS0 toe aan W; bepaal V label edges en nodes in Hasse (zie Transferia assignment): see section 4.7.2 zoek optimaal pad in Hasse; bij passeren van edge, verhoog aantal transferia bij alle participants in target part (nieuw samengesteld part) ingeval nieuw T gebruikt (ttz layers I+1 en i niet tot zelfe part behoren); stop zoektocht als voor minstens 1 participant de T limiet is overschreden; bepaal voor elke participant de kost (duur,afstand) en stop ingeval limiet is overschreden voor minstens 1 participant; omdat vanuit elk T de minimale kost tot n_A gekend is, is dit snel berekenbaar

4.8.5 Implementation - Notes

4.8.5.1 General Notes

1. TODO:LK bepaal lijst van partities van W (LayerParts): het genereren van die partities is identiek aan het genereren van partities over de participants set omdat beide zelfde cardinaliteit hebben.
2. TODO:LK algorithm levert kortste pad (afstand, tijd) maar houdt geen rekening met tijdstip.
3. TODO:LK aantal transferia per persoon kan beperkt worden (persoonlijke voorkeur). Kan tijdens evaluatie van Hasse pad triviaal worden bewaard en geverifieerd. Schakelt vrij vroeg infeasible paths uit.
4. voor efficiente implementatie van 4.7.28c moet men bitsets gebruiken

4.8.5.2 Transitive closure of Hasse diagram layers for a transferium assignment *part* (cell)

1. Refer to figure 4.7.
2. Let $R(\mathcal{P}(P))$ be the *refinement* relation over the set $\mathcal{P}(P)$ (the set of all partitions of the participants set).
3. Let Y be the set of equivalence classes induced by relation *containsSameNumberOfParts* (the set of *layer*s).
4. Consider a *part* (cell) Λ of the partition of Y currently being investigated.

5. Consider the set P_Λ of all participant partitions belonging to the layers λ_i in the Hasse diagram that belong to a specific *layer part* Λ.

$$P_\Lambda = \{p \in P | \exists \lambda_i \in \Lambda \wedge p \in \lambda_i\} \tag{4.46}$$

   They share transferium $\Lambda(Q)$.

6. Then consider the graph $\langle P(\Lambda), E \rangle$ where $E \subset P(\Lambda) \times P(\Lambda)$ and its transitive closure (stated informally ...). Then remove all nodes having at least one incoming and at least one outgoing edge. If Λ consists of layers l_i to l_{i+k} then all parts belonging to layers $l_{i+1} \ldots l_{i+k-1}$ have been removed. The subgraph has been reduced to layers l_i and l_{i+k} and their interconnections (derived from the transitive closure of the original subgraph). The subgraph has been replaced by a not necessarily injective but always surjective mapping from l_{i+k} onto l_i.

7. Paths through the Hasse diagram now are shorter because of this shortcut. This can speed up calculations since multiple paths from $p^m_{i+k} \in l_{i+k}$ to $p^n_i \in l_i$ now have been replaced by a single one. On the other hand, the equivalent of *edge labeling* described in section 4.7.2/8b now becomes more complicated.

8. It is probably not wise to implement this method unless at least 4 or 5 layers can be bridged.

4.8.6 Implementation - Open questions

1. kan men voor een participant een a priori een stel aanvaardbare *transferium sequence*s bepalen op basis van de afstanden ? Men moet dan ofwel bewijzen dat het optimum bestaat uit dergelijke sequences ofwel aannemen dat bepaalde oplossingen nooit gevonden zullen worden

2. TODO:LK Convergentie

   (a) TODO:LK op elk ogenblik reeds bereikte minimum kost bijhouden;

   (b) TODO:LK bij bepaling van embedding TTO van TPO: welke T moeten vooraan komen ?

      i. TODO:LK meeste steps verwijderd van n_A ?

      ii. TODO:LK grootste kost tot n_A (afstand, tijd) ?

      iii. TODO:LK grootste prs.km tot n_A ?

      iv. TODO:LK met minste kandidaten ?

      v. TODO:LK met meeste kandidaten ?

3. TODO:LK stopCriterium: na gegeven aantal evaluaties ? na minimale winst per evaluatie ? probleem is dat deze uitgaan van idee van continuiteit ...

4.8.7 Forward star and backward star roots determination

4.8.8 Return path

1. The problem of travelling from origin to destination is the *basic* problem. The problem of travelling back from destination to origin is the *return* problem.

2. For a given carPooling group (*pool*), the *reverse* of the origin-destination hyperpath is a solution worth evaluation for the return problem, at least when assuming that the back and forth cost to travel a link (pair of nodes) is time independent and identical for both directions.

3. The *return problem* solution is characterized as follows:

   (a) most participants do not have a car available at the start location (*basic* problem destination)

   (b) all participants who left their homes by car, except the driver, have the transferium where they left their car, as a target location.

4.9 Integration in Agent Based Modeling

4.9.1 Time

1. The method described in this chapter does not take *absolute time* into account. It simply delivers a good solution with respect to cost (distance, travel delay). The resulting solution, although feasible for each participant with respect to the stated budget limits, can be evaluated as infeasible because of travel time shift induced for some participants.

2. After a solution to the co-routing problem is found, the required starting times for each participant involved are calculated so that each of them reaches her/his destination at the stated deadline.

3. If all *disUtility* functions are *downward convex*, in theory the first derivative can be calculated. In practice this can turn out to be cumbersome if piecewise defined functions are used. It s probably a better solution to provide a method to calculate the total *disUtility of a co-route* and evaluate that one for discrete values of time (with a resolution of e.g. one minute).

Chapter 5

Cost Functions and VOT (Value of Time)

Chapter 6

Reputation

Chapter 7

Negotiation

Part III

Relations and Networks for CarPooling

Chapter 8

Introduction

8.1 Symbols Used

$\mathcal{A}$	: the set of all *agreements* (for definition see 8.2.2)
$\mathcal{I}$	: the set of all individuals
$\mathcal{P}$	: the set of all pools (for definition see 8.2.3)
$range(TOD)$	: $24 * 60 * 60$
$range(TOW)$	: $7 * range(TOD)$
$\mathcal{T}$	: The set of all *periodicTripEx* (for definition see 8.2.1)
TOD	: Time of day ; if expressed in seconds, cardinal in $[0, range(TOD) - 1]$
TOW	: Time of week ; if expressed in seconds, cardinal in $[0, range(TOW) - 1]$
$t_{\cdot,early}, t_{\cdot,late}$	: Earliest resp. latest time
$t_{d,\cdot}, t_{a,\cdot}$	: Departure resp. arrival time
$\mathcal{Z}$	: Set of *TAZ* (Traffic Analysis Zone)

8.2 Definitions

Refer to fig. 8.1

Definition 8.2.1 (periodicTripEx)**.** *A periodicTripEx is a tuple* $(i, O, D, w, t_{d,early}, t_{d,late}, t_{a,early}, t_{a,late})$ *where* $i \in \mathcal{I}$*, O and D denote the origin and destination locations respectively,* $t_{\cdot,\cdot} \in TOW$ *and w denotes a start-of-week moment in time so that the first trip execution for the given periodicTripEx starts in* $w, w + 1$*.*

Notes:

1. A *periodicTripEx* denotes the weekly execution of a trip with given characteristics by a specific individual. Individual i is called the owner of the *periodicTripEx*.
2. Examples:
   (a) $t_{d,late}$ is: *wednesday at 08:20h*
   (b) w = 2012-jun-04 00:00:00
   (c) Refer to fig. 8.2 to see intervals overlap.

Definition 8.2.2 (agreement)**.** *An* agreement *specifies operational details about the collaborative execution of all elements in the list of* periodicTripEx *to which the* agreement *applies. An agreement specifies the moment in time starting from which it holds.*

Notes:

1. An *agreement* has no termination time
2. A *periodicTripEx* is referred to by (belongs to) at most one *agreement*

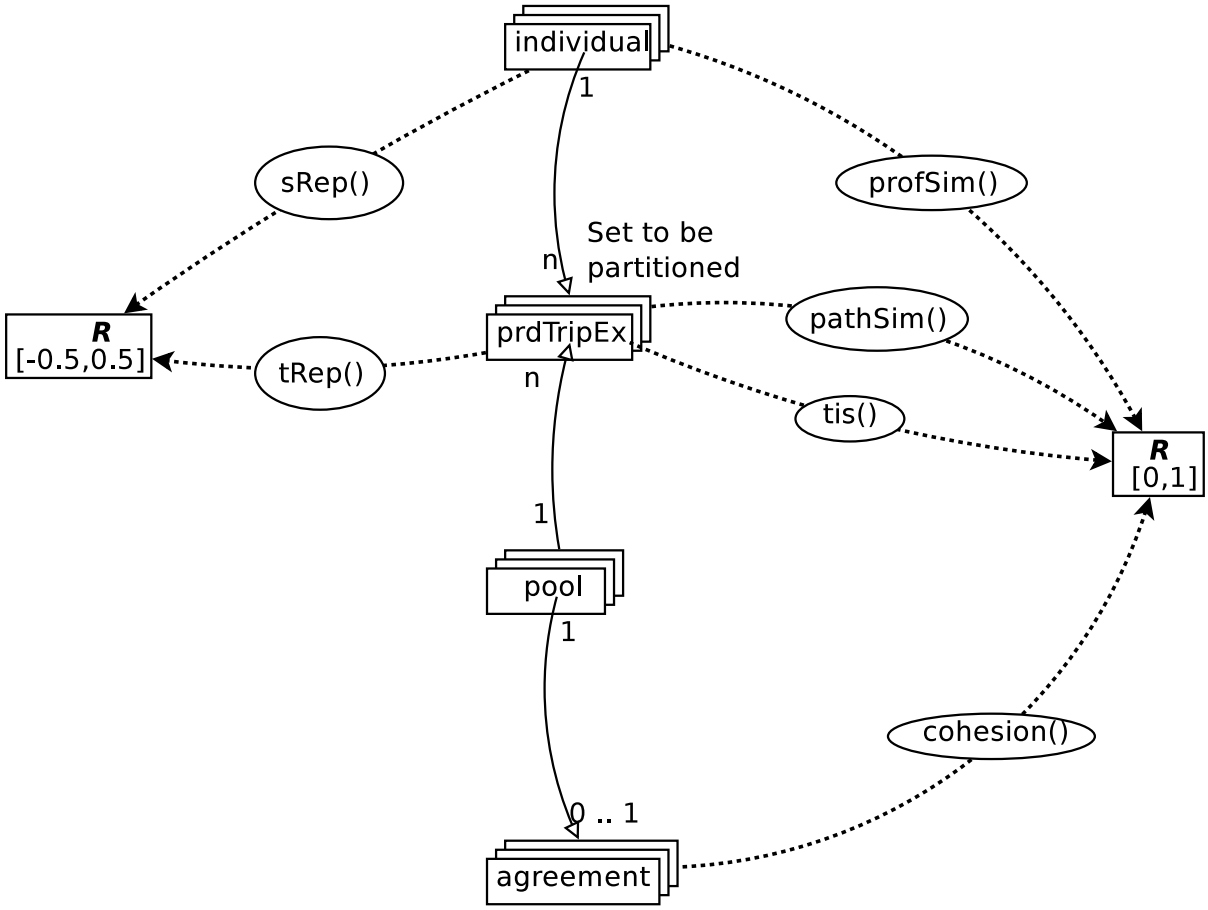

Figure 8.1: Overview of functions defined on the sets (*individual*, *prdTripEx* and *agreement*) used in the model and functions those sets are involved in. Continuous lines represent references, dashed lines represent functions.

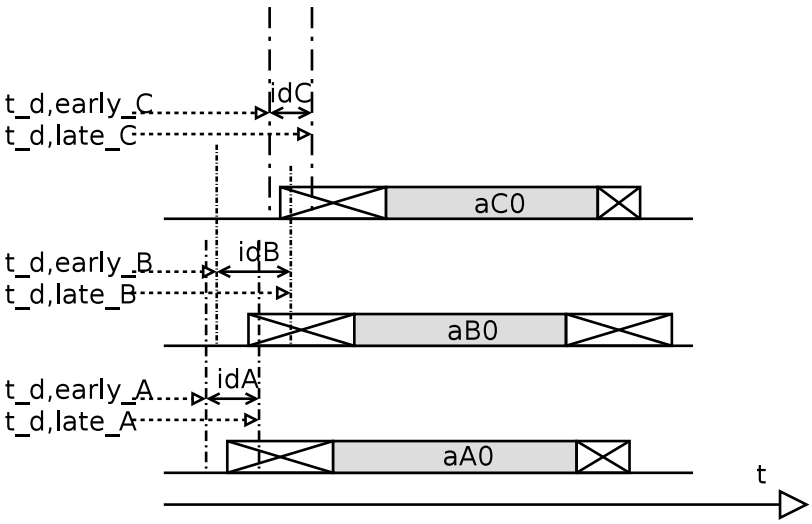

Figure 8.2: Activities $(a_{A,0}, a_{B,0}, a_{C,0})$ for individuals A, B and C and the associated trips. The valid departure intervals $i_{d,A}, i_{d,B}, i_{d,C}$ are shown. Note that B can choose to co-travel with A or C but A and C cannot co-travel.

3. *Agreement* details cover: timing, routing and driving. Since the route agreed upon can contain subroute *joins*, more drivers can be involved. For this reason the *agreement* does not explicitly mention a single driver

Definition 8.2.3 (pool)**.** *A* pool *is a tuple* $(\{a\}, T)$ *where* $\{a\}$ *denotes a set of* agreements *negotiated by the cooperating partners and* T *is a nonempty set of* periodicTripEx *so that* $(\{a\} = \emptyset \wedge \|T\| = 1) \vee (\|\{a\}\| = 1 \wedge \|T\} > 1)$

Notes:

1. The condition states that there is either a single individual without any *agreement* or multiple cooperating individuals sharing a single *agreement*.

2. Each individual occurs in at most one *periodicTripEx* in a specific *pool*:

$$\forall t_0, t_1 \in T : (t_0 \neq t_1) \Rightarrow (t_0.i() \neq t_1.i()) \tag{8.1}$$

   where $t.i()$ denotes the individual owning the *periodicTripEx*.

Definition 8.2.4 (preparedDriver)**.** *A prepared driver is an individual who stated to be able to be the driver in a* pool.

Notes:

1. Note that being a *preparedDriver* does not mean that the individual can be a driver in *any* pool (s)he can be member of. Being able to be the actual driver also depends on *path similarity* (see definition 8.2.7 and section 8.3.2).

2. Being a *preparedDriver* is a global concept i.e. it is not restricted to any particular set of combinations of *periodicTripEx*s.

Definition 8.2.5 (profile similarity)**.** *Profile similarity is a value in* $[0, 1]$ *assigned to a pair of individuals that indicates to what extent the individuals are compatible for carPooling.*

Definition 8.2.6 (pooled trip execution)**.** *A* pooled trip execution *(abbreviated by* pooledTripEx*) is the cooperative execution of a set of trips using a single car and a single driver.*

Notes:

1. In this context, it is assumed that each pooledTripEx is driven by exactly one driver. More complex cases are covered by [6].

Definition 8.2.7 (path similarity)**.** *Path similarity is a value in* $[0, 1]$ *assigned to an* ordered *pair* (pte_0, pte_1) *of* periodicTripEx *that indicates to what extent the OD (Origin, Destination) pairs involved in the respective trips, are compatible for carPooling in case the owner of* pte_0 *is assigned to be the driver.*

Notes:

1. Path similarity defines a function of *periodicTripEx* that is not symmetric in its arguments.

2. When the *single driver* constraint in *pooledTripEx* is dropped, paths driven no longer are strings of path segments but consist of *join* and *fork* trees which requires a more advanced concept of path similarity.

Definition 8.2.8 (time interval similarity)**.** *Time interval similarity is a value in* $[0, 1]$ *assigned to an ordered pair* (pte_0, pte_1) *of* periodicTripEx *having identical origins and identical destinations; it indicates to what extent the time intervals involved are compatible for carPooling.*

Notes:

1. Due to the *single driver* constraint (see definition 8.2.6, the route for each passenger's trip shall be included in the route for the *pooledTripEx* which is the route for the driver.

2. Time interval similarity can be calculated only for a pair consisting of the passenger trip and the part of the driver's trip for which the route coincides with the passenger trip route (because identical origins and destinations are required).

Definition 8.2.9 (sReputation)**.** *Safety reputation is a value in the range* $[-0.5, 0.5]$ *assigned to an* individual *(by the passengers) to qualify the individual as a safe driver.*

Notes:

1. *sReputation* is a characteristic of an individual because it is assumed that safe driving does not depend on the periodic trip driven.

2. Each individual keeps the *sReputation* value until modified by passengers after trip execution .

3. The initial value for individual i is $i.sReputation = 0.0$ (which means *neutral*).

Definition 8.2.10 (tReputation)**.** *Timeliness reputation (or accuracy reputation) is a value in the range* $[-0.5, 0.5]$ *assigned to a* periodicTripEx *in an* agreement*: it indicates to what measure the owning individual respects the timing when executing the periodic trip in the* agreement*.*

Notes:

1. *tReputation* is defined for both drivers and passengers.

2. *tReputation* has been defined as a characteristic of a tuple *(periodicTripEx,agreement)* and not as a characteristic of an *individual* or of a *prdTripEx* because an individual can behave differently on a specific *prdTripEx* pte_0 in different agreement contexts (pools). Example: the interval between a pick-drop activity to be executed by individual i_0 and the start of the *prdTripEx* in a given agreement a_0 is too short so that it is difficult for i_0 to meet the timing requirements of a_0. Within a different agreement a_1 timing constraints for pte_0 can be less severe so that the owner can met them easily.

3. The initial value for periodicTripEx *pte* in a is given by $(pte, a).tReputation = 0.0$ (which means *neutral*)

4. A individual keeps the *tReputation* value as long as the *periodicTripEx* exists and until modified by one of the co-travelling carpoolers.

Definition 8.2.11 (cohesion)**.** *Cohesion qualifies the strength of an* agreement *using a value in* $[0, 1]$ *that is a function of attributes of the* agreement *only.*

Notes:

1. An *agreement* with a high *cohesion* value is less likely to be broken whenever some of its *periodicTripEx* get a proposal to set up a new cooperation. Cohesion determines the resistance to breakdown in case opportunities for recombination come available.

2. *Cohesion* does not depend on *sReputation* since that is an attribute of an *individual* and not of an *agreement.*

3. The matcher shall derive *cohesion* from individual's negotiation feedback since individuals are assumed not to be prepared to specify and maintain *cohesion* values; furthermore, they are unable to do so since no universally valid scale or method is available.

8.3 Proposed Functions

Several specific functions can be used for the concepts defined in section 8.2. This section specifies concrete functions for a first implementation.

8.3.1 Profile Similarity Function

The `pfs` function defined in [2] will be used. `pfs` is based on the *euclidian distance* between normalized attribute values where each attribute has its specific weight.

8.3.2 Path Similarity Function

The `pts` function proposed in [2] will be used. `pts` gives the ratio between the path lengths for the driver in the *solo driving* and *carPooling* cases.

8.3.2.1 Experiment

1. An experiment has been set up where individuals are randomly selected from a `Feathers0` *FRAC1000* output as follows:
   (a) a list l_{35} of 35 subZones has been specified
   (b) an individual is added to a candidates list c_0 iff its home and all of its work locations are contained in the specified zones list l_{35}.
   (c) Finally a set s_{1000} of 1000 individuals (and hence schedules) are selected by uniform random sampling from c_0.
   (d) For each individual, the first *home-work* has been considered (hence one trip for each person). For each pair, *profile* (pfs) and *path* (pts) similarity have been calculated; a pair was kept iff the *profile* and *path* similarity exceeded a specified threshold. This results in a set s_{48} of 48 individuals.
2. The set s_{48} results in a *pts* matrix that applies to the first home-work trip in the schedules of the selected individuals. The first individual in the pair is the driver. The matrix is not symmetric. Diagonal elements have no meaning.
3. Hence we find $48^2 - 48 = 2256$ off-diagonal relevant values in the *pfs* matrix.

8.3.2.2 Beta distribution

1. The relative frequency diagram (bins sum to one) has been calculated : see fig. 8.3.
2. The mean m_s and variance v_s for the sample have been calculated. It is not known by the author whether or not they are unbiased estimators for the distribution mean m and variance v respectively.
3. For the beta distribution one has:

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)} \tag{8.2}$$

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \tag{8.3}$$

$$m = \frac{\alpha}{\alpha+\beta} \tag{8.4}$$

$$v = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \tag{8.5}$$

$$\tag{8.6}$$

hence

$$\alpha = m(\frac{m(1-m)}{v} - 1) \tag{8.7}$$

$$\beta = \alpha(\frac{1}{m} - 1) \tag{8.8}$$

Those formulas are also found on http://www.itl.nist.gov/div898/handbook/eda/section3/eda366h.htm .

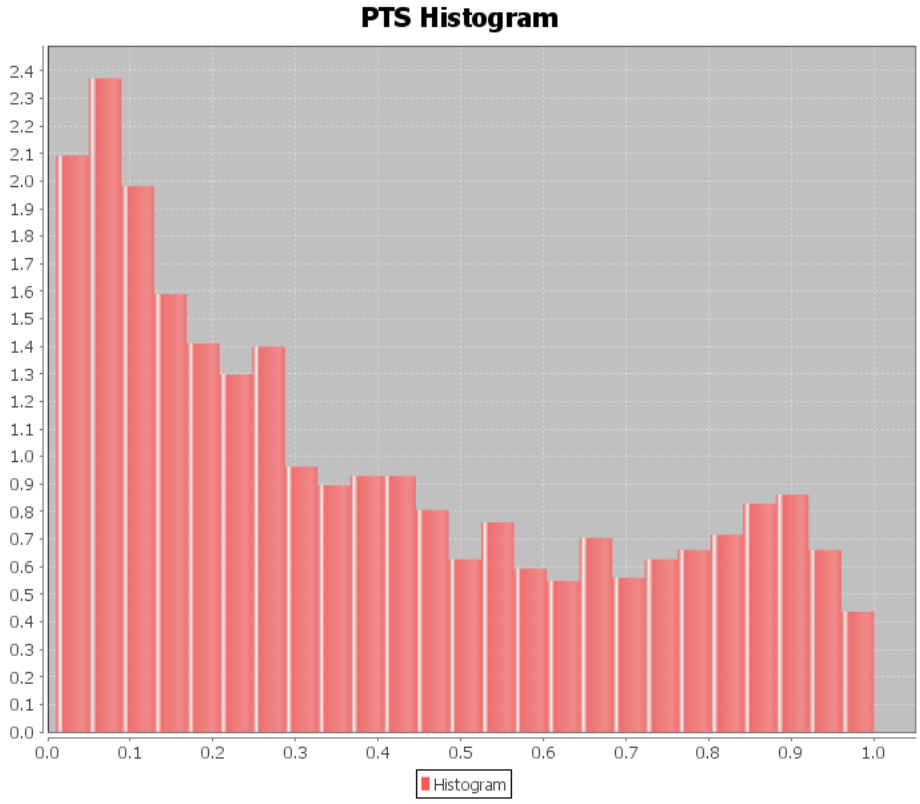

Figure 8.3: Path similarity histogram calculated for 2256 pairs. Compare to fig. 8.4.

4. Assuming $m = m_s = 0.384632$ and $v = v_s = (0.295080)^2$ we find $\alpha = 0.6609$ and $\beta = 1.057$. The resulting probability density is given in fig. 8.4.

8.3.3 Time Interval Similarity Function: obsolete version

TODO:LK See also JAIHC paper: that one contains a corrected version of this paragraph.

1. Both *departure* and *arrival* feasible intervals for the trip are used to determine the similarity.

2. Let $t_{b,\cdot}$ and $t_{e,\cdot}$ denote respectively *begin* and *end* times of intervals. Let $pte.i_d()$ and $pte.i_a()$ denote respectively the *departure* and *arrival* intervals of the periodicTripEx *pte*. Following function is proposed:

$$t_{b,max} = \max(t_{b,i_0}, t_{b,i_1}) \tag{8.9}$$
$$t_{b,min} = \min(t_{b,i_0}, t_{b,i_1}) \tag{8.10}$$
$$t_{e,max} = \max(t_{e,i_0}, t_{e,i_1}) \tag{8.11}$$
$$t_{e,min} = \min(t_{e,i_0}, t_{e,i_1}) \tag{8.12}$$
$$tis_{int}(i_0, i_1) = \max(0, \frac{t_{e,min} - t_{b,max}}{t_{e,max} - t_{b,min}}) \tag{8.13}$$
$$tis(pte_0, pte_1) = tis_{int}(pte_0.i_d(), pte_1.i_d()) * tis_{int}(pte_0.i_a(), pte_1.i_a()) \tag{8.14}$$

   The function $tis_{int}()$ is an interval overlap measure. Time interval similarity is defined as the product of the departure intervals overlap and the arrival intervals overlap.

3. This method does not produce sufficient results because the similarity value depends on both the lengths of the overlap period and the overall (union) period; the duration of the non-overlapping parts of the overall period actually is (almost) irrelevant in reality.

4. This similarity measure does not take into account any preference for specific periods: all times are considered to have identical preference values.

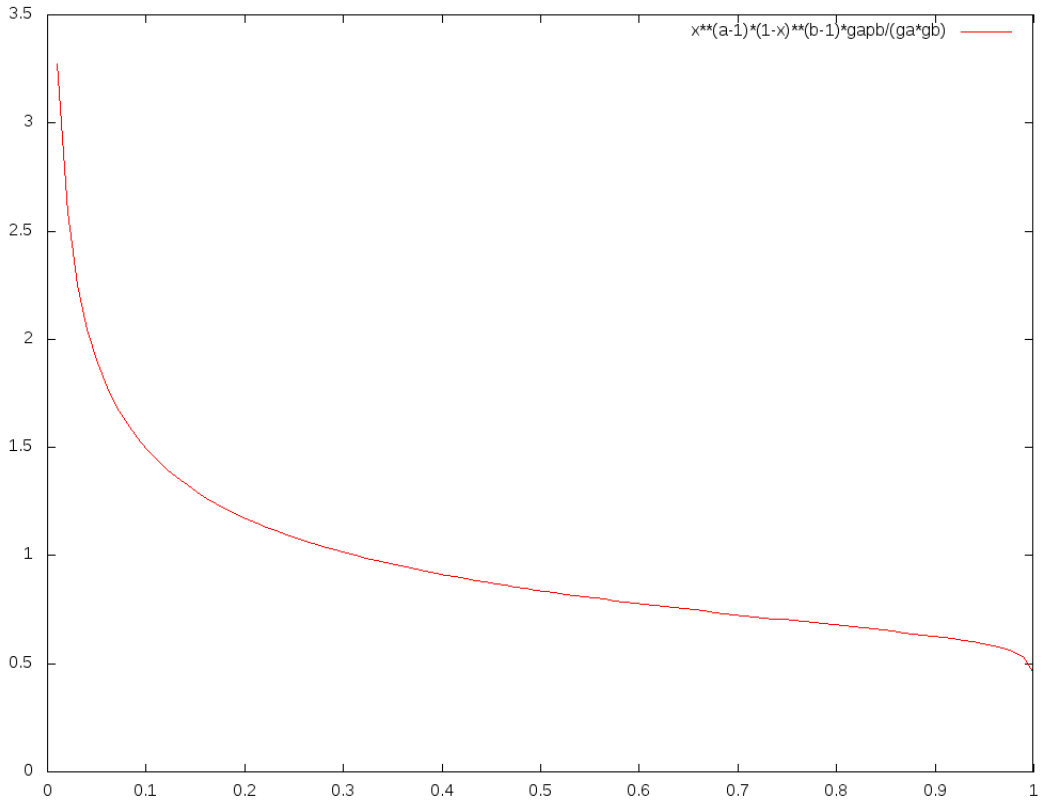

Figure 8.4: Path similarity beta density calculated for 2256 pairs. Compare to fig. 8.3.

5. This subsection describes a model that is not sufficiently realistic and hence has been replaced by 8.3.4 and 8.3.5

8.3.4 Time Interval Similarity Function for Negotiation

1. The *departure* (*arrival*) interval for a trip (*periodicTripEx*) is the time interval that suits the traveler to start (end) the trip. Let $pte.i_d()$ and $pte.i_a()$ denote respectively the *departure* and *arrival* intervals of the *periodicTripEx* pte.

2. Individual p_0's *preference* for a given moment in time is given by the function $f_{p_0} : \mathbb{R} \Rightarrow \mathbb{R} : t \mapsto f_{p_0}(t) \in [0, 1]$. The function is not required to be differentiable or continuous.

3. For each moment in time belonging to the departure and arrival intervals, the *preference* value has to be specified.

4. The *combined preference* function is the product of the preference functions associated with two *periodicTripEx*'s. It is essential to the negotiation purposes.

5. The integral of the preference over a time interval is called the *time interval suitability*.

6. *Combined time interval suitability* for intervals $i_A = [t_{i_A,0}, t_{i_A,1}]$ and $i_B = [t_{i_B,0}, t_{i_B,1}]$ is the defined by the integral of the *combined preference* (i.e. the product of the preference functions) over the overlapping part:

$$t_0 = \max(t_{i_A,0}, t_{i_B,0}) \tag{8.15}$$

$$t_1 = \min(t_{i_A,1}, t_{i_B,1}) \tag{8.16}$$

$$\int_{t_0}^{t_1} f_A(t) \cdot f_B(t) dt \tag{8.17}$$

7. The dimension of the *combined time interval suitability* value is $[prefUnit^2, timeUnit]$. In this context preference is assumed to be dimensionless hence the suitability dimension reduces to

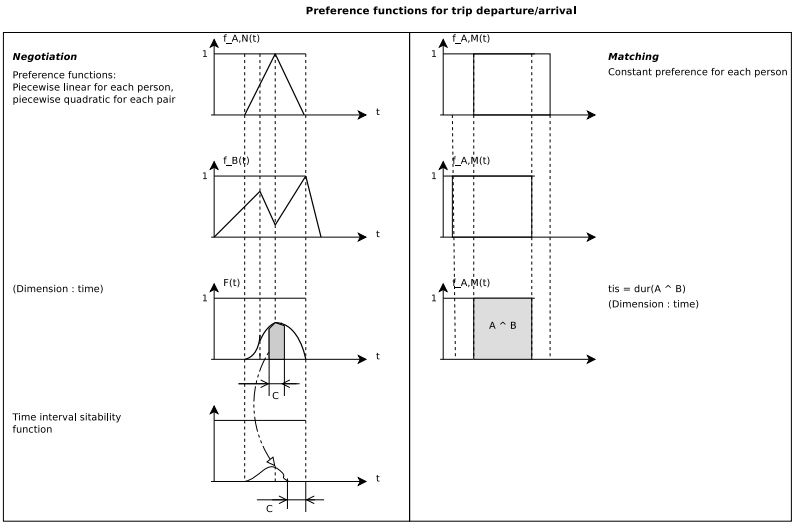

Figure 8.5: (Left) Time similarity for used while negotiating: $f_A(t)$ and $f_{AB}(t)$ are time preference functions for specific intervals. $F(t)$ is the *combined preference* and the size of the cross-hatched area is the resulting *combined time interval suitability*. (Right) Time similarity used by the matcher: all preference functions equal 1 because users are expected to only submit feasible time intervals.

$[timeUnit]$. The interpretations is as follows: the suitability specifies the equivalent amount of time available to jointly start (end) a trip (or in general to perform a shared action). The value can emerge either from a longer period of inconvenient moments in time or from a shorter period of convenient times.

8. Piecewise linear functions will be used because they can easily be specified by the user, integration is computationally cheap and they are flexible.

   (a) Consider a piecewise linear function $f(t)$ defined over an ordered set of knots $K = \{t_0 \dots t_{N-1}\}$: then $t \notin [t_0, t_1] \Rightarrow f(t) = 0$.

   (b) Assume two piecewise linear functions $f_A(t)$ and $f_B(t)$ defined over sets of knots $K_A = \{t_{A,0} \dots t_{A,N_A-1}\}$ and $K_B = \{t_{B,0} \dots t_{B,N_B-1}\}$ respectively. Consider the ordered set of consecutive knots, then the integral is given by

$$t_0, t_1, t_c \in K_A \cup K_B | (t_0 < t_1) \wedge (\nexists t_0 < t_c < t_1) \tag{8.18}$$

(8.19)

$$\tau(t_0, t_1) = \frac{(t_1 - t_0) \cdot (f_A(t_1) \cdot (2 \cdot f_B(t_1) + f_B(t_0)) + f_A(t_0) \cdot (f_B(t_1) + 2 \cdot f_B(t_0)))}{6} \tag{8.20}$$

Let $N = \#(K_A \cup K_B)$ the cardinality of the merged set of knots, then

$$\begin{aligned} \mathcal{S}(t_{min}, t_{max}) &= \sum_{j=0}^{N-2} \tau(t_j, t_{j+1}) \\ &= \sum_{j=0}^{N-2} \frac{(t_{j+1} - t_j) \cdot (f_A(t_{j+1}) \cdot (2 \cdot f_B(t_{j+1}) + f_B(t_j)) + f_A(t_j) \cdot (f_B(t_{j+1}) + 2 \cdot f_B(t_j)))}{6} \end{aligned} \tag{8.21}$$

(calculated using `mathomatic`). An example is shown in fig. 8.5.

9. Following are considered to be equivalent notations for the same concept (*combined time interval suitability*)
   $\mathcal{S}(i_A, i_B)$: expressed as a function of intervals
   $\mathcal{S}(K_A, K_B)$: expressed as function knots sets for intervals
   $\mathcal{S}(t_{min}, t_{max})$: expressed as function overlap interval boundaries

10. Combined time interval *suitability* is a value in $[0, \infty]$.

11. It is not yet clear whether or not a time interval *similarity* function having values in $[0, 1]$ is needed. It would be possible to use the *suitability* values as independent variables for the *logit* model (see section 11.4.1 item 4). In case time interval similarity is required, it can be defined as follows.

    (a) *Time interval similarity* is defined as a function of *combined time interval suitability* by

    $$tis(i_A, i_B) = \frac{\mathcal{S}}{\mathcal{S} + C} \tag{8.22}$$

    where C is an arbitrary constant. The actions considered are starting end terminating a carpool trip (resp. joining or leaving a pool of co-travellers) at given locations. C can be interpreted as the time needed for the specific action (at the specific location and day-of-the-week) and hence is a context specific reference value. Hence $tis(i_A, i_B) = 0.5$ corresponds with the minimal feasible $\mathcal{S}$ value (which equals C).

    (b) *Trip time interval similarity* for *periodicTripExecution*s is defined as the product of the time interval similarities for the departure and arrival intervals respectively.

    $$tis(pte_0, pte_1) = tis_{int}(pte_0.i_d(), pte_1.i_d()) * tis_{int}(pte_0.i_a(), pte_1.i_a()) \tag{8.23}$$

8.3.5 Time Interval Similarity Function for Matching

1. It is not feasible to ask the individuals to register the piecewise linear preference function mentioned in section 8.3.4. People are assumed to be prepared to register simply a time interval only. Hence the *preference* value is assumed to be a constant f over the time interval specified.

2. Formula 8.21 the reduces to

$$\mathcal{S}(t_{min}, t_{max}) = (t_{max} - t_{min}) \cdot f^2 \tag{8.24}$$

$$f = 1 \Rightarrow \mathcal{S}(t_{min}, t_{max}) = (t_{max} - t_{min}) \tag{8.25}$$

Since the constant value is the same for everyone, $f = 1$: then the time interval similarity reduces to the length of the overlap interval.

8.3.6 Safety Reputation of an Individual

1. For each individual, a list of qualification scores is kept.

2. A qualification score for individual i_0 can be submitted by individuals (different from i_0) who participated in at least one *agreement* with i_0. Everyone can supply at most one score for i_0.

3. The *sReputation* for i_0 is the average value of the supplied scores if any, else zero.

8.3.7 Timeliness Reputation of an Individual in a PeriodicTrip Context

1. For each *agreement* a matrix of mutual evaluation scores of participants is kept. Participants cannot evaluate themselves: hence the diagonal elements equal zero.

2. The range is for the scores is: $[-0.5, 0.5]$.

3. Scores are initialized to zero.

4. The *tReputation* of a participant is the average of the scores assigned to the individual.

5. This mechanism ensures that all participant evaluations share the same weight. Note that the *tReputation* can change simply by the fact that someone stops cooperating.

8.3.8 Cohesion of an agreement

1. *Cohesion* is supposed to be a monotonically decreasing function of the time elapsed since the creation of the *agreement*.

2. Note that cohesion does not depend on mutual evaluation of carpoolers; cohesion and reputation shall be independent concepts because all of them are fed into a probability estimator.

3. *Cohesion* is a monotonically decreasing function of *pool* size (large *pool*s are more likely to disintegrate).

4. Methods to determine the coefficients for cohesion functions still need to be defined.

Chapter 9

Agent Networks

9.1 Overview

In the *carPooling* context several relations (hence networks) are used:

SocNet	: The social network linking individuals by *acquaintance* relationship
PrivNet(i,dist)	: Part of *SocNet* containing individual i and the set of individuals who can be reached from i using at most *dist* steps
CoTravelRefused	: Enumeration of pairs of individuals who do not want to cooperate with each other for carPooling.
CpCandNet	: Network based on the relation that qualifies *periodicTripEx* as candidates to be combined for carPooling. This network is managed by the *Global Matching Service (GMS)*.
CpActNet	: Network based on the relation that links two individuals if and only if they are actually involved in carPooling.
CpHisNet	: Network based on the relation that links two individuals if and only if they ever carPooled.

Network sources

1. *SocNet* (for details: refer to 9.2) and *CpCandNet* (for details: refer to 9.3) are derived from daily *schedules* predicted by `Feathers`. Each daily schedule consists of a sequence of *episodes*. Each episode states startTime, duration and location (area) for an activity to be executed.

2. *CoTravelRefused* is constructed by individuals refusing to carPool with specific people (for details: refer to 9.4).

3. *CpActNet* and *CpHisNet* are derived from *SocNet* (for details: refer to 9.5).

9.2 Social Network (*SocNet*)

9.2.1 Building the *SocNet*

9.2.1.1 Basic Concepts Used

Following basic concepts are used to build the SocNet:

1. **Size law**: the observation that the number of people directly connected to, for most people is limited to about 150. A probability density for the number of direct connections in the social network is to be found. The probability weight function is denoted by $f_n(n)$.

   (a) in a first stage, a generally valid density is assumed

   (b) in later stages, the density can be made condiional on age, gender, educationLevel, ...

   (c) TODO:LK IF NONE CAN BE FOUND, A NORMAL DISTRIBUTION CAN BE USED AS AN APROXIMATION

2. **Distance law** (mentioned by Fosca Giannotti) stating the probability for two people to have a social relationship as a function of the distance between their respective home locations (probability decreases with distance). The probability density is given by $f_d(d)$. TODO:LK FIND THE DENSITY FUNCTION IN FOSCA'S PAPERS

3. **Profile similarity law**: the assumption that the probability to have a social relationship is positively correlated to the profile similarity (see definition 8.2.5 and section 8.3.1). If no density can be found in literature, we can take a triangular density function $f(pfs) = k \cdot pfs$ such that $f(0) = 0 \wedge f(1) = 2$. The probability densitity is denoted by $f_p(pfs)$ where pfs is the profile similarity.

9.2.1.2 Network Setup

Setting up the network of acquaintances for each agent goes as follows:

1. **Initial Phase**: builds the *out of the blue* (incidental) part of the *SocNet*

   (a) For each individual, connected individuals are sampled using the *size law*, the *distance law* and the *profile similarity* law. This phase samples the fraction f_i of acquaintances originating from random events e.g. being at the same school, attending a given conference, etc. Those are relationships that came *out of the blue* (incidental relationships).

   (b) Since connections are *incidental*, this part of the network can be built by considering mutually independent individuals.

   (c) Using the impedance matrix, a mapping is built:

   $$M_{d,z} : \mathbb{R} \Longrightarrow 2^{\mathcal{Z}} : d \mapsto y = M_{d,z}(d) \tag{9.1}$$

   where d is a distance and $\mathcal{Z}$ is the set of *TAZ*. y is the set of zones at distance d from the zone denoted by z. After sorting $domain(M_{d,z})$ it can be used to find zones at a distance sampled from the *distance law*.

2. **Expansion Phase**: builds the network by using transitivity (acquaintances that became connected via previously existing connections). This is the fraction $(1 - f_i)$ of acquaintances who entered the network by starting to know people via already existing connections (*via-via* mechanism).

   (a) New acquintaince is sampled at distance two in the already existing network. Thereby, the *profile similarity* is used but the *distance law* is ignored. The reason is : the probability to find an individual in PrivNet(i,2) i.e. the given set (of people at *SocNet* distance two from i) can be zero or very small which can result in respectively no solution or a very time consuming sampling phase (and hence performance problem).

   (b) This part of the *SocNet* depends on previously availabe asquaintances (links). Therefore, the set of individuals whose neighbourhood is still to be extended, shall not be processed in a deterministic order. One after another, individuals still needing more connections, are selected at random (using a uniform distribution) and assigned one more link. Let i_0 denote the selected individual. First, a profile similarity level is sampled, then an individual i_1 having sufficient profile similarity with i_0 is sampled from the neighbours at *SocNet* distance 2 from i_0. The pair (i_0, i_1) is added to the *PrivNet(i,1)* so that i_1 now is at distance one from i_0 and the set of neighbours at *SocNet* distance 2 from i_0 is extended.

   (c) Since uniform sampling is used, the neighbourhoods for individuals are expected to grow more or less at equal pace so thet their structures do not depend on the order of processing.

   (d) For any individual i_0, not only individuals i_1 belonging to *PrivNet(i,2)* immediately after the *initial phase* are added in the *expansion phase* because the neighbourhood at distance two is extended after each addition. Hence, during the expansion phase, neighbours of individuals added during the expansion phase can get selected.

3. **Verification phase**: after the complete *SocNet* has been set up, the network characteristics need to be determined in order to verify to what measure the result conforms to the laws mentioned in 9.2.1/9.2.1.1 since extending the *SocNet* using transitivity (the *via-via* mechanism) can break those laws. This investigation shall be performed for several values of f_i.

Algorithm 9.1 specifies helper functions. The network setup is outlined by algorithm 9.2. Notation used in the algorithms:

1. $f_u^w(s)$ denotes the uniform *pwf* over set s
2. $f_u^d([0,1])$ denotes the uniform *pdf* over interval $[0,1]$
3. $sample(f(s))$ denotes the action of sampling an element from set s using *pdf* or *pwf* (whichever applies).

Algorithm 9.1 SocNet builder helpers

```
function addElemsToSet(source, target, forbidden)
    for all e ∈ source \ forbidden do
        target ← target ∪ {e}
    end for
end function

function sampleUsingPfs(f(s), pfs_min)
    repeat                                          ▷ select random element from s
        e ← sample(f(s))
    until pfs(e) ≥ pfs                              ▷ Sufficient similarity
    return e
end function
```

9.2.1.3 Links Decay and Network Shrinking

1. Link decay with time is not modeled because this concept is not required in the carPooling application.
2. An individual never is removed from the SocNet (even not in case the individual is marked as having died): see section 9.5.2 and equation 9.7.

9.3 Global Matching Service Network (*CpCandNet*)

The *CpCandNet* is based on the *Carpooling Candidate* relation consisting of ordered pairs of *periodicTripEx*. The first element (t_0) in each pair is the *periodicTripEx* whose owner is a *preparedDriver*.

9.3.1 Carpooling Candidates Relation (*CpCandRel*) and Graph (*CpCandGraph*)

1. Let
   (a) tpe_0 and tpe_1 denote *periodicTripEx*s
   (b) pfs_{min} be a threshold value
   (c) pts_{min} be a threshold value
   (d) tis_{min} be a threshold value

Algorithm 9.2 SocNet builder

$f_i \leftarrow f_u([0,1])$ ▷ Sample fraction of incidental acquaintances
for all $i \in \mathcal{I}$ **do** ▷ For each individual
  $N_i \leftarrow sample(f_n(\cdot))$ ▷ determine network size
  $n_{i,i} \leftarrow f_i * N_i$ ▷ Size of incidental network
  $n_{i,v} \leftarrow N_i - n_{i,i}$ ▷ Size of via-via network

  $nNeeded \leftarrow n_{i,i}$ ▷ Number of incidental connections
  while $nNeeded > 0$ **do** ▷ Build incidental SocNet part
    $d \leftarrow sample(f_d(\cdot))$ ▷ Find distance between homes
    Select $\overline{d} \in domain(M)$ so that $|d - \overline{d}|$ is minimal
    $z \leftarrow sample(f_u^w(M(\overline{d})))$ ▷ Find TAZ at sampled distance
    $\overline{pfs} \leftarrow sample(f_p(\cdot))$ ▷ Sample minimal profile similarity
    $p \leftarrow sampleUsingPfs(f_u^w(synPopLivingAt(z)), \overline{pfs})$ ▷ Sample individual
    if $(i,p) \notin SocNet$ **then** ▷ Ok, a new one
      $addToSocNet((i,p))$ ▷ Add pair (i,p) to socNet
      $nNeeded \leftarrow nNeeded - 1$
    end if
  end while ▷ Enough incidental members added

  $i.S_2 \leftarrow \emptyset$ ▷ Build set of individuals at distance two
  for all $a_0 \in i.neighbours()$ **do**
    $addElemsToSet(a_0.neighbours(), i.S_2, \{i\} \cup i.neighbours())$ ▷ Exclude elems at dist. 0 and 1
  end for ▷ Set S_2 of individuals at distance two now ready
  $i.nNeeded \leftarrow n_{i,v}$ ▷ Number individuals to be sampled at distance two
end for ▷ All individuals processed: end of **initial phase**

$S_e \leftarrow \{j \in \mathcal{I} | j.nNeeded > 0\}$ ▷ Set of individuals needing more links selected transitively
while $S_e \neq \emptyset$ **do** ▷ More individuals need additional connections
  $i \leftarrow sample(f_u^w(S_e))$ ▷ Randomly select individual for which to add connection
  if $i.S_2 \neq \emptyset$ **then** ▷ Has available connections at distance two
    $\overline{pfs} \leftarrow sample(f_p(\cdot))$ ▷ Sample minimal profile similarity
    $v \leftarrow sampleUsingPfs(f_u^w(i.S_2), \overline{pfs})$ ▷ Randomly select at distance two
    if $v \neq NIL$ **then**
      $addToSocNet((i,v))$ ▷ Connect i and v in SocNet
      $i.nNeeded \leftarrow i.nNeeded - 1$
      $v.nNeeded \leftarrow v.nNeeded - 1$
      $i.S_2 \leftarrow i.S_2 \setminus \{v\}$ ▷ i no longer is at distance two from v
      $v.S_2 \leftarrow v.S_2 \setminus \{i\}$ ▷ v no longer is at distance two from i
      if $v.nNeeded = 0$ **then** ▷ $v \in S_e$
        $S_e \leftarrow S_e \setminus \{v\}$ ▷ Remove, has sufficient connections
      end if
      if $i.nNeeded = 0$ **then** ▷ $i \in S_e$
        $S_e \leftarrow S_e \setminus \{i\}$ ▷ Remove, has sufficient connections
      end if
      $addElemsToSet(v.neighbours(), i.S_2, \{i\} \cup i.neighbours())$
      $addElemsToSet(i.neighbours(), v.S_2, \{v\} \cup v.neighbours())$
    end if
  else ▷ Has no (more) connections at distance two
    $S_e \leftarrow S_e \setminus \{i\}$ ▷ Has small set of connections ($i.nNeeded$ connections unfulfilled)
  end if
end while ▷ End if **expansion phase**

avgPoolSize	:	2.2	:
fEffCarpoolPassengers	:	0.0656	: nHomeWorkTripsCarPsngr/nHomeWorkTripsNonSlow
fMatchableCands	:	0.5.	:
nCpCandidates	:	464589.	: nEffCarPoolers/fMatchableCands
nCpCandsStartingEachDay	:	255.	: nCpCandidates/nDaysInStartPeriod
nDaysInStartPeriod	:	1825.	: nYearsInStartPeriod * 365
nDriversReqd	:	105588.	: nHomeWorkTripsCarPsngr/(avgPoolSize - 1)
nEffCarPoolers	:	232294.	: nHomeWorkTripsCarPsngr * avgPoolSize/(avgPoolSize - 1)
nHomeWorkTripsCar	:	1599832.	:
nHomeWorkTripsCarPsngr	:	126706.	:
nHomeWorkTripsNonSlow	:	1930630.	:
nIndividuals	:	4791028.	:
nYearsInStartPeriod	:	5.0	:

Table 9.1: Named values and derived quantities relevant for the simulation.

Then *CpCandRel* is defined by

$$\begin{aligned}(tpe_0, tpe_1) \in CpCandRel \Leftrightarrow & (pfs(tpe_0, tpe_1) \geq pfs_{min}) \wedge \\ & (pts(tpe_0, tpe_1) \geq pts_{min}) \wedge \\ & (tis(tpe_0, tpe_1) \geq tis_{min}) \wedge \\ & (tpe_0.i(), tpe_1.i() \notin CoTravelRefused) \wedge \\ & (tpe_1.i(), tpe_0.i() \notin CoTravelRefused) \end{aligned} \tag{9.2}$$

2. The *CpCandGraph* is defined by (V, E) where $V \in \mathcal{T}$ and $E = CpCandRel$. *CpCandGraph* is a digraph since the *CpCandRel* is not symmetric because in general $pts(a, b) \neq pts(b, a)$.

9.3.2 Building the *CpCandNet*

Initially the *CpCandNet* is empty. It gets built up by running the simulator. This is done using a series of steps as indicated below. For some quantities we specify an order of magnitude or a specific real-life value valid for the simulated region (Flanders, Belgium). In order to keep the text readable, values have been given a name. The named values have been summarized in table 9.1. The values have been taken from a specific `Feathers0` *FRAC2* run for a monday.

1. Build the set of individuals who commute using *nonSlow modes* and load the carPool relevant part of their profile (attributes). This is a subset of the individuals who are nodes in the *SocNet*.

2. Let d_{start} denote the start date. Assign a to each individual a date sampled from a uniform distribution over the range $[d_{start}, d_{start} + nDaysInStartPeriod]$. This is the date at which the search for partners starts in case the individual decides to carPool.

3. Determine the fraction $fEffCarpoolPassengers = \frac{nHomeWorkTripsCarPsngr}{nHomeWorkTripsNonSlow}$: this is an estimate of the current fraction of effective carPool passengers among *nonSlowMode* commuters.

4. Estimate the poolSize *avgPoolSize* (see table 9.1). The required amount of drivers is

$$nDriversReqd = \frac{nHomeWorkTripsCarPsngr}{avgPoolSize - 1} \tag{9.3}$$

The estimated number of effective carPoolers is :

$$\begin{aligned} nEffCarPoolers &= nDriversReqd + nHomeWorkTripsCarPsngr \\ &= nHomeWorkTripsCarPsngr * \frac{avgPoolSize}{avgPoolSize - 1} \end{aligned} \tag{9.4}$$

5. Estimate the fraction of carPool candidates who can find a match : *fMatchableCands* (see table 9.1). Then the required number of candidate carpoolers is : $nCpCandidates = \frac{nEffCarPoolers}{fMatchableCands}$

6. Select *nCpCandidates* individuals by sampling from the set established in step 1 using a uniform distribution and mark them as carPool candidates. Note that the individuals also are *SocNet* nodes: so some of the *SocNet* nodes now have been marked.

7. Verify that at least *nDriversReqd* of the *nCpCandidates* individuals are *preparedDriver*s i.e. have car and drivers license available. This should not be a problem because we sampled from the population performing nonSlowMode commuting.

8. Start the simulation.

9. For each day in simulation time,

$$nCpCandsStartingEachDay = \frac{nCpCandidates}{nDaysInStartPeriod} \tag{9.5}$$

   individuals start looking for carPool partners.

10. When simulation time reaches the start date for an individual, the individual first performs *local* exploration (see 10.1/2) trying to find a carPool partner for each commuting trip; if no *agreement* could be negotiated with any of the available candidates, the commuting trip is registered with the *GCPMS* (see 11.3.1/3c).

9.4 CoTravelRefused Relation

CoTravelRefused $R_{CoTravelRefused} \subset \mathcal{I} \times \mathcal{I}$. This relation indicates whether or not two individual can cooperate in an *agreement*: see also 11.3.2.2/4 , 11.3.1/3l and 11.3.1/3m

9.5 *CpActNet* and *CpHisNet*

9.5.1 Network of Actual CarPoolers (*CpActNet*)

1. *CpActNet* is defined by the relation *CpActRel*.

2. *CpActRel* is the set of all pairs of individuals actually involved in an *agreement* for carPooling. Formally: let $i_0, i_1 \in \mathcal{I}$ denote individuals, then CpActRel is defined by:

$$(i_0, i_1) \in CpActRel \Leftrightarrow \exists a \in \mathcal{A} | (t_0, t_1 \in a.T() \wedge t_0 \neq t_1 \wedge i_0 = t_0.i() \wedge i_1 = t_1.i()) \tag{9.6}$$

9.5.2 Network of Actual and Former CarPoolers (*CpHisNet*)

1. Every time a pair is added to *CpActNet*, it is added to *CpHisNet* too. Elements never are removed from *CpHisNet*: hence $CpActNet \subseteq CpHisNet$.

2. Since effective carPooling partners are added to *SocNet* (see section 11.1.2/5) following holds:

$$CpActNet \subseteq CpHisNet \subseteq SocNet \tag{9.7}$$

9.6 Network Characteristics Reporting

This section specifies the list of network characteristics to be reported for analysis in order to understand the structure of the respective networks (*SocNet*, *CpActNet*, *CpHisNet*, *CoTravelRefused*, *CpCandNet*).

1. Notes and Terms

   (a) *pdf* denotes probability density for continuous variables

   (b) *pwf* denotes probability weight function for discrete variables

(c) *mcc* denotes maximal connected component

(d) *smcc* denotes the set of *maximal connected components* of the complete *CpCandNet*

2. Following list of characteristics for the complete *CpCandNet* network is to be calculated for investigation (in order to understand the structure of the network)

   (a) the number of *maximal connected components* (cardinality of *smcc*)

   (b) *pwf* for the node degree

   (c) the number of pairs linked uni/bi-directionally

   (d) the fraction $\frac{nUniDirLinks}{nUniDirLinks+nBiDirLinks}$

   (e) *pwf* for number of *prepared driver*s available to each individual

   (f) *pwf* for number of individuals a specific individual is a *prepared driver*s: designated by $nPotentialPassengers$

   (g) *pwf* for $\frac{nPotentialPassengers}{carSize}$ (where $carSize$ is specific for each individual prepared driver)

   (h) *pdf* for the ratio $\frac{newDist}{origDist}$ where $origDist$ denotes the distance driven individually for path p_b in an ordered path pair (p_a, p_b) of similar paths, and $newDist$ denotes the distance driven during a carPooling trip covering both p_a and p_b. Refer to [2] for details on the respective path length calculations.

3. Following list of characteristics for the set of maximal connected components (*smcc*) of the complete *CpCandNet* network is to be calculated for investigation.

   (a) a table showing the number of links and the number of nodes for each maximal connected component

   (b) *pwf* for the size *mcc*

   (c) scatter plot containing (nNodes,nLinks) tuple for each maximal connected component

   (d) the *pdf* for ratio $\frac{nLinks}{nNodes}$

   (e) the fraction of the *complete maximal connected subnets* (the fraction of maximal connected subnets where everyone is connected to everyone else)

4. In case the number of *maximal connected components* turns to be very small (order of magnitude = 10), following list of characteristics shall be reported for each *maximal connected component*:

   (a) same as for item 9.6/2

9.7 Notes

Since *SocNet* and *CpCandNet* are mutually independent concepts, a given ordered pair of individuals can belong to none, one or both of the relations.

Chapter 10

Pool Building Concepts

10.1 Exploration and Matching - UseCase

1. Individuals considering *carPooling* need to find partners. Each individual can decide to look for carPooling partners at any moment in time. They try to do so by exploring first their set of acquaintances and in a second stage a larger network of candidates presently possibly unknown to the exploring individual.
2. The first (*SocNet* based) method is called *local exploration*.
   (a) Each individual can propose to carPool to each *individual* directly connected (hence at distance 1) individual in *SocNet*.
   (b) Furthermore, the social network is explored up to a distance of N (order of magnitude: $N = 2$) to account for *via-via* information. Individuals set up bilateral contacts at their own discretion. N is to be sampled from a rapidly decreasing *pwf* so that $prob(N = 3) \approx 0$.
3. The second (*CpCandNet* based) method is called *global exploration*. It is assumed to be performed by a *Global CarPooling Matching Service (GCPMS)*. Such service is assumed to exist as a globally available web based service (like http://www.carpoolplaza.be/.
   (a) It is assumed that the large scale *matching* described in chapter 11 is performed as an integrated part of this kind of service.
   (b) The matcher is activated
      i. each time an individual registers
      ii. each time apool disintegrates

   At each activation, the matcher tries to combine existing *pool*s in order to optimize an objective function (see section **??**). Note that if groups would not disintegrate, the matcher only would have to find an optimal match for each newcomer.

10.2 *Pool* disintegration

1. After a period of time determined by stochastic sampling, a randomly selected individual leaves the *pool* for a reason that cannot be explicitly modeled (example reasons are : changing work habits, new working times imposed by employer, ...).
2. At random times, people change *home* and/or *work* locations and hence leave *pool*s.
3. The probability that an individual i_0 accepts an offer to move from *pool* p_0 to *pool* p_1
   (a) decreases with the *cohesion level* of p_0
   (b) increases with the $gain(i_0, p_0, p_1) = cost(i_0, p_1) - cost(i_0, p_0)$
4. At random times, individuals can notify the service using the `looksForAlternative()` (see 11.3.1/3i) method that cohesion for a specific *agreement* has decreased.

10.3 Pool Similarity Matrices

1. Similarity functions for agent profiles (*pfs*), paths (*pts*) and time intervals (*tis*) have been defined (see 8.2.5, 8.2.7, 8.2.8).

2. Consider two *pool*s p_0 and p_1 and their respective *periodicTripEx* sets T_0 and T_1. Then consider the members of $T_0 \times T_1$.

   Definition 10.3.1 (similarityMatrix)**.** *S is a similarityMatrix for pools p_0 and p_1 if and only if* $(sim \in \{pfs, pts, tis\}) \wedge (\forall t_0 \in p_0.T(), t_1 \in p_1.T() : S[t_0, t_1] = sim(t_0, t_1))$

3. *pfs* and *tis* similarity matrices are symmetric, *pts* in general is not.

10.4 Pool Size

1. Pool size is limited to 2 (carPooling by pairs) in the first implementation for practical reasons related to matching. This case requires pair matching only; hence regular graphs can be used: see also section 11.3.

2. Later cases will consider larger pools, limited by the car capacity. An optimal set partitioning problem the is to be solved. Hypergraphs (hyperEdges) need to be considered.

Chapter 11

Matching - Finding CarPool Partners

11.1 Introduction

11.1.1 Exploration/advisory and Negotiation Phases

Matching is applied in both *local* and *global* exploration/advisory phases. For definitions: refer to sections 10.1. *Matching* is done in the exploration/advisory phase and thus precedes the negotiation phase where final decisions to carPool are taken. Mechanisms used in the exploration/advisory phases shall be consistent with mechanisms in the negotiation phase. It is not possible to predict the negotiation phase with certainty (if it would, the negotiation phase would be superfluous); reasons are:

1. Negotiation covers drivers selection, co-route determination and re-scheduling (schedule adaptation) for the cooperators. Schedule adaptation makes use of VOT (individual specific Value Of Time). An advisory mechanism does not have all required data available nor has any knowledge of the private goals (and in general the *BDI* (Beliefs, Desires, Intentions)) of the individuals (agents) involved in a negotiation.

2. The total distance driven cannot be predicted by the *matcher* when carPool parkings are involved because in such cases the co-route can be tree structured: see chapter 4 item 4.2/5. Hence the path similarity function delivers only an approximation of the data involved in negotiation.

11.1.2 Principle of Operation of the CarPooling Model

1. Informally, a *pool* corresponds to a set of people carPooling (see also formal definition 8.2.3). An individual can be represented by a *pool* containing one element.

2. An individual looks for other individuals to cooperate while executing *periodicTripEx*s: this is called *exploration*.

3. *Local* exploration is applied before *global* exploration. If *carPooling* partners can be found within an individual's *SocNet neighbourhood*, they will be contacted first (as preferred candidates).

4. *Global* exploration is applied only in a second stage when no suitable *pool* was found in the *SocNet*. In the *Global* exploration phase, the *matcher* provides advice about which *pool*s an individual should negotiate with. This corresponds to the use of an online service by a candidate exploring the set of formerly unknown carPooling candidates.

5. If *pool*s are merged, each pair of involved *individual*s is added to the *SocNet* of every other participant in the *pool* (if still required) so that if i_0 and i_1 cooperate in a *pool*, (i_0, i_1) and (i_1, i_0) belong to each others *SocNet*. Because links never are removed from the *SocNet*, if i_0 and i_1 ever carPooled, $(i_0, i_1) \in SocNet$.

6. Candidates register, join and leave *pool*s at random moments in time. As a consequence the main data structures dynamically change due to events external to the matching process.

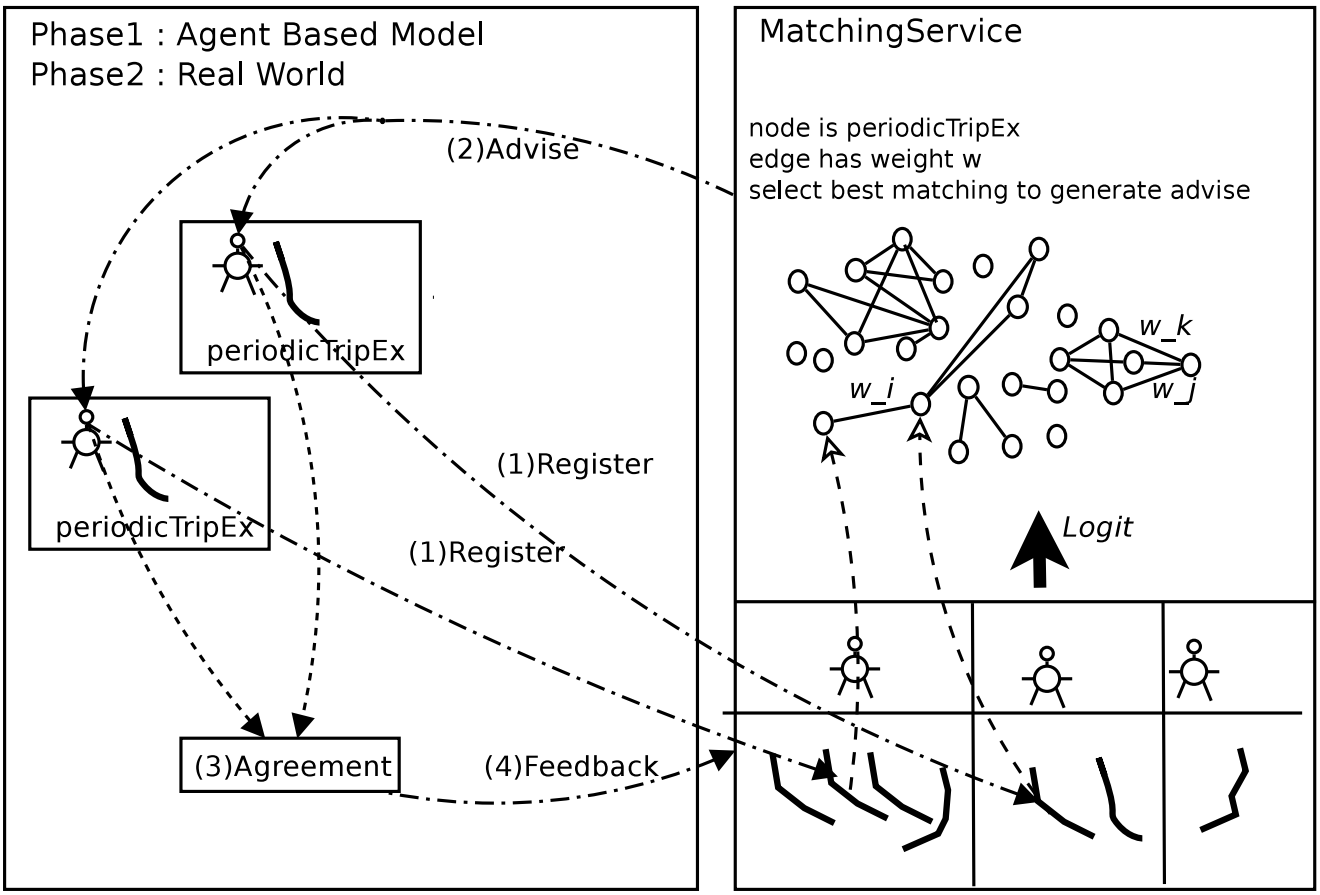

Figure 11.1: Application context: the right hand side shows the *matcher service*. People register some descriptive data about themselves and trips to be executed periodically (*periodicTripEx*). Those constitute a graph : the edges are labeled with the probability that negotiation will succeed when the trip owners are advised to carpool. Negotiation result is fed back to train the *logit* predictor. The left hand side shows the entities execising the *matcher service* in consecutive phases.

11.2 *Local* Exploration and Matching

TODO:LK

11.3 *Global* Exploration and Matching - Overview

For a definition of *matching* refer to chapter 10. The context where the *global matching* mechanism is used, has been described in 10.1/3a.
This section does not limit the size of the *pool*s: it contains general statements. Statements about the simpler pairwise matcher can be found in 11.4. Refer to figure 11.3 and figure 11.1.

11.3.1 Operations

1. The matching service is aimed to support users looking to set up *agreements* for *carPool commuting* (i.e. *agreements* for a long *term periodic carPooling* that starts a given time and stops when explicitly terminated). An individual can negotiate multiple simultaneaously existing *agreements*: e.g. every tuesday home-work-home with partner A and every thursday home-work-home with *pool* consisting of partners B and C.

2. In order to provide sensible advice on partners to match, the service needs sufficient and accurate data. It is assumed that a sensible business model is used (e.g. mutual payment is via the service) so that users are willing to provide the required data.

3. Hence, the service is assumed to support following operations. The operations are listed below in order to specify what data are available to the matching service at any moment in time and to indicate how data evolve over time. The first argument in each operation, identifies the issuer (caller).

   (a) `Cand registerCandidate(issuer,socDemo)`
   Register *issuer* with its socio-demographic data *socDemo* as a candidate for carPooling. This method can be used to re-register a previously unregistered individual.
   PreCond: *issuer* not yet registered

   (b) `void unregisterCandidate(issuer)`
   Unregister *issuer*: notifies the *GCPMS* that the issuer no longer is interested in any advice from the *GCPMS*.
   PreCond: *issuer* registered before

   (c) `Pool registerTrip(issuer,periodicTripEx,fromTs,canDrive)`.
   Add *periodicTripEx* for *issuer* for a period starting at $fromTs$ and indicate whether or not *issuer* can be the driver if carPooling is negotiated. Note that this returns a *pool* containing a single *periodicTripEx* for the time being.
   PreCond: *issuer* is registered individual, valid *periodicTripEx*, *issuer* is *periodicTripEx* owner, *fromTs* in future

   (d) `void unregisterTrip(issuer,periodicTripEx)`.
   Remove *periodicTripEx* for *issuer* from the *GCPMS*.
   PreCond: *issuer* is registered individual, valid *periodicTripEx*, *issuer* is *periodicTripEx* owner, *periodicTripEx* not covered by an *agreement*: if necessary, `leavePool()` is to be used to remove *periodicTripEx* from an *agreement*

   (e) `Pool getSuggestedPool(issuer,periodicTripEx)`
   Returns a *pool* for which it is worthwhile to negotiate to join. This is the resulting advice produced by the *matcher*.
   PreCond:*issuer* is registered individual, *issuer* owns the *periodicTripEx*

   (f) `Boolean mergePools(issuer,agreement,fromTs)`
   Notifies that a new pool is to be constructed from *agreement* . This operation is performed to register a newly negotiated *agreement* with the *GCPMS*. The *pool*s containing the *periodicTripEx* referred to by the *agreement*, cease to exist (since a *periodicTripEx* can be part of at most one *pool*).
   PreCond:*issuer* owns exactly one element referred to by the *agreement*, *agreement* is the result of a succeeded negotiation, method to be executed by exactly one participant in the *agreement*
   PreCond: a neccessary but not sufficient condition is that there is at least one *preparedDriver* among the *individual*s (more than one driver can be required)
   PostCond: Each *periodicTripEx* belongs to exactly one *pool*

   (g) `Boolean splitPool(issuer,pool,agreementsList)`
   Split existing *pool* into a set of pools according to the *agreements* specified in *agreementsList*. Note that a pool consisting of a single *periodicTripEx* can have a *nil agreement* : it is always possible to specify some regular agreements and the correct amount of `nil` *agreements* to split a *pool* in an unambiguous way.
   PreCond: number of *pool* participants equals the number of *periodicTripEx* covered by the regular *agreement*s plus the number of `nil` *agreements* specified in *agreementsList*

   (h) `Boolean leavePool(issuer,pool,periodicTripEx,fromTs)`
   Remove *periodicTripEx* from *pool* so that cooperation stops at $fromTs$.
   PreCond:*issuer* is *periodicTripEx* owner

   (i) `Boolean looksForAlternative(issuer,agreement)`
   The issuer executing the operation notifies the system that (s)he did not yet leave the specified *agreement* but intends to do so as soon as a good alternative is proposed. This lowers the *cohesion* of the *agreement*.

PreCond: *issuer* has been registered before, *issuer* participates (i.e. owns a *periodicTripEx*) in *pool* covered by the *agreement*

(j) `Boolean reConfirm(issuer,agreement)`
The issuer executing the operation notifies the system that (s)he does not want to receive suggestions to leave the agreement (due to decreased cohesion) in the near future. This resets the *cohesion* of the *agreement* to the initial level (hence it is assumed that *issuer* confirms on behalf of all participants in the *agreement*).
PreCond: *issuer* has been registered before, *issuer* participates (i.e. owns a *periodicTripEx*) in *pool* covered by the *agreement*

(k) `Boolean evalPartner(issuer,partner,periodicTripEx,score)`
Register evaluation *score* for a given *partner* on a *periodicTripEx*
PreCond: both *issuer* and *partner* have participated (owned a *periodicTripEx*) in *pool*

(l) `void excludePartner(issuer,partner,toTs)`
Individual *issuer* refuses to cooperate with *partner* in any carPooling trip up to time $toTs$. The excluded individual (*partner*) is not necessarily explicitly notified of the exclusion.
PreCond: both *issuer* and *partner* are registered individuals
PreCond: `toTs` in the future

(m) `void unExcludePartner(issuer,partner)`
Individual *issuer* no longer refuses to cooperate with *partner* in any carPooling trip. The unExcluded individual (*partner*) is not necessarily explicitly notified of the end of the exclusion.
PreCond: both *issuer* and *partner* are registered individuals

4. Note that because not necessarily $CpCandNet \cap i.neighbours() = \emptyset$, individuals who registered with the global carPool matching service (*GCPMS*) and decide to carPool as a consequence of *local exploration*, are required either to unregister (all of them) or to notify the global service by executing a `mergePools()` operation. Without this registration, the global service will provide wrong advice due to lack consistent data. This is one of the reasons we need to assume that the global service has a sufficient business model.

5. Not every individual is expected to react after having received an advice or to register a failed negotiation. Hence the *GCPMS* is assumed to keep track of a list of *pending proposals*. A time-out mechanism is assumed to be in use in order to mark negotiations corresponding to pending proposals as having failed.

11.3.2 Problem Statement

11.3.2.1 Set Partitioning

1. Consider a set $T \subseteq \mathcal{T}$ of *periodicTripEx*. The objective of the service is to optimally assign the elements into n-tuples. Each such n-tuple corresponds to a set of individuals carPooling for a specific one of their *periodicTripEx*s. The problem comes down to find a set partition that is optimal with respect to given criteria while fulfilling a set of constraints.

2. Evolution over time

   (a) Characteristics of *periodicTripEx* and *agreements* change over time (see 8.3.8).

   (b) The set T evolves over time: elements get added to and removed from T.

   (c) Constraints evolve over time since characteristic values of subsets (pools, partition cells) evolve over time.

   Hence the problem is reduced to deriving a new partition from an existing one.

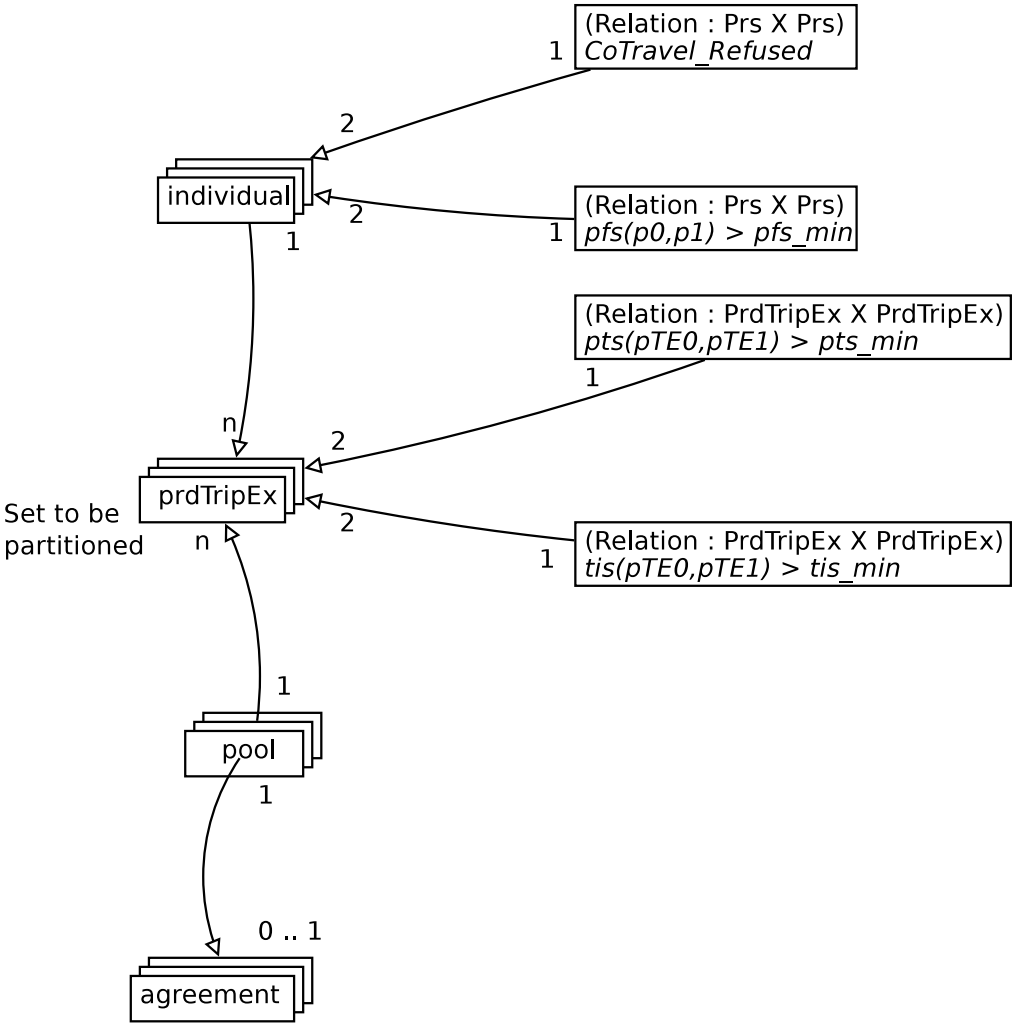

Figure 11.2: Overview of sets (*individual*, *prdTripEx* and *agreement*) used in the model and relations those sets are involved in.

11.3.2.2 Relations

Refer to fig. 11.2

1. Profile similarity: symmetric, reflexive, non-transitive

$$pfs(i_0, i_0) = 1.0 \tag{11.1}$$
$$pfs(i_0, i_1) = pfs(i_1, i_0) \tag{11.2}$$
$$R_{pfs} \subset \mathcal{I} \times \mathcal{I} \tag{11.3}$$
$$(i_0, i_1) \in R_{pfs} \Leftrightarrow (pfs(i_0, i_1)) \geq \overline{pfs}) \tag{11.4}$$

2. Path similarity: non-symmetric, reflexive, non-transitive

$$pts(t_0, t_0) = 1.0 \tag{11.5}$$
$$R_{pts} \subset \mathcal{T} \times \mathcal{T} \tag{11.6}$$
$$(t_0, t_1) \in R_{pts} \Leftrightarrow (pts(t_0, t_1)) \geq \overline{pts}) \tag{11.7}$$

3. Time interval similarity: symmetric, reflexive, non-transitive

$$tis(t_0, t_0) = 1.0 \tag{11.8}$$
$$tis(t_0, t_1) = tis(t_1, t_0) \tag{11.9}$$
$$R_{tis} \subset \mathcal{T} \times \mathcal{T} \tag{11.10}$$
$$(t_0, t_1) \in R_{tis} \Leftrightarrow (tis(t_0, t_1)) \geq \overline{tis}) \tag{11.11}$$

4. CoTravelRefused $R_{CoTravelRefused} \subset \mathcal{I} \times \mathcal{I}$. This relation indicates whether or not two individual can cooperate in an *agreement*. This relation is maintained using operations `excludePartner()` (see 11.3.1/3l) and `unExcludePartner()` (see 11.3.1/3m)

5. Drivers $F_{drivers} : 2^{\mathcal{P}} \Rightarrow 2^{\mathcal{I}} : p \in \mathcal{P} \mapsto \{i | \exists t \in p.T() : t.i() = i\}$ where $\mathcal{P}$ denotes the set of all *pool*s. This functions returns the set of drivers in a *pool*.

11.3.2.3 Constraints

1. A pool shall contain at most one *periodicTripEx* for each individual.

2. The number of *periodicTripEx*s in a pool shall be less than the largest one of the *preparedDriver*s car sizes.

3. Incompatible individuals shall not get advice to carPool: hence they shall not be linked in the *CpCandNet* (see section 9.3.1).

$$(i_0, i_1) \in R_{CoTravelRefused} \Rightarrow (\nexists (pte_0, pte_1) | pte_0.i() = i_0 \wedge pte_1.i() = i_1 \wedge (pte_0, pte_1) \in CpCandRel) \tag{11.12}$$

11.3.2.4 Optimisation - Preferential Grouping

1. Subsets of *individual*s cooperating on an *agreement* a_0 in a *pool* with high cohesion level, are preferred partners to cooperate in other pools (for other *periodicTripEx* (of course)). Matching shall try to use such maximal subsets when creating new pools. See also section 11.4.3

2. The same idea applies to proposed cooperations. Hence proposals issued but not yet confirmed to have evolved to an *agreement*, shall be considered too. This statement is based on the assumption that when someone receives two sets of proposals, (s)he will chose the one where the total amount of individuals involved, is minimal.

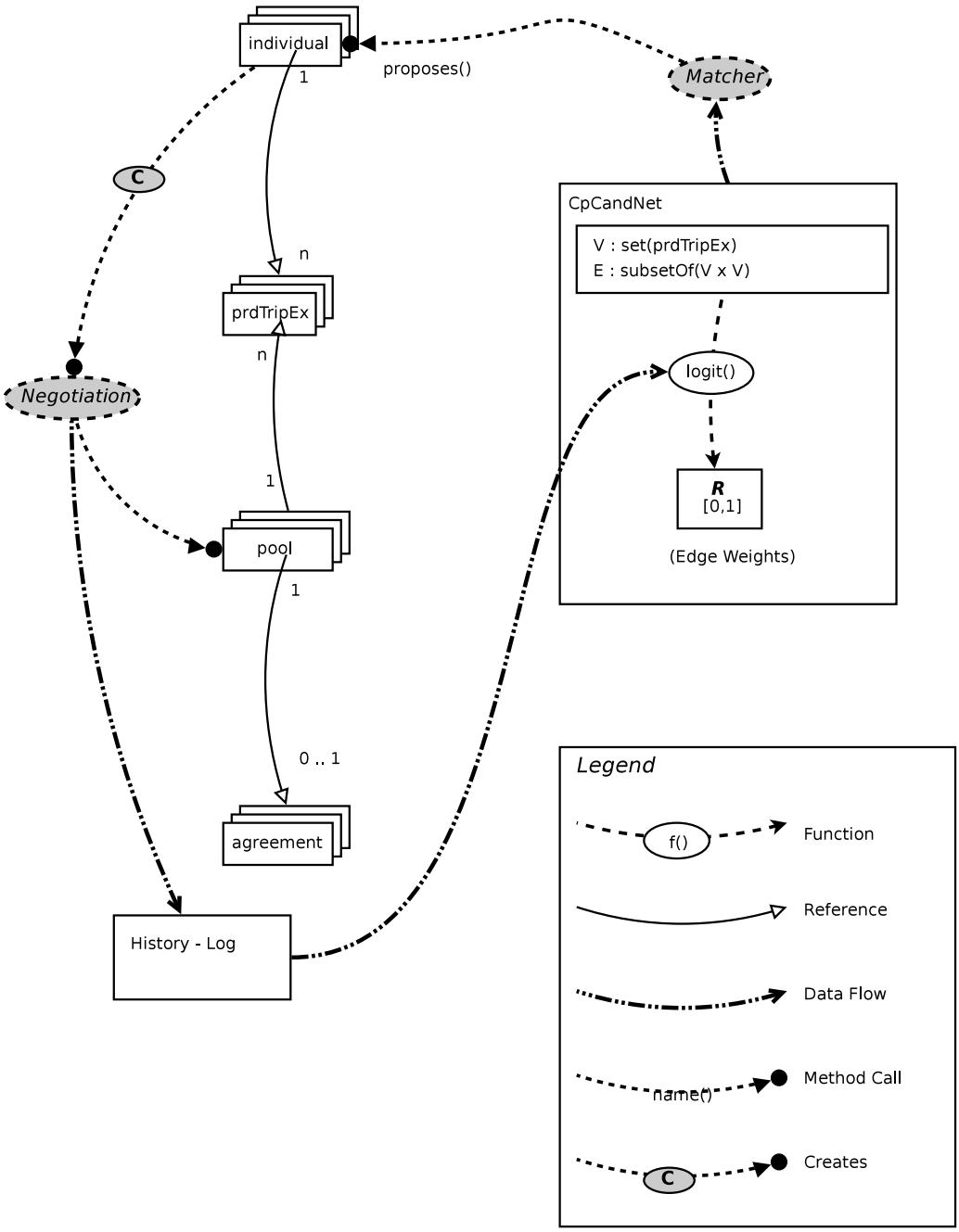

Figure 11.3: Matching model overview showing data flows, relations and method activations.

11.4 *Global* Exploration and Matching - Pairwise Matcher

11.4.1 Negotiation Outcome Prediction

Refer to fig. 11.3 for an overview of data flows, relations, method activations.

1. The outcome of a negotiation is a discrete variable with values : *success* (yes) and *failure* (no).
2. Independent variables influencing the negotiation are continuous : *pfs*, *pts*, *tis*, *cohesion* (c_0 and c_1) and *sReputation* (r_0 and r_1). In the pairwise case
   (a) exactly one *sReputation* value applies since the graph is directed, each edge points to the driver and *sReputation* applies to the driver: hence a single *sReputation* value applies to each edge.
   (b) The *tReputation* of an individual i_0 applies to an existing *agreement* and only exists as long as the agreement holds. It can only be affected by the partners in the agreement different from i_0 (in the pairwise case, there is only one such partner). The *tReputation* is an evaluation score assigned by the partners; the default value equals the neutral value : see definition 8.2.10.
   (c) exactly two *cohesion based indicators* apply (one for each of the *pool*s the candidates belong to (if any)). Each of the cohesion values applies to an *agreement* and for a given *periodicTripEx* pair c_0 and c_1 can relate to either a single or different *agreements*. The meaning of the tuple (c_0, c_1) thus depends on whether or not c_0 and c_1 apply to the same *agreement* or not. Therefore, it is proposed to combine cohesion values into a single one using the function given in equation 11.15; the first case in equation 11.15 corresponds to the case where both *periodicTripEx* are partners in an *agreement*. Let $pte_0, pte_1 \in \mathcal{T}$ the *periodicTripEx*s involved. Let c_0 and c_1 denote the respective *cohesion* values corresponding. Let $pte.a()$ denote the *agreement* covering *pte* when it belongs to a *pool*.

$$pte_x.a() = \begin{cases} nil & if \quad \nexists p \in \mathcal{P} | pte_x \in p.T() \\ p.a() & if \quad \exists p \in \mathcal{P} | pte_x \in p.T() \end{cases} \tag{11.13}$$

$$c_x = \begin{cases} 0 & if \quad pte_x.a() = nil \\ pte_x.a().c() & else \end{cases} \tag{11.14}$$

$$\overline{c} = \begin{cases} c_0 * c_1 & if \quad pte_0.a() = pte_1.a() \neq nil \\ (1 - c_0) * (1 - c_1) & if \quad pte_0.a() \neq pte_1.a() \end{cases} \tag{11.15}$$

   In case both *periodicTripEx* belong to the same *agreement*, the cohesion values are taken from that *agreement* and in fact $c_0 = c_1$. In the other case (which also covers the case where at least one of the *periodicTripEx* is not covered by an *agreement*), the complement of the *cohesion* values is used. When neither of the *periodicTripEx* belongs to an *agreement*, $\overline{c} = 1$.
3. Since the driver is known for each edge in the graph, an estimate can be calculated for
   (a) d_{solo} : the total distance driven when both *periodicTripEx* use solo-driving
   (b) d_{pool} : the total distance driven when the *periodicTripEx* use carPooling
   (c) $d_{gain} = (d_{solo} - d_{pool})$: the difference of previous ones (distance not driven when carPooling)

   A financial cost can be associated with those distances.
4. A `logit` model will be used to predict the negotiation outcome. Negotiation results fed back to the *Global CarPooling Matching Service (GCPMS)*, are used to determine the coefficients for the `logit` model by linear regression.

11.4.2 Objective Functions and CpCandNet Edge Weights

1. Edges in *CpCandGraph* are assigned a weight based on the probability for the negotiation to carpool for the *periodicTripEx*s denoted by the respective nodes.

$$q(e) = k(e) * prob_{logit}(success | pfs, pts, tis, \overline{c}, r_0) \tag{11.16}$$

2. The value for $k(e)$ depends on the objective function chosen. Let E denote the set of edges in the *CpCandGraph*. Objective functions of the form

$$o(\mathcal{T}) = \sum_{e \in E} k(e) * q(e) \tag{11.17}$$

calculate the expected value of the function $k(e)$.

$k(e) = 1$	maximizes the expected value for the number of negotiated *agreements*
$k(e) = d_{pool}(e)$	maximizes the expected value for the distance driven using carPooling
$k(e) = d_{gain}(e)$	maximizes the expected value for the distance not driven due to carPooling (the gain)

Note that the matcher is not aware of the *value* of an agreement (e.g. the gain, the cost saved). Hence, the matcher only can maximize the expected *number* of successful agreements but not the expected *total benefit* resulting from the successful agreements (expected gain or cost savings).

11.4.3 Preferential Grouping

1. See also section 11.3.2.4
2. This is an open problem : see 15.1/4
3. It is not trivial to solve the *preferential grouping problem* using the technique described in section 11.4.2 because *preferential grouping* depends on characteristics of *pending* (not yet agreed) proposals also; for those, there is no feedback available to feed the regression. It is unclear which quantity shall be used as independent variable to feed the *logit* predictor.
4. Lack of *preferential grouping* is expected not to cause negotiations to fail but rather to cause proposals to be ignored : see time-out mechanism in 11.3.1/5. Those phenomena are different and can be distinguished by the *GCPMS* and if an *MNL* model were used instead of a the *logit* model but for the matching only the success rate is relevant not the accurate prediction of the fail mode.

11.5 *Global* Exploration and Matching - General Set Partitioner

11.5.1 Objectives

Objectives are listed in order of decreasing relevance (weight).

1. Equivalent main objectives
   (a) Matching shall minimize the amount of car-kilometers driven (or equivalently: minimize the cost) over the considered base period.
   (b) Matching shall maximize the car occupation level (equivalent to minimize the car-kilometers driven since the amount of person-kilometers is a given constant) over the considered base period.
2. Secondary comfort related objectives
   (a) Matching shall minimize the number of people an individual needs to cooperate with for carPooling during the base period (one week). Note that
      i. an individual can cooperate in several pools.
      ii. the requirement minimizes the number of individuals and not the number of groups within which to cooperate; this is because if i cooperates in *pools* p_0 and p_1 consisting of groups g_0 and g_1 respectively and such that $g_0 \subset g_1$, then the groups differ but the set of cooperators is minimal. From an organisational point of view this kind of difference between *groups* is not harmful.
   (b) Matching shall minimize the number of changes in weekly plans for each individual (i.o.w. each individual prefers to execute the same travel plan every week (in order to keep organisational burden minimal)).

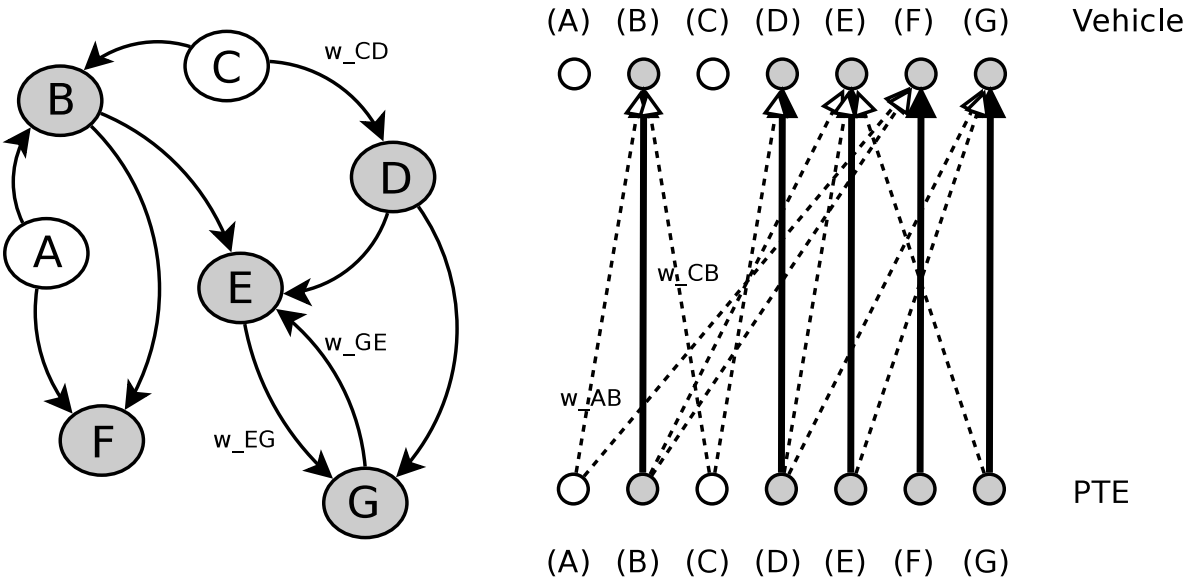

Figure 11.4: The leftmost diagram shows the graph where vertices correspond to PTE (*periodicTripEx*) and the edges are labeled with the success probability for the negotiation (if that is sufficiently large). The rightmost part shows the same information in bipartite graph showing both *PTE* and *vehicles* . Continuous line arcs connect a PTE to the vehicle of its owner; dashed lines show potential participation as a passenger. Grey vertices correspond to PTE where the owner is prepared to drive. Some, but not all of the edge edges have been labeled with their weights.

11.5.2 Research Topics - Conceptual and Technical Problems to Solve

1. The final decision to join a *pool* is not taken by the matcher but by the negotiation process between a pool and a candidate applying to join. How does this influence the matching process ? What kind of behavioral warranties (invariants) are required on behalf of the negotiation process in order to make the matching process effective ? The negotiation process needs to be compliant with the goals/objectives of the *matcher*, otherwise their cooperation will fail.

2. The study of the matcher's efficiency is a research topic in this project.

11.6 Advice determination - Assignment problem

11.6.1 Graphs

1. Consider a graph $G(V, E)$ where each vertex $v \in V$ corresponds to a *periodicTripEx* (periodic trip execution) and where an edge joins two vertices if and only if the success probability for the negotiation to perform two *periodicTripEx* by carpooling, is sufficiently high.

2. G is a directed graph since the success probability for the negotiation depends on who is the driver. Indeed, both *path similarity* and *time interval similarity* are asymmetric (see definition 8.2.7 and section 11.3.2.2).

3. The solution for the general case where *periodicTripEx* combination is limited by the car capacity (as opposed to binary matching) can be viewed as a problem of assigning *periodicTripEx*s to cars under following constraints:

   (a) the car capacity shall not be exceeded

   (b) the car shall be driven by its owner

   Figure 11.4 shows a sample problem.

   (a) The left hand side of the figure shows the *weighted digraph* discussed before. Each vertex corresponds to a *periodicTripEx*. Grey colored vertices correspond to *periodicTripEx* for

which the owner is prepared (able) to drive; white colored vertices are owned by people who neccessarily need to be a passenger. Edges point to the *periodicTripEx* that will supply the driver (car). Note that the w_{EG} and W_{GE} can differ because of the path similarity. Also note that in the sample shown, many pairs only have a single link between them (probably also due to the asymmetry in path similarity).

(b) The right hand side shows the corresponding bipartite graph. The set of vertices at the bottom is the set of vertices shown in the lefthand side graph. The vertices at the top of the diagram represent the vehicles. There is a continuous line arrow between a *PTE* and a *vehicle* if and only if the *PTE*-owner is prepared to drive (hence for each grey-colored vertex in the lefthand side graph). A *PTE* is linked to a *vehicle* by a dash-line arrow if and only if an edge links the corresponding vertices in the lefthand side graph.

11.6.2 Optimal Assignment problem

The optimisation problem can easily be derived from the rightmost graph in figure 11.4 as follows. Let $G_B(V_B, E_B)$ with vertex set $V_B = V_p \cup V_v$ and edge set E_B denote the bipartite graph of figure 11.4. V_v denotes the set of vehicle vertices, V_p denotes the set of *PTE* vertices. A variable x_e is associated with each edge e. Value *one* (*zero*) means that the edge has (not) been selected. $P(e)$ denotes the *PTE* (source vertex) for the edge; $Veh(e)$ denotes the vehicle (target vertex). $cap(v)$ denotes the vehicle capacity (defined as the number seats, excluding the driver seat). Let $Veh(p)$ denote the vehicle owned by the owner of the *PTE* p. The weight associated to each edge that links a *PTE* to the owners vehicle, is set to zero because those links do not contribute to the objective of maximizing the number of succeeded negotiations: $\forall e : (Veh(e) = Veh(P(e))) \Rightarrow (w_e = 0)$. The problem to be solved is:

$$maximize \sum_{e \in E} w_e \cdot x_e \tag{11.18}$$

subject to

$$\forall e \in E : x_e \in \{0, 1\} \tag{11.19}$$

$$\forall p \in V_p : \sum_{\{e \in E | P(e) = p\}} x_e = 1 \tag{11.20}$$

$$\forall v \in V_v : \sum_{\{e \in E | (V(e) = v) \wedge (Veh(P(e)) \neq v)\}} x_e \leq cap(v) \tag{11.21}$$

$$\forall v \in V_v : \forall e, f \in E | (e \neq f) \wedge (V(e) = V(f) = v = Veh(P(f))) : x_e \leq x_f \tag{11.22}$$

Equation 11.19 limits the range of the (boolean) variables. Equation 11.20 requires that each *PTE* shall be assigned to exactly one vehicle (i.e. the trip shall be executed). Equation 11.21 states the limited capacity for each vehicle. Equation 11.22 follows from the requirement that each car can be driven by its owner only.

$$\forall v \in V : (\exists e \in E | (Veh(e) = v) \wedge (x_e = 1)) \Rightarrow (\exists f \in E | (Veh(P(f)) = v) \wedge (x_f = 1)) \tag{11.23}$$

Equation 11.23 says that if there is a passenger in the car who is not the car owner, then the car owner is in the car too.

11.6.3 Optimisation problem

1. By numbering both the PTE and vehicles from 0 to $N - 1$, the edges can be identified by x_{p,j_v} where i identifies a *PTE* and j identifies a vehicle. The problem from section 11.6.2 then can rewritten as follows:

$$maximize \sum_{i_p \in [0, N-1], j_v \in [0, N-1]} w_{i_p, j_v} \cdot x_{i_p, j_v} \tag{11.24}$$

subject to

$$\forall i_p, j_v \in [0, N-1] : x_{i_p,j_v} \leq 1 \tag{11.25}$$

$$\forall i_p \in [0, N-1] : \sum_{j_v \in [0,N-1]} x_{i_p,j_v} = 1 \tag{11.26}$$

$$\forall j_v \in [0, N-1] : \sum_{i_p \in [0,N-1]} x_{i_p,j_v} \leq cap(v_{j_v}) \tag{11.27}$$

$$\forall i_p, j_v \in [0, N-1], i_p \neq j_v : x_{i_p,j_v} - x_{j_v,j_v} \leq 0 \tag{11.28}$$

2. This problem can be solved using linear programming i.e. no integer programming is required. THIS CONTAINS AN ERROR: THE ONLY STATEMENT ONE CAN TRY TO PROVE IS THAT THE POLYTOPE HAS $\{0,1\}^N$ VERTICES IF ALL CAPACITY CONSTRAINTS ARE REDUNDANT

   Theorem 11.6.1. *The constraints defined by equations 11.25, 11.26, 11.27 and 11.28 define a solution polytope for which the vertices correspond to vectors whose component values equal either 0 or 1.*

   Proof. (a) Replace the inequalities 11.25, 11.26, 11.27, 11.28 by equations to specify the hyperplanes bounding the feasible domain.

   (b) All coefficients in the resulting equations equal either -1, 0 or 1.

   (c) Consider variables x_{i_p,j_v} and x_{i_p,k_v} with $j_v \neq k_v$. There is at most one equation that contains those variables; if such equation exists, it stems from 11.26.

   (d) Consider variables x_{i_p,j_v} and x_{k_p,j_v} with $i_p \neq k_p$. There is at most one equation that contains those variables both having a coefficient equal to 1; if such equation exists, it stems from 11.27.

   (e) Consider variables x_{i_p,j_v} and x_{k_p,j_v} with $i_p \neq k_p$. There is at most one equation that contains both those variables having coefficients with different signs; if such equation exists, it stems from 11.28 and has the form $x_{i_p,j_v} - x_{j_v,j_v} = 0$ where $i_p \neq j_v$.

   (f) If an equation of the form $x_{i_p,j_v} - x_{j_v,j_v} = 0$ where $i_p \neq j_v$ exists, then there is exactly one equation that contains $x_{i_p,j_v} + x_{j_v,j_v}$ (derived from a capacity constraint equation 11.27).

   (g) All right hand side coefficients are non-negative integers.

   (h) Any set of two equations taken from 2c and 2d then contains at least three different variables.

   □

   Corollary 11.6.1. *The* simplex *method (as opposed to* integer programming*) can be used to solve the problem.*

   Proof. Since the optimum value for a linear objective function over a convex domain bounded by linear constraints, occurs at a vertex of the domain, the *simplex method* will find one of the polytope vertices to be the optimum. According to theorem 11.6.1 those vertices have integer coordinates. □

   Note that all coefficients and for the *slack variables* originating from the inequatilies. Since the right hand sides all are integer values and the solution values shall not exceed 1, the vertices of the simplex all have vector components that either equal 0 or 1. Hence the *simplex* method can be used (no specific integer programming techniques required). The solution method in general cannot profit from the special structure of the large sparse matrix.

3. Consider again the rightmost part in figure 11.4. Let $x_{i,j}$ denote an arc linking *PTE* i to vehicle j. The unknows $x_{i,j}$ and $x_{k,j}, k \neq i$ can occur as a term in at most one equation or inequality because they all are terms in the constraint that applies to i. The unknowns $x_{i,j}$ and $x_{i,k}, k \neq j$ occur

as terms also in at most one inequality (the capacity constraint that applies to vehicle j). The variable $x_{i,i}$ occurs in multiple inequalities, ecah time together with exactly one variable $x_{k,i}, k \neq i$: this follows from equations 11.22 and 11.28 (which are equivalent. Since the initial coefficients in the matrix all $\in \{-1, 0, 1\}$ elimination of an unknown from a row r_0 while inverting the matrix in the simplex method, either adds a row r_1 or subtracts it. Taking into account the pairing of the terms mentioned above, the resulting coefficients again $\in \{-1, 0, 1\}$. It is improbable that standard solvers can benefit from this knowledge. On the other hand, the eliminating variables (bringing variables *into the base* in Simplex terminology) in general makes the matrix less sparse.

11.6.4 Reduction to matching problem

Part IV

Implementation Issues

Chapter 12

The Use of Janus

Chapter 13

Grid Computing

Part V

Running Simulations

Chapter 14

Data

Chapter 15

Scenarios

Part VI

Open Questions

15.1 Questions on Functions and Mechanisms

1. Can it be helpful to keep for each individual, details about preceding negotiations (if possible keeping the reason for failure also) ?

2. Is it useful to keep track of the history of an individual (i.e. to keep a list of former collaborations) ?

3. Does it make sense to keep track of the number of succeeded, failed and ignored (timed-out) negotiation proposals for each individual ? Can this help to increase predictability ? How do those quantities to be initialised ? What is the minimum number of proposals processed by an individual in order to become confident about those values ?

4. Solution for *preferential grouping* problem is to be found: does *preferential grouping* make negotiations mutually dependent and hence make the use of the *logit* model invalid ?

Bibliography

[1] Sungjin Cho, *Carpooling terminology*, IMOB internal report, 2011.

[2] Sungjin Cho, Ansar Yasar, and Luk Knapen, *CarPoolingAgent-based carpooling design (Part-A)*, IMOB internal report, 2011.

[3] ———, *CarPoolingAgent-based carpooling design (Part-B)*, IMOB internal report, 2011.

[4] Hon Wai Chun and Rebecca Y.M. Wong, *N* - an agent-based negotiation algorithm for dynamic scheduling and rescheduling*, Advanced Engineering Informatics (2003), no. 17, 1–22.

[5] Giorgio Gallo, Sang Nguyen, Giustino Longo, and Stefano Pallottino, *Directed hypergraphs and applications*, Tech. Report 89.00208.12, National Research Council of Italy, under Grant n., 1992.

[6] Luk Knapen, Daniel Keren, Ansar Yasar, Sungjin Cho, Tom Bellemans, Davy Janssens, and Geert Wets, *Analysis of the co-routing problem in agent-based carpooling simulation*, Procedia Computer Science (Niagara Falls), Procedia Computer Science, Elsevier, 2012.

[7] Michael Spivey, *A generalized recurrence for bell numbers*, Journal of Integer Sequences **11** (2008), 1–3.