TERM PROJECT

Generating Day Plans From Origin-Destination Matrices

Marcel Rieser
August 2004
ETH Zurich, Switzerland

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abstract

Traffic demand modeling was based for a long time on demand matrices, also called origin-destination-matrices (OD-matrices). These matrices were used by various traditional assignment models like VISUM [13] or EMME/2 [4]. But more and more microsimulation tools are becoming a major role in traffic demand modeling, because they offer several advantages over the assignment models.

Because microsimulation models include decision making processes, some knowledge about the individual's plans has to be given. This includes also information about activity patterns, which are lacking in OD-matrices. Nevertheless, OD-matrices contain many useful information, particularly because they are often immediatly available. The question comes up how OD-matrices could be used to regenerated plans, which is discussed here.

First approaches are shown by comparing VISUM—a traditional assignment model—with MATSIM [8]—a microsimulation model—in a case study of the Zurich Area of Switzerland.

Contents

1 Introduction

Microsimulation is becoming more and more important in traffic simulation, traffic analysis and traffic forecast (see [14]). Some advantages over conventional assignment models are:

- Computational savings in the calculation and storage of large multidimensional probability arrays.
- Larger range of output options, from overall statistics to precise information about each specific synthetic traveler in the simulation.
- Explicit modeling of the decision making processes of the individuals.

The last point is important since it is not a vehicle which produces traffic, it is the person who drives it. Persons do not just produce traffic, instead each of them tries to execute his day (week, life) in a profitable way. They go to work to gain money, they go hiking for their health and pleasure, they visit their relatives for pleasure or because they feel obliged to do so, they shop to cook a nice dinner at home, and so on. Since not all of this can be done at the same location they have to travel, which produces the traffic. To plan a day efficiently, a lot of decisions are made by each person:

- Which route should I take to get to work? → Route choice decision
- Which mode should I use to go to the lake? → Mode choice decision
- Should I drink another beer before going home? → Activity duration choice decision
- Should I go shopping near my home or at the mall? → Location choice decision
- When should I do sports today? → Activity starting time choice decision
- Whom should I take along? → Group composition decision
- Should I go swimming before or after work? → Activity chain choice decision

There are many more decisions to make, some of them are planned hours (days, months) in advance and others are made as a spontaneous reaction to specific situations. Many decisions induce other decisions. For example, if I am late for work, I am supposed to work longer, so there's no time left to go shopping today, so I need some time tomorrow to do the shopping. This example shows the importance of describing plans for each individual in a simulation model, because it is the plan and the decisions made by the person who adhere to this plan that produces the traffic.

To simulate a typical day in an urban area, microsimulation tools need precise information about the plans of each individual and also some knowledge about people's decision making process. But to extract daily plans (activity chains) from persons is not a trivial task. On the other hand conventional assignment models like VISUM [13] or EMME/2 [4] typically use OD-matrices which hold the information about the trips from an origin to a destination (usually zones or regions), but these do not provide links back to individuals. This leads to the main question of this paper:

> Is it possible to impute (reconstruct) people's activity chains from a set of origin-destination matrices (OD-matrices)?

This paper demonstrates first approaches to answer this question. It is organized as follows: first, the case study to which the approaches are applied is discussed including the description of the OD-matrices and how they have been generated. This is followed by a detailed description of the "activity chain generation process". To assess the quality of the results we compare VISUM and MATSIM [8] results employing OD-matrices and daily activity chains respectively. Finally, concluding remarks and recommendations for future research are outlined.

2 Overview

2.1 Terminology

The area under investigation is divided into different *zones*. A zone is an area containing a number of inhabitants, a number of work places and other characteristics. Usually, a zone corresponds to a city, a village or a municipality.

Origin-destination matrices (OD-matrices) show the number of trips from every zone in the area under investigation to all the other zones. The matrices are also called demand matrices, based on the meaning of the data. In this paper, *OD-matrices* is used because of its more general meaning.

If people are on the road, they usually travel from one activity to another activity. That means they usually follow an *activity-chain*. Activity chains, also called *activity patterns*, list all activities of one person in chronological order. An activity chain starts and ends usually with an activity referred as "being home" or just "home". A typical pattern is `home-work-leisure-work-home`, or simply `h-w-l-w-h`.

Between two activities, one or more *legs* have to be defined. A leg describes which transportation mode is used to get from one activity to the next one. It is possible, that the transportation mode changes (e.g. from car to train when using 'Park&Ride'). In those cases, more than one leg can be stored between two activities.

When a person travels from one place to another (optionally with one or more stops in between), they often speak from a *journey*. A journey can either be a round-trip or only one-way, e.g. going on holiday. But before going on a journey, people usually make a *plan* where to travel to. A plan is an example of a activity chain, consisting of a start location, an end location (both with an activity assigned) and several stations along the journey. The difference between a plan and a journey is that the latter is fixed, while the plan only shows the intention a traveller has. The journey based on a plan differs in most cases from the plan as unforeseen events may happen during the execution of the plan.

This case study focuses on *day plans*—plans for one day (plans can usually be of any duration of time from a few minutes up to several years). But it must be noted that the presented methods could work with any kind of activity-chains, and thus with any kind of plans.

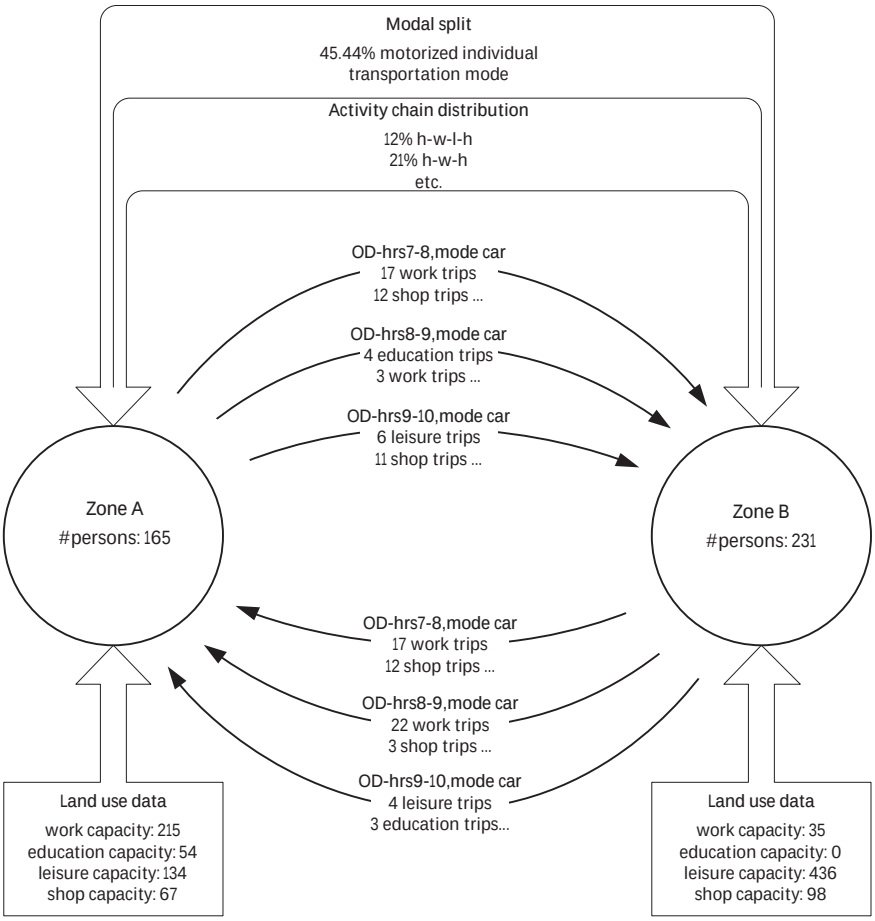

Figure 1: Summary of the used input data

2.2 Input Data

In order to generate day plans out of OD-matrices, additional data is required. The geographical location of zones is needed as well as demographical information and behavioral data of the population. Figure 1 shows a very simple example of required (and available) data.

All input data is stored in multiple text files. The format of the data is either compatible with VISEM [13] or consist simply of space-delimitted values. Appendix A lists examples for all files mentioned in the following chapters.

The data used as input were either calculated using VISEM [13] as described in [12], or were compiled to use as input for VISEM in [12]. The results of VISEM can be used by VISUM [13] to do a static assignment of the traffic in the OD-matrices. As the input for MATSIM [8] is essentially based on the input or output of VISEM, meaningful comparisons between MATSIM and VISUM

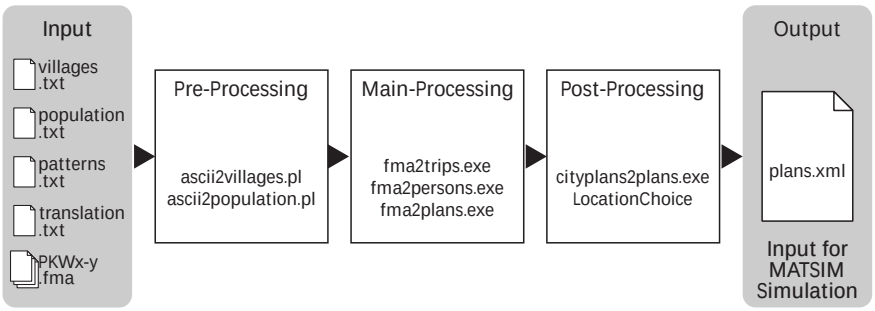

Figure 2: Overview of the involved processes

are possible.

2.3 Processes

The above mentioned input files are processed by various scripts and programs. Some processes can be run sequentially to batch-process multiple input files.

In a first stage the various plain text files are converted into XML files. After this pre-processing, day plans are generated by one out of tree different programs: *fma2trips*, *fma2persons* or *fma2plans*. These programs and their differences will be discussed later. In a third stage, the generated plans are post-processed by *cityplans2plans* and, depending on which main process was used, with *LocationChoice* to validate against *plans.dtd* from MATSIM [8].

Figure 2 shows an overview of the processes discussed in this paper. The detailed data flow, including intermediate data files, will be shown later for each main process.

The main processes *fma2trips*, *fma2persons* and *fma2plans* as well as *cityplans2plans* share a common configuration file *config.xml*. This file contains the paths of input- and output-files and has the same format as the MATSIM-configuration file. While these file paths could also be passed as arguments, collecting them in a configuration file helps simplifying running different scenarios.

It has to be said that each run produces only persons of a specific transportation mode (car, bike, public transport, ...). The mode is also set in *config.xml*.

Additionally, a file *settings.xml* exists which contains lookup values used by more than one process.

2.4 Output Data

The resulting day plans have to validate against a document type definition, *plans.dtd* (see Appendix A.10) from MATSIM [8]. Each plan needs to be assigned to a person with a unique ID and consist of a series of activities. For

each activity x- and y-coordinates must be provided, as well as a time information (either duration or start- / end-time), which should be given in the format "hh:mm". To travel between the locations, a `leg` mode has to be set (see Appendix A.9 for an example of a valid plan file).

3 Preprocessing

3.1 File Formats

VISEM [13] has its own ASCII-based file format, whereas MATSIM [8] uses XML-files for storing information. Thus, data has to be converted from ASCII format into MATSIM XML-files. This preprocessing of the ASCII-files is done by two Perl scripts. Perl was chosen because it easily parses complex lines of text, containing different values separated by blanks. Perl is a standard on Unix, Linux and Mac OS X; for Windows, ActivePerl [1] is freely available.

3.2 ascii2villages.pl

```
usage:
 perl ascii2villages.pl input-file output-file
```

This script converts a plain text file containing information about zones into a corresponding XML file. The *input-file* (usually *villages.txt*, see Appendix A.1) is expected to contain the village-id (integer value), the village-name (string value), and the x- and y-coordinate of the village-center, both integer values. Optionally, a fifth parameter can be listed, containing the radius of the populated area (also an integer value). All the values should be separated by tabulators. The village-name can contain digits, spaces or other special characters.

Such data can be generated by most GIS-applications [5] for the area of interest.

The output, an XML representation of the given data (see Appendix A.5), is written into the file specified in the second argument, *output-file*. This file is only used by *cityplans2plans*, when it is started with the option `-cityid2xy`.

3.3 ascii2population.pl

```
usage:
 perl ascii2population.pl IN_villages IN_popuplation
        IN_patterns [IN_translation] IN_% OUT_population.xml
```

This script writes out the number of people living in each zone, split up into different activity patterns. The given distribution of the activity chains is used to assign a chain to the persons. Notice that minor rounding errors of the number of people can occur because of distributing the population to the zones, reducing the population according the given modal split and assigning the activity chains.

The argumgents have the following meaning:

IN_villages: A plain ASCII file containing zone-id, zone-name, x-coordinate and y-coordinate, separated with tabs, in this order (*villages.txt*, Appendix A.1). This is the same input file as for *ascii2villages.pl*.

IN_population: Contains a matrix with the zones in rows and different population groups in columns. The values in the matrix contain the number of people of a distinct population group within a zone. Row and column headers display zone-ids and the population group names, respectivly. The very first cell (intersection of header row and header column) needs to be empty (*population.txt*, Appendix A.2).

VISEM [13] uses the same format for input of zones and residents. As MATSIM [8] does not use population groups, only the total count of people per zone will be used further.

IN_patterns: A list containing activity patterns in the first row and their global frequencies in percents (0 – 100) in the second row. Activity patterns consist of multiple characters in a row, each character depicting an activity (*patterns.txt*, Appendix A.3).

IN_translation: (optional) Depending on the origin of the input data, the name of activity patterns must be translated to match the terms used by MATSIM [8]. To automate this translation, provide a file containing two rows of characters. The first row must contain the shortcuts (one character) of activities in the original language, while the second row should contain the corresponding characters in the new language (see *translation.txt*, Appendix A.4 for an example). A specific character can not be used in both lists – the lists must exist of completly different charsets. Capital and small letters are distinguished as different characters.

IN_%: Modal Split to use in percents (0 – 100). This value is used to generate only the given percentage of persons listed in the file *IN_population*. Usually, the whole population is listed in the *IN_population* file. But it is very unlikely that all people use the same transportation mode at a time, therefore this percentage allows an easy setup to only generate that part of a population, which indeed uses the simulated transportation mode.

If reduction of the population by the same percentage in every zone is unwanted, it is recommendable to adjust the number of people in the file *IN_population* and to use `100` as parameter (see chapter 6.2, scenarios, for a detailed discussion on how to use this parameter).

OUT_population.xml: XML-output of the script given at the beginning, containing a list of zones (see *population.xml*, Appendix A.6). For each zone, every possible activity pattern and the number of persons behaving along this pattern in a zone are listed. Every pattern will be used at least once in each zone.

At the beginning, *OUT_population.xml* contains the so called *start population*. When new people are created, the numbers of people in *OUT_population.xml* is decreased. This means that the file contains at every time only those people which have not yet left their home, why it is also referred to as containing the *home population*.

The script ensures that in every zone, every activity patterns is used at least once. Additionally, all fractional values are rounded using the operator `ceil()`. This leads to slightly more persons generated than listed in *IN_population*. But because the difference is very small and for other reasons shown later, it has no real influence on the results (see chapter 6).

4 Generating Day Plans

4.1 Introduction to Day Plans

Day plans, for simplicity usually referred as *plans*, consist of detailed information about the daily activities of a person. Activity patterns are used as a base for schedules. Activity patterns list all activities of one person in chronological order for one day, for which they are also called "activity chains". An activity chain starts and ends usually with an activity referred as "being home" or just "home".

In an activity chain single activities are classified as primary or secondary activities. Usually, the activity with the longest duration is selected as the primary activity, while all other activities are graded secondary. This implies, that there is only one primary activity in most cases. If two or more activities have the same, maximum duration, the first is chosen as primary. The distinction between primary and secondary activities is needed for the MATSIM-module *LocationChoice*, which depends on this information (see chapter 5.2).

Following the rule of duration `work` is a typical primary activity. If work is not part of a pattern, `education/e`, `leisure/l` or `shop/s` can take the position of primary activity.

In the example `h-w-l-w-h`, the first occurence of `work` would be chosen as primary activity, whereas in the pattern "`h-l-s-h`" the second activity, `leisure`, would be designated as primary (assuming `leisure` has a longer duration than `shop`).

Each activity in a day plan has different attributes containing additional information about the activity, such as the location where an activity takes places, as well as duration or other temporal information (e.g. ending time). MATSIM uses the format "hh:mm" or even "hh:mm:ss" to store times. While time indications in the "hh:mm:ss" form are very human-readable, the format is not very well suited for electronic processing. Thus, the main processes described below use a simple integer value for storing durations or times, which opens up the possibility to use simple mathematical operators to modify times. The stored values represent the minutes since midnight and are converted into the standard time format in a post-processing step (see *cityplans2plans*, 5.1).

MATSIM [8] can use plans as a base for its simulations.

4.2 fma2trips

```
usage:
 ./fma2trips.exe config.xml data.fma
```

For every trip in the OD-matrix *data.fma* a new person with exactly the one trip from the OD-matrix is generated. So, basically this program only creates *trips*, but not real day plans where people end their days at the place they started it.

In the configuration file, the following entries are needed:

leg_mode: A string containing the mode to use for the `leg`-tags between activities, usually 'car'.

output_plans: Path of a file to write out the newly generated plans. The file will not be overwritten, but new plans will be appended to it. Thus,

the file misses the ending `</plans>`-tag, so that other plans can be appended in a later iteration.

inout_counter: Because *fma2trips* always creates new day plans, it never reads in any plans-file. To avoid problems with the uniqueness of person-ids, *fma2trips* stores the highest used person-id in the file specified in *inout_counter*, and reads this file to initialize the person-id to use upon startup. If the file is not found or cannot be read, *fma2trips* starts with a person-id of `1`.

The application, written in C++, assigns both generated activities (the one before and after the trip) the type `home/h`. This proved to be the most stable way for MATSIM [8] even if the second activity has place in another zone than the first activity and cannot really be a `home`-activity. This behaviour was chosen to prevent conflicts in MATSIM [8]. For the first activity, the attribute `end_time` is set with a random value between the start-time and the end-time of the read-in OD-matrix. The time information is stored as the number of minutes since midnight. The second activity, which is also the last activity for every plan, contains no time information. Figure 3 shows the data flow including intermediate files. Note that not all input files are used as *fma2trips* uses a very simple algorithm.

4.3 fma2persons

```
usage:
 ./fma2persons.exe config.xml data.fma
```

This process generates for each trip stored in the OD-matrix *data.fma* a new person—as long as people are available in a zone. From the data in the OD-matrix, the home-location and the location of the primary activity are assigned to the new person, but not the locations of secondary activities. Latter are set by *LocationChoice* in a post-processing step (see 5.2).

In the configuration file, the following entries are needed:

leg_mode: A string containing the mode to use for the `leg`-tags between activities, usually 'car'.

input_population: Path of a file containing information about the starting population. In the first iteration, this is the file generated from *ascii2population.pl* (*population.xml*, Appendix A.6). In following iterations *output_population* should be used, as this file contains only the count of those people which have not been assigned a plan yet.

input_plans: Path of a file containing day plans. If this file exists, the plans in this file are copied to *output_plans* and newly generated plans will be added to this file with continued ids. If not, newly generated plans will start with an id of `1`.

input_settings: Path to the *settings.xml* file, which contains the default durations of activities as well as the definition of the primary activity for each activity pattern.

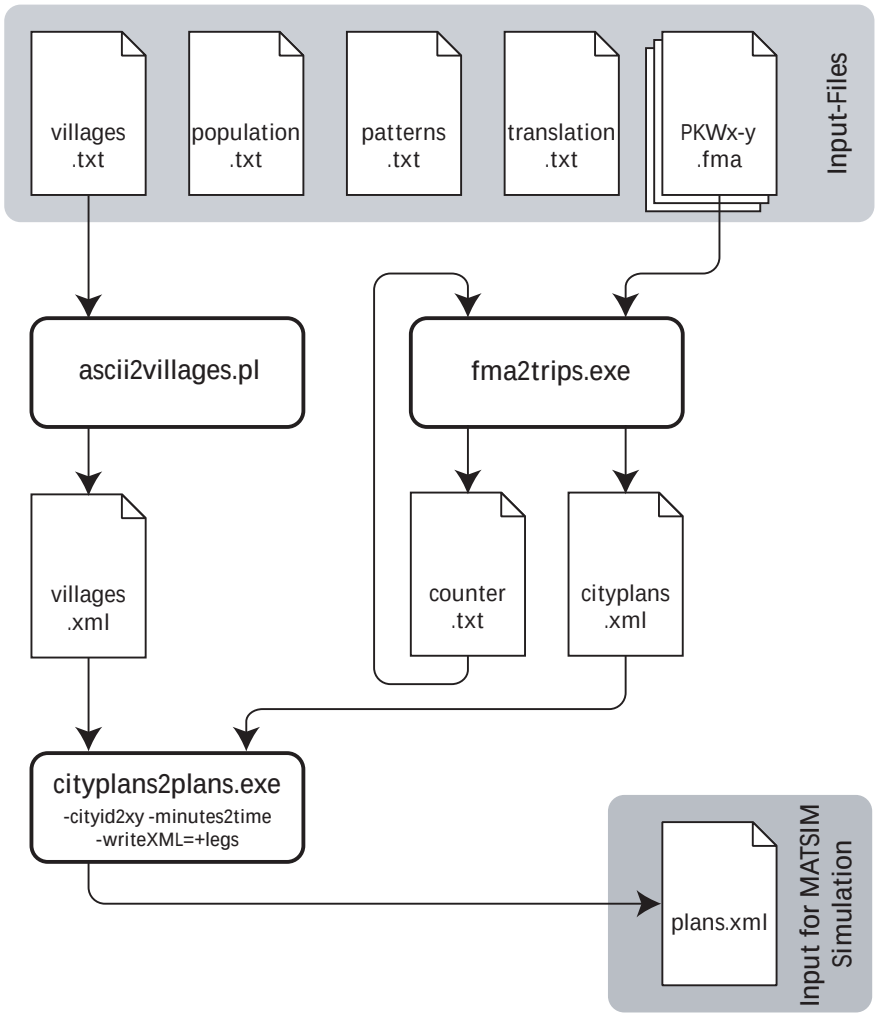

Figure 3: Data flow when using *fma2trips*

output_population: Path of a file to list the population at the end of one iteration. This file should be used as *input_population* for the next iteration.

output_plans: Path of a file to store the newly generated plans, including any previous plans from *input_plans*. This file should be used as the input for the next iteration, or as input to *cityplans2plans* when called with the option `-cityid2xy` (see 5.1).

The application, also written in C++, reads one line after the other from the OD-matrix. Every cell represents the number of persons who travel from one zone to another zone). The first zone is interpreted as home-city, while the destination is interpreted as the location of the primary activity. The algorithm randomly picks this number of persons from the population in the home city, decreasing the count of people in the population list. Every chosen person already has an activity pattern assigned from the preprocessing stage.

Every chosen person and its plan is written into *output_plans*. When writing, each activity gets a duration assigned. The default duration for each activity type is set in the settings file (see Appendix A.14). If an activity type occurs more than once in an activity pattern, the duration for each single activity is divided by the number of occurences of the activity type. This way, the sum of durations of all the activities of one type in the activity pattern matches the default duration in the settings file.

Even though those assumptions are sensible, sometimes the durations can be quite wrong. For example, the duration of the shopping activity in the activity chain 'home-shop-home' could be just some minutes (i.e. buying a bread at the bakery next door) or it could also be about 10 hours (i.e. a shopping day at the mall). The same question applies for the leisure activities. On the other hand, the average duration of 'work' and 'education' usually has less variance. Nevertheless, for lack of more detailed assumptions the generation process will be employing these assumptions for the time being.

As the count of available people is decreased while reading one OD-matrix after the other, it can happen that there are no more people available in a zone. In this case, no new plans are generated anymore. It is assumed, that the missing rides are generated by persons that started earlier in another zone to which they have to return. As these rides are already listed as part of other activity chains, we do not lose much information by ignoring them (see chapter 6 for details).

The data flow including intermediate file is shown in figure 4.

It must be noted that it is almost impossible to reach a perfect match, depending on the used OD-matrices. As an example, the output matrix for a whole day generated by VISEM [13] as explained in [12] prevents a perfect match: for many zones the number of trip starts differs from the number of trip ends—making it impossible for persons to be at home at the end of a day. As MATSIM-plans are all round-trips, the number of trip starts and trip ends will be the same for each zone.

4.4 fma2plans

```
usage:
 ./fma2plans.exe config.xml data.fma
```

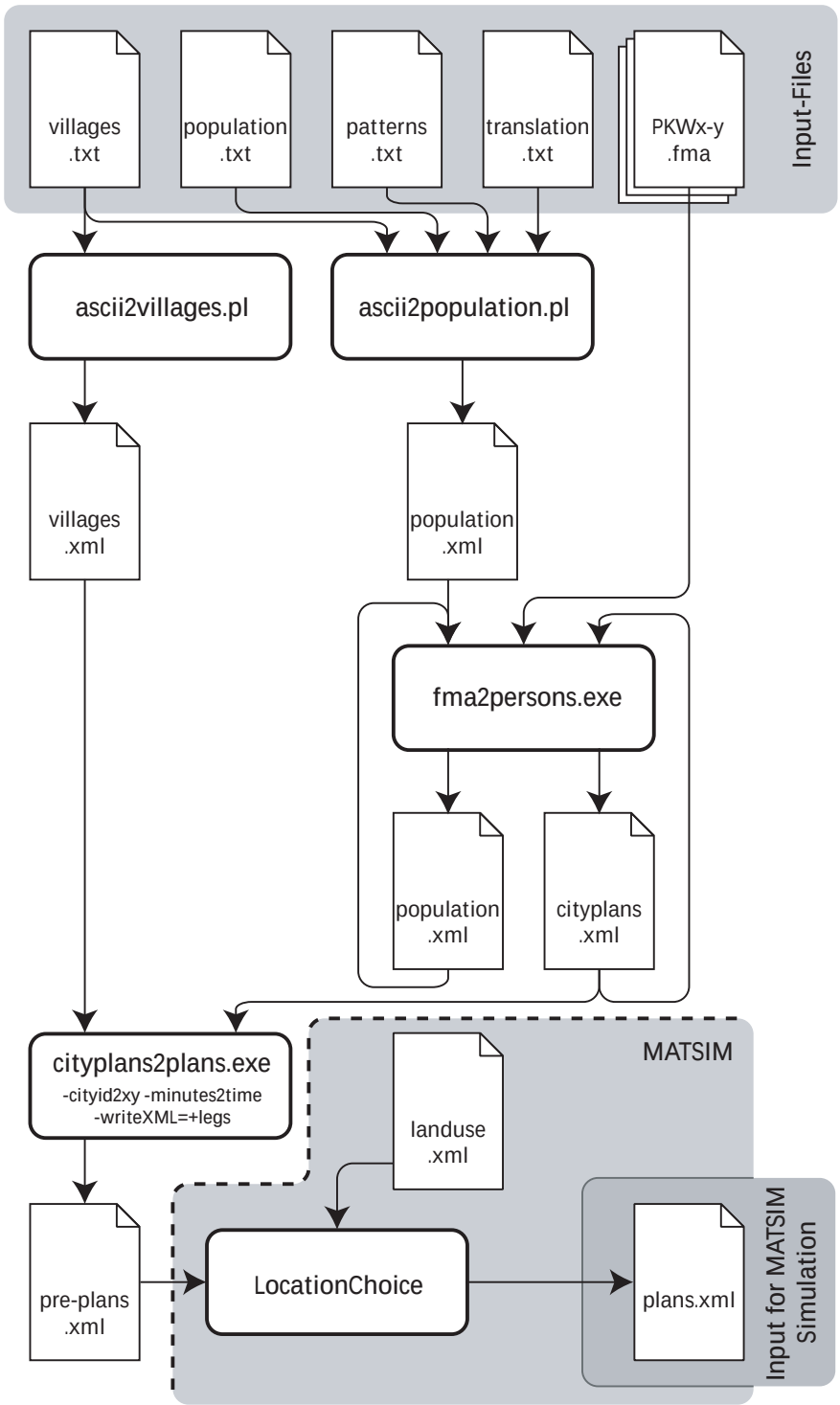

Figure 4: Data flow when using *fma2persons*

Both *fma2trips* and *fma2persons* have some drawbacks. The first one does not generate real day plans, while the latter in combination with *LocationChoice* (see 5.2) does not reproduce the trips in the OD-matrices. *fma2plans* tries to generate complete day plans with as many trips as possible from the OD-matrix.

At a first glance, the algorithm works similar as *fma2persons*: When generating a new person, the person is assigned an activity pattern. But while *fma2persons* assigns and writes out all activities of the pattern at creation time, *fma2trips* only generates the first activity (home) and the second activity, both located according to the trip from the OD-matrix. When the process is started again with another OD-matrix, the existing plans are read in. Every person whose plan is not yet complete is placed as *visitor* in the zone of its last activity. When new trips have to be generated, either such a visitor or a new person may be used for the trip.

In the configuration file, the following entries are needed:

leg_mode: A string containing the mode to use for the `leg`-tags between activities, usually 'car'.

input_population: Path of a file containing information about the starting population. In the first iteration, this is the file generated from *ascii2population.pl* (*population.xml*, Appendix A.6). In following iterations, the *output_population* should be used, as this file contains only the count of those people, which have not yet been assigned a plan.

input_plans: Path of a file containing day plans. If this file exists, the existing plans will be copied to *output_plans* and newly generated plans will be appended to this file with continued ids. If not, newly generated plans will start with an id of `1`.

input_settings: Path to the *settings.xml* file, which contains the default durations of activities.

output_population: Path of a file to write out the population at the end of one iteration. This file should be used as *input_population* for the next iteration.

output_plans: Path of a file to write out the newly generated plans, including previous plans from *input_plans*. This file should be used as the input for the next iteration, or as input to *cityplans2plans*.

output_complete_plans: Complete plans will be written to the file specified in *output_complete_plans*. This helps speeding up the process as those plans do not have to be read in again in later iterations.

Figure 5 shows the data flow including the mentioned files.

After all available OD-matrices have been processed, it may happen that there are still plans which have not yet been completed. Those plans can be completed in two ways. One possibility is to just add a home-activity at the end of the plans, ensuring all people are at home at the end of their plans, leaving out any other, not yet located and scheduled activity. A second way to complete the plans is to add all missing activities and marking all already assigned activities as primary activities. This allows to assign the locations to the missing activities using *LocationChoice* (see 5.2).

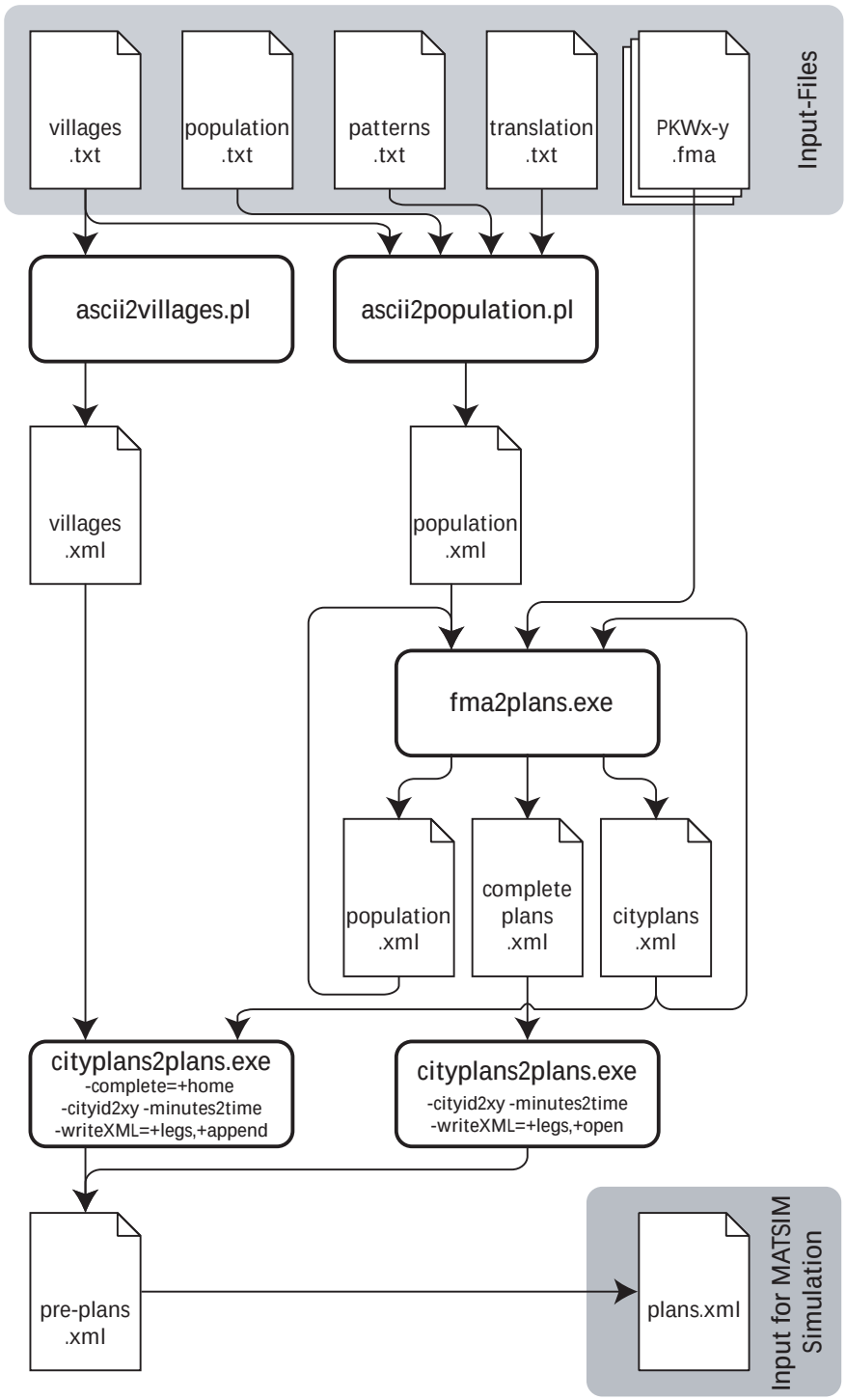

Figure 5: Data flow when using *fma2plans*

4.4.1 Choosing persons for trips

When a new trip from zone A to zone B needs to be generated, *fma2plans* has to decide whether it creates a new person, or if it should try to add the trip to a *visitor* in zone A. One way to do so is shown in algorithm 1.

Algorithm 1: Choosing a person for a trip from zone A to B

```
plan := find visitor in zone A who can travel to B;
if no plan found then
    plan := generate new person from zone A, if possible;
else
    if last activity of plan ends after end-time of OD-matrix then
        if home population of zone A > 0 then
            plan := generate new person from A;
            add first home activity to the new plan;
        end
    end
end
if plan = NULL then
    /* the trip cannot be generated                                    */
else
    add activity to plan;
    set location of activity to zone B;
    remove plan from visitors in zone A;
end
```

The algorithm tries to find a visitor who can travel from A to B. A visitor cannot travel this leg, if he is only missing the last home-activity and B is not its home zone. If no visitor is availabe or the found plan is scheduled to start after the processing hour, the algorithm tries to generate a new person from the home population of zone A. A new person cannot be generated when the whole population from A has already a plan assigned. If either a visitor or a new person is chosen, a new activity according to the persons activity pattern is generated and appended. The location of the new activity is set to zone B. The durations of the generated activities are set the same was as in *fma2persons* (4.3).

Algorithm 1 uses visitors to generate new trips whenever it is possible and feasible. As this penalizes the home population, the algorithm was modified. Algorithm 2 first chooses randomly between the home population and visitors, the probabilities corresponding to the count of people in each group.

To switch between the two algorithms, a compiler flag `FAVOR_VISITORS` has been defined. Set it to `1` to use algorithm 1, and to `0` to use algorithm 2.

4.4.2 Speed Optimizations

Both of the proposed algorithms above search the visitors lists of zones to find matching people for every trip that needs to be generated. This opens a huge potential for optimizations and speed increases.

Caching search results is not feasible, as the lists continuously change. Additionly, the recently found plans will often be used and thus removed from the

Algorithm 2: Choosing a person for a trip from zone A to B without favoring visitors

```
countVisitors := number of visitors who can travel from A to B before the
end-time of the current OD-matrix;
countPopulation := number of people still at home in zone A;
if (countVisitors + countPopulation) > 0 then
    if (random()*(countVisitors + countPopulation)) < countVisitors then
        /* choose a visitor                                          */
        plan := find visitor in zone A who can travel to B;
    else
        /* generate a new person from home population                */
        plan := NULL;
    end
else
    /* there is no one who can to to zone B                          */
    plan := NULL;
end
/* the rest of the algorithm is basically the same as algorithm 1    */
if plan = NULL then
    plan := generate new person from zone A, if possible;
else
    if last activity of plan ends after end-time of OD-matrix then
        if home population of zone a > 0 then
            plan := generate new person from A;
            add first home activity to the new plan;
        end
    end
end
if plan = NULL then
    /* the trip cannot be generated                                  */
else
    add activity to plan;
    set location of activity to zone B;
    remove plan from visitors in zone A;
end
```

list, invalidating the chached information. In order to reduce computational time, shortening the lists as much as possible seems the best way to speed things up, besides keeping the lists sorted which allows to stop a search before the end of the list is reached.

A first step that helps to keep lists short is to insert only those plans into the visitors list which wait for additional activities. This means that complete plans will not be inserted into the visitors-list. Completed plans are even written into another file, so they do not have to be read in and parsed in later iterations.

A second possibilty is not to insert those people who just wait to drive home in the time frame of the current OD-matrix. Instead of inserting them into the visitors list, it is checked if a trip home is available according to the OD-matrix. If this is the case, the home activity is appended to the plan and the count of trips from the current zone to the home location of the person is decremented.

Having a lot of short activity patterns with only three activities, this helps a lot to keep the visitors-list short. But it must be noted that in this case, people waiting for home trips are favored over other visitors having more than one activity open or over people still being at home. A `#define` statement allows to switch easily between the described behavior above or the standard behavior where all incomplete plans are added to the visitors list. Set `FAVOR_HOME_TRIPS` to `1` to sort those plans out which end their journey in the current time frame and keep the visitors list short. Set `FAVOR_HOME_TRIPS` to `0` to treat those plans the same way as all other incomplete plans.

4.4.3 Other Characteristics

When a visitor gets an additional activity assigned, the person is removed from the visitors list of its current location. But it is not added to the visitors list at the new location. This induces that a person can only have one trip within an hour, but not more. As the results (see chapter 6.4) will show, it may happen that in some cases trips cannot be generated because neither any visitor can travel to the desired destination, nor are there any people in the home population left who could start a new day plan. But it could be possible that there is a person coming from another zone who could travel further to the destination the trip provides.

To work around this limitation, *fma2plans* can work through the OD-matrix multiple times. This makes it possible that people can be chosen in a second iteration for trips which could not be generated before. If people are added to the visitors list in the destination zone of the newly generated trip, zones that are processed later already have the new people available unlike zones at the start of an iteration. To prevent this, a parallel update mechanism has to be chosen: Moved people are not added to the visitors list, but are added to another list ("newcomers") inside the zones. After a OD-matrix has been processed and when not all trips could be generated, the remaining visitors are merged with the newcomers. The OD-matrix will then be processed a second time.

5 Postprocessing

The plans file written by one of the three main-processes *fma2trips*, *fma2persons* and *fma2plans* would not yet validate against the document type definition (see *plans.dtd*, A.10). In particular, the following points would cause an error:

- The location of activities are still only set to zones (if at all), whereas MATSIM [8] expects x- and y-coordinates.
- Durations and end-times are stored as minutes since midnight instead of the "hh:mm" time format.
- The plans do not yet contain any `leg`-tags. The leg-mode is only stored as an attribute in the `plan`-tag. Appendix A.7 shows an example of such a file.

In addition to the mentioned points above, the generated plans are not always complete or are even split up in multiple files.

Instead of having multiple small processes, each parsing the plans and doing one conversion, one application was developped which could easily be extended to support additional conversions. This conversion tool named *cityplans2plans* can solve many of the points mentioned above.

To choose locations for activities, which don't even have a zone assigned (e.g. the output plans from *fma2persons*), an existing application *LocationChoice* [7], which is part of MATSIM [8], was modified to support more general plans as it has done before.

5.1 cityplans2plans

```
usage:
 ./cityplans2plans.exe config.xml -module1[=+param1,...] [...]
```

cityplans2plans can perform a variety of different conversions on plans. The different conversions are implemented as modules, while *cityplans2plans* acts like a framework for those modules. This setup allows to simply extend the applications features just by including and recompiling with an additional module.

Each module can be activated using arguments on program start. The argument that activates a module can contain additional, simple parameters, which are passed to the module. Additionly, the modules have the possiblilty to load information from a configuration file.

The following modules are implemented and compiled with *cityplans2plans*, and are used in several combinations when postprocessing the plans files from the three main-processes:

-complete This option loads the module `MakeComplete` which adds missing activities to plans, based on the activity pattern stored in an attribute to plan.

By default, all missing activities are appended with the location set to the home-location. Because this is not useful in most cases, the module can take two parameters:

+home When set, each incomplete plans has only a home-activity added. If other activities are missing, they will be ignored. If this parameter is not set, every missing activity will be added to the plan, with the location set to the home location of the person.

+markPrim If this option is set, existing activities are marked with the `primary="true"`-attribute. It makes most sense if `+home` is not set, as this allows to assign the locations of missing locations using *LocationChoice* (see 5.2).

In most cases, only one of the two parameters is needed.

Because this module adds additional activties, it has to know the default durations of activities. Like *fma2persons* and *fma2plans* it expects those informations in the file *settings.xml*, whose path has to be declared in *config.xml* with the name `complete_settings`.

-cityid2xy This option activates the module `cityid2xy`. For each activity it checks if the attribute `cityid` exists. If it exists, the module chooses a random point within the populated area of the zone and replaces the

attribute `cityid` with the attributes `x100` and `y100`. The populated area is calculated using information from *villages.xml* (see 3.2, *ascii2villages.pl*, and A.5). The path to *villages.xml* is set as parameter in *config.xml* with the name `cityid2xy_villages`.

To choose random points in the zones, the algorithm simplifies the shape of zones into circles. For every zone, the algorithm searches for the nearest neighbor and takes the half of the distance as radius for a circle around the center of a zone, unless the radius of the circle was specified in *villages.txt* or *villages.xml* respectively (see chapter 3.2, *ascii2villages.pl*, for more information). Using this procedure, the zone-areas will not overlap each other. There are even areas, where nobody 'lives'—which makes sense, as zones often have centers where people concentrate, while the population density on the borders of zones is far lower.

The algorithm takes care that whenever a home activity is encountered, the same coordinates will be used as in previous home activities for the same person.

-minutes2time This module converts the time information stored in the attributes `dur` and `end_time` into the "hh:mm" format MATSIM [8] uses. This module should only be called, if all time information are given in minutes since midnight. It does not recognize if a time information is already given as "hh:mm" and tries to interpret this as integer value which will then be converted again into that format—what may lead to undesired results.

-svg When enabling the module `SVGWriter`, a graphic in the SVG-format (see appendix B) is created. The path of the file must be specified in *config.xml* using the parameter name `svg_output`. Additionally, the following parameters must be specified in the configuration file:

svg_minX, svg_minY, svg_maxX, svg_maxY The smallest and largest x- and y-coordinate. These are essentially the same as in the configuration file for *LocationChoice* (see chapter 5.2).

The content of the graphic can be specified with the following parameters to `-svg`:

+homeN Draws a cross on the home location of every N-th person. If N is omitted, 100 will be used.

+legsN Draws all the legs of every N-th person. If N is omitted, 100 will be used.

Figure 6 shows a possible distribution of home locations, generated by the module *cityid2xy*, visualized with the option `-svg=+home20`. It is easy to recognize zones with a high population density by the darker shade of the circles.

-writeXML This option activates the module `XMLWriter` which writes out the processed plans into the file specified in the configuration file with the parameter name `writeXML_output`. Also in *config.xml* must be a parameter with the name `writeXML_DTD`, containing the value which

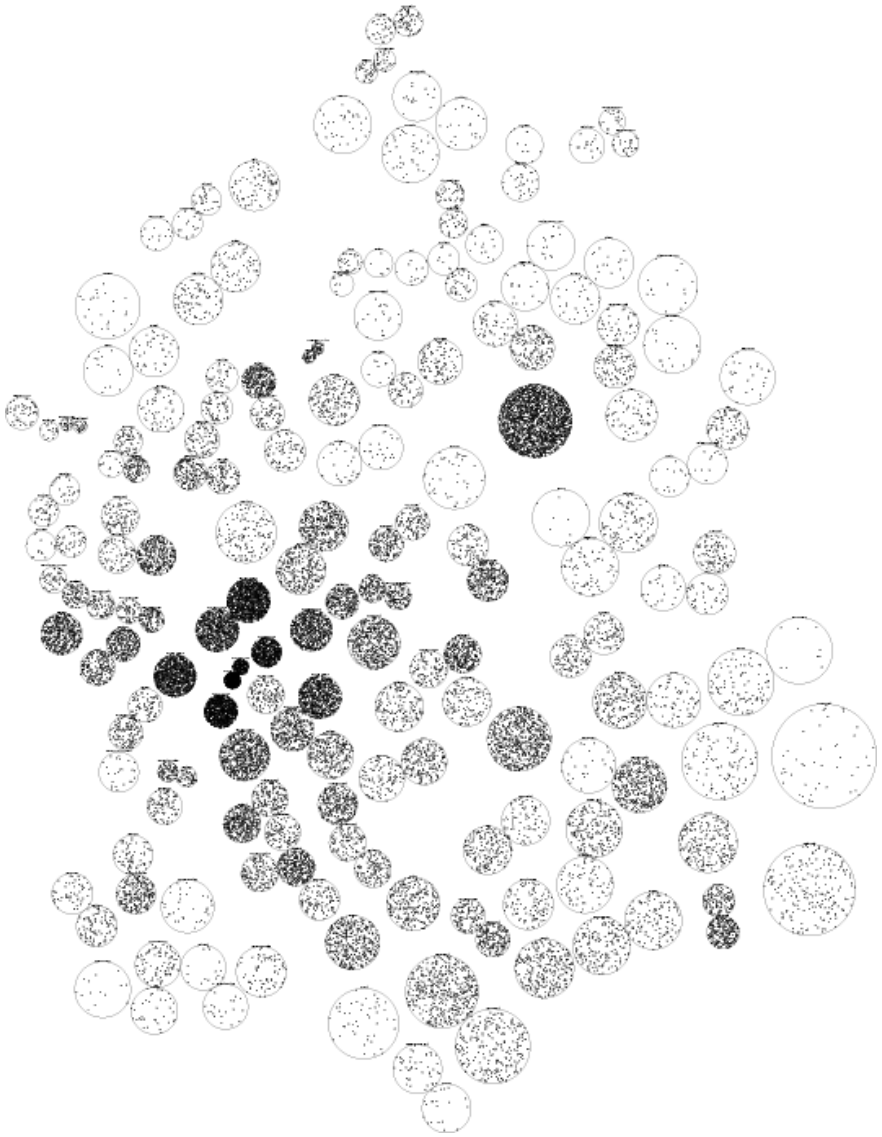

Figure 6: Chosen home locations of every 20th persons of the population in the area of Zurich, Switzerland.
Circles represent the habitable area of zones, each dot representing one person.

gets written as `DOCTYPE` into the output file. This is needed as *LocationChoice* validates the file while parsing and thus needs a valid path to the file *plans.dtd*.

This module can be called with several parameters:

+legs Write legs between activities. If this option is not set, the legmode will only be stored as attribute of the person (see appendix A.7 for an example). If it is set, it is interpreted that a valid plans file has to be created. Thus, the attributes `home_city` and `pattern` will not be written, too.

+append Does not overwrite an existing file, but rather append the converted plans to the existing file.

+open Does not write the trailing `</plans>`-tag. This allows to append plans at a later time.

In order that *cityplans2plans* can be run, it has to know which plans file to convert. The path of the input file must be stored in the configuration file with the parameter name `input_plans`.

5.2 Location Choice for Secondary Activities

5.2.1 Introduction

MATSIM [8] contains a module *LocationChoice* [7], which chooses locations for secondary activities in such a way, that each agent has an optimal day plan. The home-location and the location of the primary activity remain unchanged.

The application, written in Java, uses genetic algorithms to determine the locations for each agent. Each agent (based on the previously created day plans) has a limited knowledge about places in the area of interest and searches within these places for a path, such that his day plan reaches (a local) maximum utility. Additionally, each agent "knows" some other agents and exchanges information about places after every iteration. This way, a *social network* is built that helps the agents to improve the utility of their day plans.

5.2.2 Modifications

The original application [7] assumed, that all plans contain the activity `work`, which was interpreted as primary activity. Additionally, the application expected every day plan to have one or two secondary activities. As the activity patterns and thus the day plans from [12] didn't fulfill these restrictive requirements, the application was modified. In a first step, it was ensured that short plans with no secondary activities did not crash the application. Such day plans will now be ignored for the iterative calculations and simply be written out unchanged at the end. In a second step, the code was modified to respect the `primary`-tag (see Appendix A.7) for activities instead of simply checking for activities of type `work`. The `primary`-tag is set by *fma2persons* or by *cityplans2plans* when called with `-complete=+markPrim`.

In the original setup, other activity types were used than the ones in [12]. As an example, `education` wasn't accepted as activity, but `service` and `business` were. The missing activity `education` was added to the code, replacing `business`.

5.2.3 Usage

The quality of the results depends strongly on correct data for the file *landuse.xml* (Appendix A.15). This file contains a listing of map-cells. Each cell contains the capacities for the various activities. Because there is yet no data available about educational landuse, the `education`-activity was mapped to the values of the `business`-activity. This is only a temporary solution to be able to run the process, until better data is available.

6 Model Verification and Results

6.1 Used Data

6.1.1 Population and Activity Patterns

The model used in this paper is the same as in [12] and has 182 zones with a total population of 1'247'566 persons. Using a percentage of 45.44% for the modal split of cars (as calculated by VISEM [13] in [12]), *ascii2population.pl* generates a home population of 568'853 persons for the simulation. These are about 2'000 persons too much in respect of the total population and the modal split. Reasons for this difference were already mentioned in *ascii2populationl.pl* (see 3.3).

Instead of using all 100 activity patterns from [12], only a subset was chosen. In a first step, the number of activities was reduced by merging several rarely used activities into the `work`-activity. Notably, these were the activities `Begleitung/B` (escort), `Service/W` (service), `Geschaeftsreise/G` (business trip) and `Dienstreise/D` (travelling on company business).

In a second step, all activity patterns matching at least one of the following requirements were chosen:

- Activity pattern consists of 4 or less activities.
- Number of occurences is at least 1% or more of all patterns.

This way, 21 activity patterns remained, representing nearly 93% of all 100 given activity patterns. Table 1 shows the different patterns with their frequencies, when the total population with a modal split of 45.44% is used to generate the home population.

6.1.2 OD-matrices

VISEM [13] can generate OD-matrices on a hourly basis. The used OD-matrices in this work were generated as described in [12].

It must be noted that VISEM generates and stores fraction of trips in its OD-matrices. This makes sense from a mathematical point of view, but it does not otherwise. As MATSIM [8] only manages complete day plans, the values in the OD-matrix are rounded to integer values.

VISEM generates a total of 1'372'323 car trips within 24 hours (see [12]), fractional trips included. If only the rounded trips are counted, a total of 1'345'625 trips results.

Table 2 shows a listing of the number of trips in the OD-matrices as well as the number of trips when rounding the values in the matrices.

Pattern	number of persons	trips per pattern	number of trips
h-l-h	157'392	2	314'784
h-w-h	149'849	2	299'698
h-s-h	94'391	2	188'782
h-e-h	69'097	2	138'194
h-w-l-w-h	17'442	4	69'768
h-l-l-h	13'867	3	41'601
h-w-s-w-h	9'960	4	39'840
h-s-l-h	8'996	3	26'988
h-l-s-l-h	6'214	4	24'856
h-w-w-h	10'014	3	30'042
h-s-s-h	5'027	3	15'081
h-l-s-h	4'568	3	13'704
h-l-w-h	4'113	3	12'339
h-w-l-h	5'643	3	16'929
h-w-s-h	4'516	3	13'548
h-e-l-h	2'309	3	6'927
h-s-w-h	2'698	3	8'094
h-e-e-h	1'217	3	3'651
h-l-e-h	659	3	1'977
h-w-e-h	494	3	1'482
h-e-s-h	387	3	1'161
total	568'853	weighted avg = 2.23	1'269'446

Table 1: Frequencies of the 21 patterns used with start population "home".

hour	0 – 1	1 – 2	2 – 3	3 – 4	4 – 5	5 – 6
n. of trips	140	236	104	833	3'365	11'226
rounded	4	33	1	415	2'588	10'195
hour	6 – 7	7 – 8	8 – 9	9 – 10	10 – 11	11 – 12
n. of trips	62'436	94'332	69'250	65'955	65'980	95'810
rounded	61'182	92'971	67'999	64'774	64'822	94'404
hour	12 – 13	13 – 14	14 – 15	15 – 16	16 – 17	17 – 18
n. of trips	112'805	108'363	74'579	77'953	103'590	134'711
rounded	111'362	107'010	73'293	76'605	102'156	133'293
hour	18 – 19	19 – 20	20 – 21	21 – 22	22 – 23	23 – 24
n. of trips	95'019	67'735	46'963	30'690	30'386	19'863
rounded	93'653	66'366	45'664	29'338	29'019	18'478

Table 2: The number of trips in the OD-matrix
trips: the sum of all trip counts, unrounded
rounded: the sum of all rounded trip counts

As can be seen from table 2, the fewest trips have place in the hour from 2 o'click to 3 o'clock in the night. (Note: in Table 2, the sum of all trips is 1'372'324, one trip more because of rounding errors).

6.2 Scenarios

The availability of three different main processes with a variety of other settings allows to run a multitude of different scenarios.

The processes can be started with different orders of the OD-matrices as well as different start-populations. Further, the durations of the activities can be varied. In the case of *fma2plans*, it has itself several variations which must be chosen and set at compile time.

Table 3 shows a list of possible scenarios. The first value in the compile flag row stands for `FAVOR_HOME_TRIPS`, the second value for `FAVOR_VISITORS` (see chapter 4.4, *fma2plans*).

	process	start time	end time	population (see text)	activity durations w . e . l . s [hours]	compile flags (see text)
1	fma2trips	00	24	home	8 . 4 . 4 . 1	
2	fma2persons	00	24	home	8 . 4 . 4 . 1	
3	fma2persons	03	27	home	8 . 4 . 4 . 1	
4	fma2persons	03	27	auto2	9 . 6 . 5 . 2	
5	fma2persons	03	27	auto2	10 . 8 . 6 . 3	
6	fma2plans	00	24	home	8 . 4 . 4 . 1	00
7	fma2plans	03	27	home	8 . 4 . 4 . 1	00
8	fma2plans	00	24	home	8 . 4 . 4 . 1	01
9	fma2plans	03	27	home	8 . 4 . 4 . 1	01
10	fma2plans	00	24	home	8 . 4 . 4 . 1	10
11	fma2plans	03	27	home	8 . 4 . 4 . 1	10
12	fma2plans	00	24	home	8 . 4 . 4 . 1	11
13	fma2plans	03	27	home	8 . 4 . 4 . 1	11
14	fma2plans	00	24	auto2	8 . 4 . 4 . 1	00
15	fma2plans	00	24	auto1	8 . 4 . 4 . 1	11
16	fma2plans	00	24	auto2	8 . 4 . 4 . 1	11

Table 3: Possible scenarios.

The start-population can be one of the following three groups:

home The total population of every zone (totally 1'247'566 persons) is multiplied with the global modal split of 45.44%. This means that in every zone the same percentage of people use their cars. While ensuring that every activity pattern is used at least once in every zone, a total of 568'853 people is listed as start population in *population.xml* (Appendix A.6).

auto1 As there are differences between the population in cities and in the county, it is not very realistic that the modal split is the same everywhere. VISEM [13] asks for differentiated population groups for input. Based on those information, it is possible to get the number of people who own or

at least have access to a car for every zone. This gives a total of 789'710 people available for generating trips.

auto2 The number of people when using the population *auto1* is higher than when the global modal split is used. To prevent that too much people are generated, the distribution of car-owners is normalized over all zones so that the total number matches the number of people when using the global modal split. After rounding the values and distributing the activity patterns among the zones, a total of 568'909 people are stored in *population.xml*.

In all scenarios, all 24 hours of a day get processed. But while the simulation starts at midnight in some cases, the start-time was set to 3am in other cases. 3am was chosen as this is the hour after the one with the fewest traffic, which is often interpreted as the "real" end of a day. The OD-matrices for the hours from 24 to 27 are basically the same as for 00 to 03. Only the time information in the files needs to be changed to reflect the desired behaviour, because *fma2persons* (as well as the other main processes too) uses this internally stored time to set the end-time of the first activity for newly generated people.

6.3 Number of Generated Day Plans

The goal of each process must be to generate as many trips as possible according to the trips in the OD-matrices. Where the population is respected (*fma2persons* and *fma2plans*), it is also important to generate as many people as available in the start population (and thus this number of day plans), because the number of trips depends heavily on the number of people in transit. Generating too many people is not desired as the numbers should reflect the reality.

6.3.1 fma2trips

Because *fma2trips* generates a new day plan for every trip in the OD-matrices, the number of day plans corresponds exactly to the number of trips. This means that in the model used, a total of 1'345'625 day plans are generated. But it must be noted, that those "day plans" are not real day plans, because they only describe a single trip and no round trips.

6.3.2 fma2persons

Generating day plans using *fma2persons* according to scenarios 2 and 3 shows that every person from the previously generated population gets a day plan assigned. The last person is scheduled to leave its home in the hour between 7pm and 8pm, no matter whether the simulation-day starts at midnight or at 3 o'clock in the morning. This is of interest, as other situations may exist where not enough trips from a zone are available in the OD-matrices for everyone to leave.

The first time where not all trips listed in a OD-matrix can be generated is in the hour between 9am and 10am, again in both scenarios. This means also, that in this hour the first person will travel further from its first location in the day

plan (home-location not counted) to the second place (which might be home again).

Scenario 4 and 5 generate both the exact same number of people for every hour. The different durations do not make any difference. This can be easily explained with the fact that once a person is generated, it is written out to a file and has no longer any influence on other plans. The durations will first start to make a difference in the MATSIM-simulation.

Comparing the different start-populations (scenarios 2 or 3 vs. 4 or 5) gives no surprise: The hour in which for the first time no persons can be generated differs between the two variants, being the hour between 9am and 10am for the `home`-variant versus the hour from 7 to 8 for `auto2`. This is interesting especially given that scenario 4 starts with a slightly bigger population than scenario 3 (568'909 vs. 568'853 people).

Another interesting point is, that not all people from the start-population are used in scenarios 4 and 5. They both generate only 568'568 day plans, making 314 people to stay the whole day at home. Even if these scenarios run until 27 o'clock, the last person is generated before 24 o'clock—in the remaining three hours no more people can be generated. This leaves the question open whether the *auto2*-distribution of the population is indeed better than the general *home*-distribution.

Figure 7 shows a graph depicting the number of trips generated by VISEM [13] and the number of plans generated with *fma2persons* according to the settings to scenarios 3 and 4. The values for scenario 2 are not shown as there is no visible difference to scenario 3, while scenario 5 is not shown as it has exactly the same number of plans as scenario 4. Remarkable are the three peaks (morning and evening rush hour, whereas the peak at noon cannot easily be explained, see [12]) which VISEM generates.

The figure nicely shows how the number of trips diverge from the number of newly generated plans starting at about 10 o'clock.

6.3.3 fma2plans

In a first stage, each of the four variations of *fma2plans* was run once with the OD-matrices starting at midnight and once with the OD-matrices starting at 3 o'clock. The four variations are distinguished through the different allocation of the two compile-flags. Three additional scenarios with a different start-population were computed, leading to a total of 11 scenarious for *fma2plans*. Table 4 gives an overview over the number of generated day plans for different scenarios. As can be seen, the difference of comparable scenarios starting at midnight or at 3 o'clock are minimal in relation to the number of generated plans as well as to the number of plans not completed. Regarding the percentages of plans generated, it is obviously that the combination of *not* favoring home trips and *not* favoring visitions gives the best results concerning the number of generated day plans.

If the number of completed plans is examined, those processes compiled with `FAVOR_HOME_TRIPS = 1` seem to be better at a first glance. This can easily be understand as the algorithm tries to use existing plans, before it generates new plans. This leads to more plans being completed than when new plans would be generated. Because incomplete plans will be completed (see chapter 5.1, *cityplans2plans*, option `-complete`), the total number of trips will

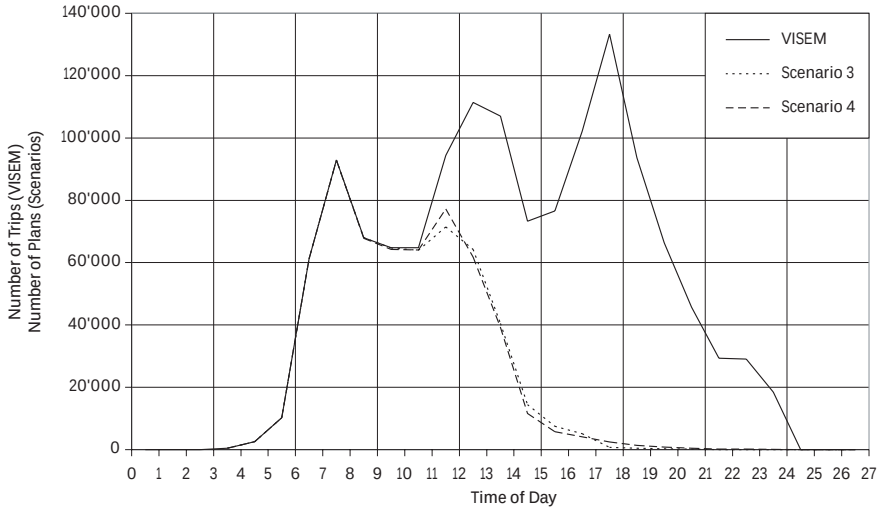

Figure 7: Comparison of the number of trips generated by VISEM and the number plans generated by *fma2persons*.

scenario *fma2plans*		plans generated [a]		plans completed	plans not completed	
6	00 – 24 home 00	568′330	99.9%	480′674	87′656	15.4%
7	03 – 27 home 00	568′315	99.9%	480′916	87′399	15.4%
8	00 – 24 home 01	540′384	95.0%	500′296	40′088	7.4%
9	03 – 27 home 01	540′412	95.0%	500′560	39′852	7.4%
10	00 – 24 home 10	551′581	97.0%	467′029	84′552	15.3%
11	03 – 27 home 10	551′473	96.9%	466′533	84′940	15.4%
12	00 – 24 home 11	540′221	95.0%	500′147	40′074	7.4%
13	03 – 27 home 11	540′018	94.9%	500′085	39′933	7.4%
14	00 – 24 auto2 00	566′771	99.6%	500′236	66′535	11.7%
15	00 – 24 auto1 11	650′761	114.4%	549′816	100′945	15.5%
16	00 – 24 auto2 11	553′111	97.2%	518′759	34′352	6.2%

[a] for population "home": 100% = 568′853 plans
for population "auto1": 100% = 568′853 plans
for population "auto2": 100% = 568′909 plans

Table 4: Number of generated and completed plans per scenario

be similar for both variants. For traffic simulations, the total number of trips, and thus the total number of day plans, is more important than the accuracy of the generated plans, because the plans will be re-scheduled anyway during the simulation.

6.4 Number of Trips Generated

VISEM [13] generated a total of 1'372'323 trips for cars within 24 hours (see [12]), fractional trips included. If only the rounded trips are counted, a total of 1'345'625 trips result. Table 1 shows that using the start-population `home`, *fma2plans* could not generate more than 1'269'446 trips. This is slightly less than the number of VISEM. The difference may be explained in the way the activity patterns were chosen (see chapter 6.1).

6.4.1 fma2trips

Because every trip from the OD-matrix is transformed into a simple day plan, the number of trips generated by *fma2trips* is the same as the number of trips in the OD-matrix. But again the same arguments as in 6.3.1 (*Number of Day Plans Generated – fma2trips*) should be considered: These plans are not real day plans and are mainly used to compare the results of other processes after being processed by MATSIM [8].

6.4.2 fma2persons

When using the start population `home`, all plans are generated. This means, that also the total number of trips will be generated. If the start population `auto2` is used, about 300 people will stay at home the whole day, not executing their activity pattern. It follows that about 700 trips will not be generated—less than 0.5‰ of all trips.

6.4.3 fma2plans

In the case of *fma2plans*, the results depend on the used variation. As is shown in table 5, scenarios 6 and 7 are again the best within the group which start with the `home`-population. Clearly visible is the influence of the compile flag `FAVOR_HOME_TRIPS`: When it is set (scenarios 8, 9, 12 and 13) the number of plans not completed is only about half of the number of when the flag is not set (scenarios 6, 7, 10 and 11).

Looking at the values of scenarios using another start population (scenarios 14 – 16), scenario 15 with the start population `auto1` attracts attention: Nearly all trips are generated after processing the 24 OD-matrices! But at the same time a big number of plans are not completed at this time, leading to too many trips generated when completing those plans. This can be explained that using the population `auto1` more people are available in the zones than in other start populations When in other variations no trips could be generated, there are still people available when using the population `auto1`. This leads to more trips generated as well as to more plans generated which itself is responsible for the higher number of not completed plans at the end.

	scenario *fma2plans*	Trips generated		Plans not completed	Trips of plans completed with "home"	
6	00 – 24 home 00	1'175'032	87.3%	87'656	1'262'688	93.8%
7	03 – 27 home 00	1'175'234	87.3%	87'399	1'262'633	93.8%
8	00 – 24 home 01	1'165'072	86.6%	40'088	1'205'160	89.6%
9	03 – 27 home 01	1'165'368	86.6%	39'852	1'205'220	89.6%
10	00 – 24 home 10	1'131'431	84.1%	84'552	1'215'983	90.4%
11	03 – 27 home 10	1'130'747	84.0%	84'940	1'215'687	90.3%
12	00 – 24 home 11	1'164'654	86.6%	40'074	1'204'728	89.5%
13	03 – 27 home 11	1'164'429	86.5%	39'933	1'204'362	89.5%
14	00 – 24 auto2 00	1'193'634	88.7%	66'535	1'260'169	93.6%
15	00 – 24 auto1 11	1'339'006	99.5%	100'945	1'439'951	107.0%
16	00 – 24 auto2 11	1'199'009	89.1%	34'352	1'233'361	91.7%

Table 5: Number of generated trips per scenario (100% = 1'345'625)

Scenarios 14 and 16 both use the start population `auto2`. They have slightly better numbers when looking at the number of trips generated before completing plans. But because they also generate more complete plans, less trips will be added when completing plans with `home`-activities. But even then, these two scenarios yield better results than many of the other scenarios.

It can be said that the variants with both deactivated `FAVOR_HOME_TRIPS` and `FAVOR_VISITORS` give the best results, both in the number of generated day plans as well as the number of generated trips. The difference between runs with the start-population `home` and the start-population `auto2` is too small to be able to decide which one is better. More thorough tests are necessary to find the one solution that fits better.

6.5 Execution Time

The execution time has significant differences between the various scenarios. Figure 8 shows the times for the different scenarios, split up into the different processes. The times were measured on an Apple PowerBook G4, 1.25GHz G4 processor (Motorola PowerPC 7475, a low-power RISC-Processor for mobile computers) with 1 GB RAM, running Mac OS X 10.3.4 and with the built-in harddisk running at 4200 RPM.

One can clearly see that *fma2trips* is the fastest process—but again with the least meaningful results. The times for *fma2persons* itself is quite constant, but the durations of *LocationChoice* fluctuates significantly. This may be explained in the different numbers of plans which have secondary activities depending on which start-population was used.

Remarkable are the times for *fma2plans* with both compiler flags deactivated (see chapter 4.4). They run significantly longer than any other variant, but also give the best results as seen earlier. This is mainly because the visitors list in each zone is searched more often and more thorough than in other variants. The (a little) shorter execution time when using the start-population `auto2` can be explained that through the better disposition less people are stuck in a zone, keeping the visitors list short.

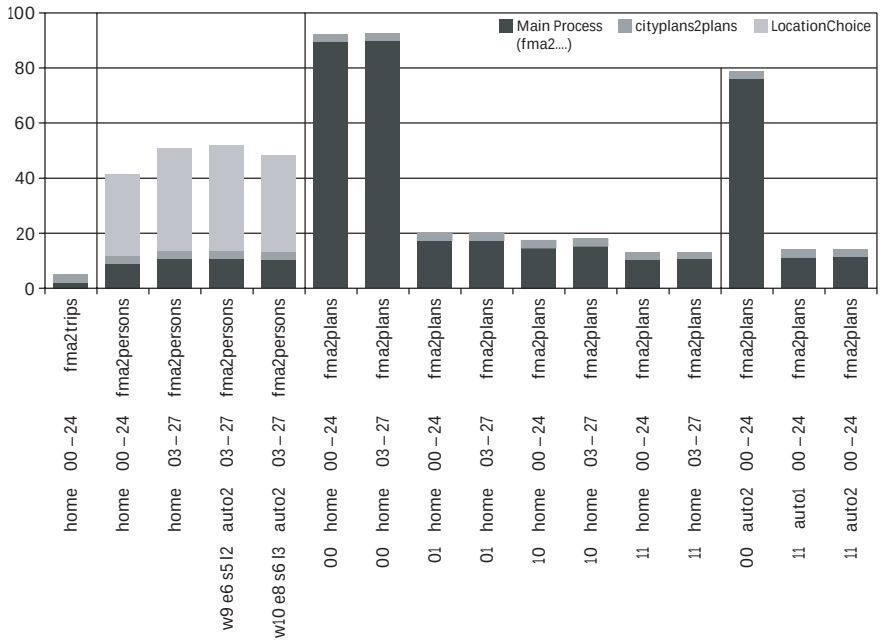

Figure 8: Execution Time in Minutes

6.6 Comparison with Counting Stations

VISUM [13] output will stay as the reference to which the MATSIM [8] output will be compared. MATSIM is known to produce results similar to VISUM [11]. Nevertheless, a similarity check is also done by a one-to-one conversion of each trip of the OD-matrices into 'one-trip-plans', as generated by *fma2trips*. As already mentioned, those are not 'real' plans but the produced traffic should be similar to the one VISUM generated.

The following scenarios are used for the following comparison:

fma2trips: scenario 1 from table 3

fma2personsW8: scenario 2 from table 3

fma2personsW9: scenario 4 from table 3; extended default durations for activities

fma2personsW10: scenario 5 from table 3; even more extended durations for activities

The main distinction is the different duration definition, since this is the main assumption made for the day plan generation process. The three generated day plans are used as initial plans for MATSIM [8] iterating 50 times using the route replanning modul [11]. At the same time VISUM can generate a steady assignment based on the OD-matrices.

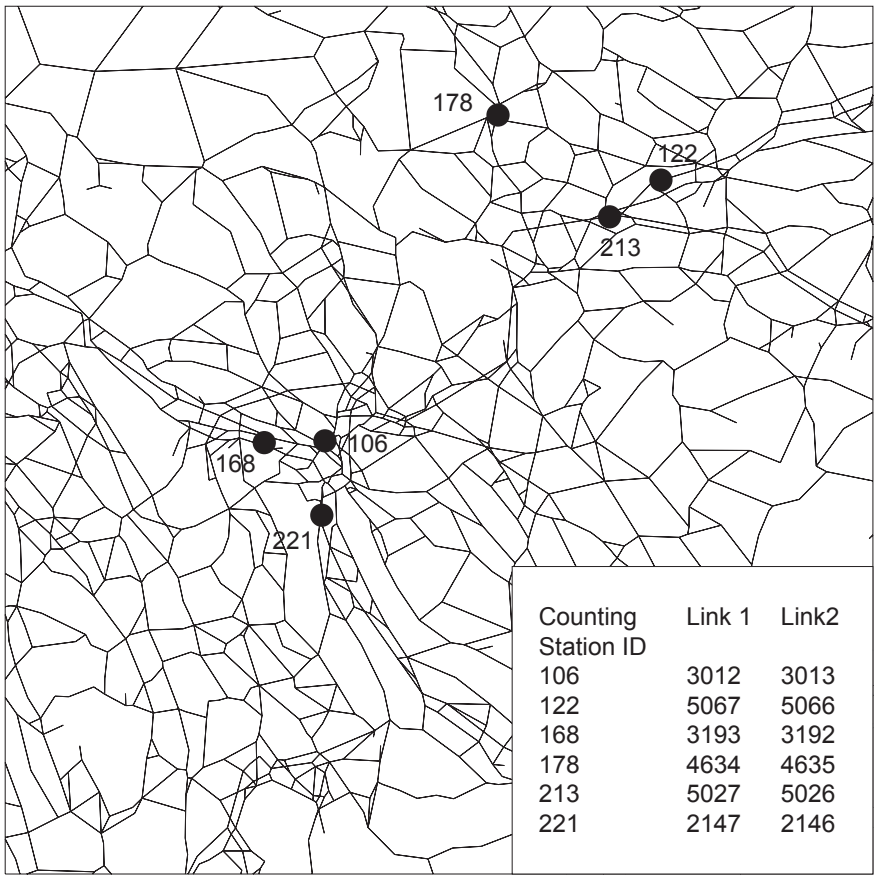

Figure 9: Location of the used Counting Stations in the Area of Zurich, Switzerland.

The results of MATSIM [8] will be compared to the VISUM reference output. The comparison will be done only for the motorized private transport since MATSIM [8] is only able to handle this mode at the moment.

The MATSIM-simulation as well as the VISUM assignment use the same underlying network [6]. Some changes made to this network are described in [15]. The comparison of the outcome of the simulations will be done at 12 links (see Figure 9) of the given network where also counting data is available [2]. But direct comparison with the field volumes will not be made since the VISEM output has already produced too much traffic, mentioned above.

Figure 10 shows the hourly volumes of four of the twelve links for which counts are available. VISUM shows the three peaks already mentioned above. This differs substantially from the hourly volumes recorded at the counting stations, so a comparison to real world data does not make sense until the calibration of VISEM has been improved. The comparison between VISUM

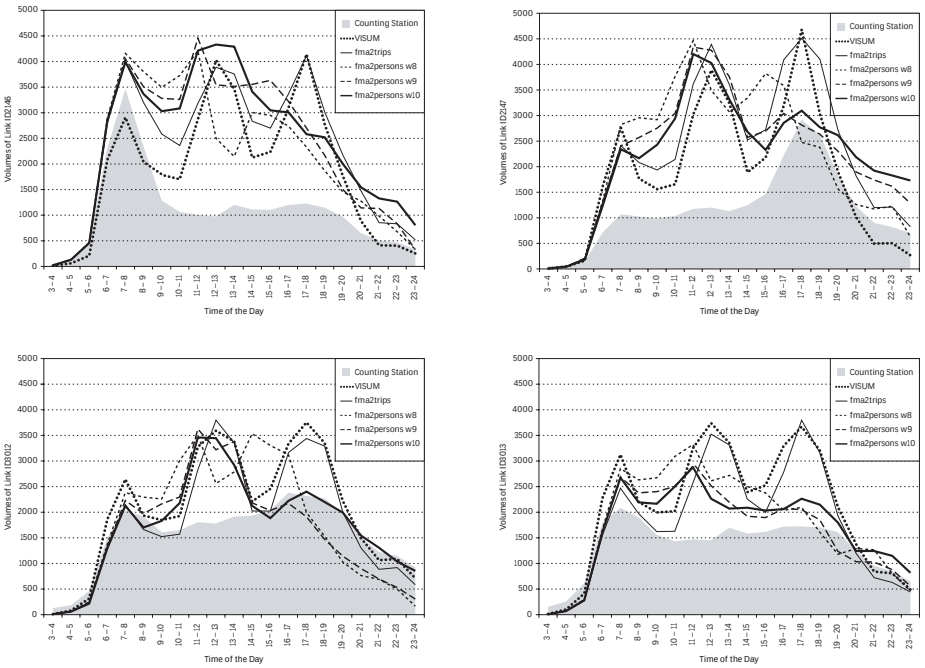

Figure 10: Hourly Volumes from VISUM, MATSIM and Counting Stations

and OD2Trips shows known similarities [11]. That confirms again that VISUM and MATSIM produce comparable results.

At a first sight, the hourly volumes of *fma2personsW8* differ strongly from the expected results. This is not really surprising since the activity durations are just mere assumptions. But giving it a closer look, there are still some similarities:

- The three peaks are still there
- The morning peak matches pretty well
- The other peaks appear too early in the day (noon peak around 11am, evening peak around 4pm)

This leads us to the other two duration definitions (*fma2plansW9*, *fma2plansW10*). Especially *fma2plansW10* matches the first two peaks quite well. Also, the third peak appears at the right time but with too low volumes, while the volumes late in the night are too large.

This also shows up if we compare the absolute and relative errors (see Figure 11) from each MATSIM [8] run with VISUM averaged over the twelve given links in Figure 9.

The absolute error is calculated as follows:

$$E_{absolute} = \frac{1}{n} * \sum_{i=1}^{n} |v_i - p_i| \tag{1}$$

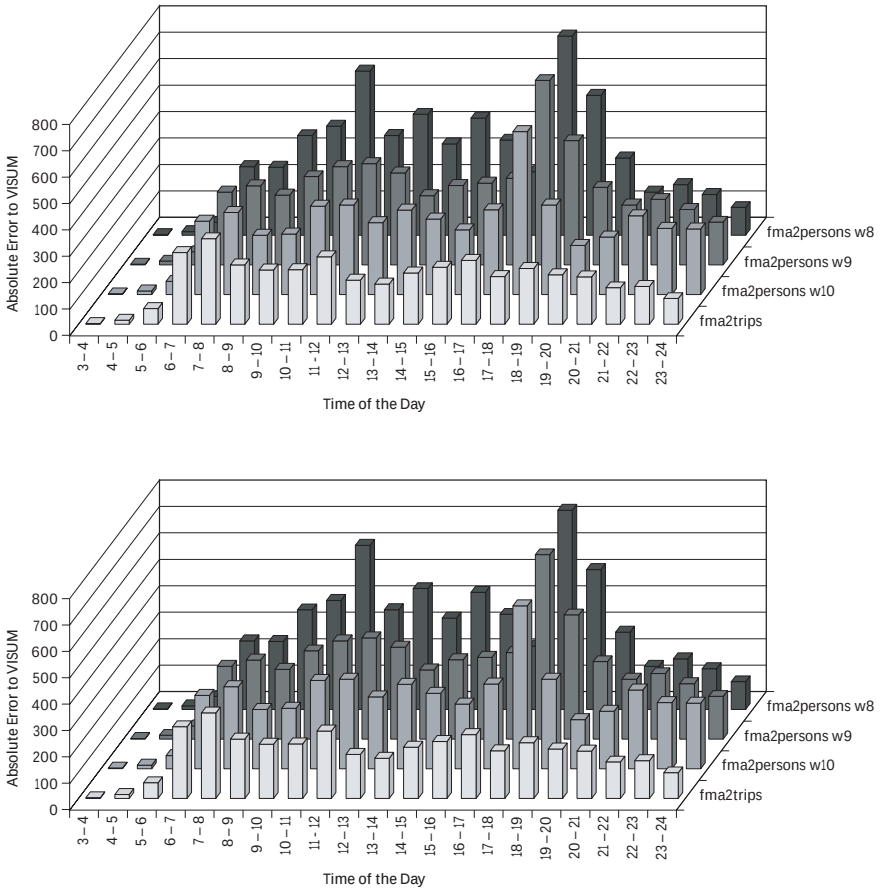

Figure 11: Hourly Average Absolute and Relative Errors of all 12 Counting Stations.

with n the number of counting stations (= 12), v_i the number of trips calculated by VISUM and p_i the number of trips according to the used day plan generation process.

The relative error is calculated as follows:

$$E_{relative} = \frac{1}{n} * \sum_{i=1}^{n} \frac{|v_i - p_i|}{v_i + p_i} \tag{2}$$

fma2plansW10 matches the best. Until 5pm the errors of *fma2plansW10* and *fma2trips* are surprisingly similar. More discrepancies do not appear until the evening peak. The reason for this difference lies again in the duration definition. The duration of a day plan therefore can differ between 3 hours (e.g. `h-s-h`) to 16 hours (e.g. `h-w-l-w-h`). If more activity chains were used, the duration of a day plan could go up to 27 hours (`h-w-l-e-s-h`)! In one sentence, short chains normally produce too short out-of-home durations while

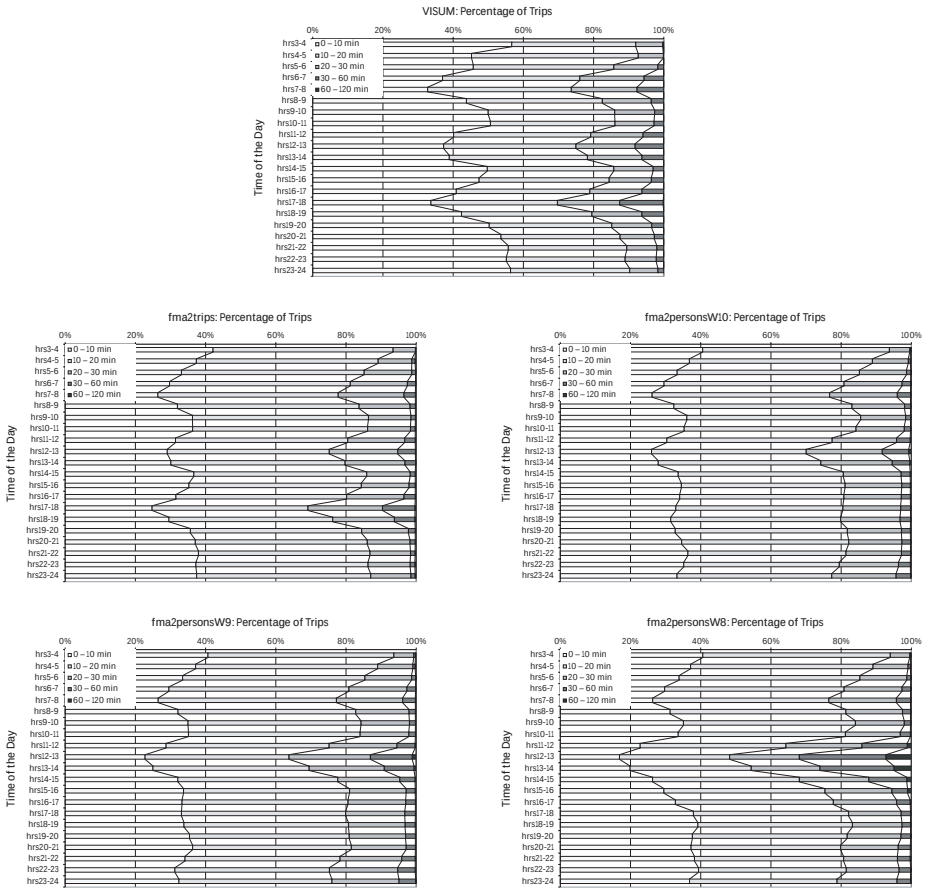

Figure 12: Trip Travel Time Distribution from VISUM and MATSIM

the reverse is true for longer ones. The results are still surprisingly good for the crudeness of the assumptions made.

6.7 Travel Time Analysis

Figure 12 shows the travel time distributions of VISUM [13] and MATSIM [8]. First, one can recognize that the three peaks again shows up in VISUM, *fma2trips* and also in *fma2personsW10*, since the travel times increase at these times of the day. *fma2personsW9* and *fma2personsW8* on the other hand just show the first two peaks. Even more, *fma2personsW8* produces very long trips during noon time which is not surprising because the noon and the evening peak are squeezed together.

Another interesting point shown in Figure 12 is that VISUM produces less average trips (10–30 minutes) than *fma2trips* but more short and long ones. The reason for that could be due to the fact that all individuals in MATSIM try to

optimize their travel times by using the MATSIM router. If individuals find faster routes, they slow down the other individuals using those routes. But during the MATSIM iteration process, the width of the travel time distribution decreases as well as the average travel time.

7 Conclusions

Converting demand or OD-matrices to day plans offers new possibilities to get input data for agent based micro-simulations. The described processes show a promising approach to reconstructing day plans from OD-matrices, even if this problem is highly underdetermined. Moreover, since the behavior of individuals is not easy to determine without expensive surveys, day plan reconstruction is an alternative.

The experience gained from this project shows that by using OD-matrices, a lot of behavioral data gets lost which is necessary for micro-simulations like MATSIM [8]. It must be questioned whether it is easier trying to convert OD-matrices into day plans, or if creating day plans from other statistical input might be more useful. The latter should also be feasible when no microcensus data but only common census data is available. By using additional information like land-use data, population census data, modal split and activity chain distributions, the reconstruction can be done, too.

8 Future Work

While the first results look promising, there are still several issues where further research can be done:

Better default durations for activities In a simple step, the definition of the average duration can be extended. For example, the activity chain `hlh` defines an average duration of leisure which is longer than the leisure activity in `h-w-l-w-h`. With this extension, the problems mentioned above will be reduced. Because for *fma2persons* the primary activity for each activity pattern already must be defined, one can think about an extended definition of activity patterns including the duration for each activity in a pattern.

Dynamic Allocation of Departure Times and Durations Another promising, partly explored approach for MATSIM [8] optimizes departure time and the duration of an activity [15], so that badly chosen activity durations will be corrected. Before it can be applied, the utility function needs to be properly estimated, but see [3] for first experiences.

Alter Default Durations of Activities Based on OD-matrices When using the main-process *fma2plans* it can happen that a person leaves a zone sooner (or later) than its default duration defines. Until now, the person is moved to the new location, but the duration of the last activity is not changed—it stays the same as if the person had left at the scheduled time. By changing the durations according to their real departure time might yield in better results.

option	possible values
random seed	no seed (current status) user-defined seed 1 user-defined seed 2 other seeds ...
time of simulation	00 – 24 03 – 27 05 – 29
activity durations	w8 e4 l4 s1 w9 e6 l5 s2 w10 e8 l6 s3 separately defined for each activity pattern
start population	population `home` with modal-split 45.44% population `auto1` with modal-split 100% population `auto2` with modal-split 100%

compile flags (only *fma2plans*)	`FAVOR_HOME_TRIPS 0 0 1 1`
	`FAVOR_VISITORS 0 1 0 1`
	different behavior in iterations (see above)

Table 6: Possible Options for Future Scenarios

There are a lot of smaller changes which would not change the whole process, but where it might be interesting to see in what changes they result:

- What happens when in *fma2plans* people were chosen by default from the home population instead of randomly from visitors and home population or only from the visitors list?
- *fma2plans* cycles twice through each OD-matrix to generate trips (see 4.4). Already in the first pass people can be chosen which are scheduled to leave at a later hour. It might make more sense that in the first pass only people are chosen which are scheduled to leave in or before the current hour, and only in the second pass people whose activity durations must be shortend.
- Until now, the processes do not use any random seed. It should be checked whether using different, explicit random seeds have a big influence on the results.
- Described above are variants either starting at midnight or at 3am. How would results differ it the day plan generation processes are started at 5am?
- The algorithm to choose random points within a zone is not yet optimal. Small zones in the country with big distances to other zones get for the populated area a too big circle assigned. This could be counteracted by using custom circle-radii for the populated area. The processes are already prepared for this data (see *villages.txt* and *villages.xml*, Appedices A.1 and A.5 respectively).

Table 6 gives an overview over the different settings and parameters that could be used in further tests.

Finally, it would be interesting to generate dayplans directly from the input data VISEM uses instead of the output data of VISEM. In that case, OD-matrices would not be used anymore as input but statistical information about population and traffic in the area under investigation. Of substantial interest would be to see if this direct conversion yields in the same three peaks as VISUM generates (see figure 5), or if the peak at noon would be far smaller as expected.

Acknowledgements

My biggest thank goes to Michael Balmer, who supervised this work and was greatly involved in analyzing the results. His earnest interest in this project was always very motivating for me. I also appreciate very much Prof. Dr. Kai Nagel for coming up with this great topic and his supervision of my project.

I highly esteem the discussions with Prof. Dr. Kay W. Axhausen. His proposals were very interesting and brought in many helpful aspects from outside a computer-scientist's view. I also like to thank Philipp Fröhlich for his help with VISUM.

References

[1] `http://www.activeperl.com`.

[2] Swiss Federal Road Authority. Automatic traffic counts 1999. Bern, Switzerland, 2000. `http://www.astra.admin.ch`. Accessed July 2004.

[3] Charypar, D. and Nagel, K. Generating complete all-day activity plans with genetic algorithms, 10th International Conferece on Travel Behaviour Research, Lucerne, Switzerland. August 2003. `http://www.ivt.baum.ethz.ch/allgemein/iatbr2003.html`. Accessed July 2004.

[4] INRO. `http://www.inro.ca`. Accessed July 2004

[5] ESRI. `http://www.esri.ca`. Accessed August 2004

[6] Federal Office for Spatial Development (ARE). http://www.are.admin.ch/are/en/service/sitemap/index.html. Accessed July 2004.

[7] A. Altenhoff (2003) *Modellierung des Location choice Prozesses in einem aktivitätenbasierten Modell mit kooperierenden Agenten*, Institute for Computational Science, ETH Zürich.

[8] Multi Agent Traffic Simulation, `http://www.matsim.org`

[9] Nagel, K; Balmer, M; Raney, B. *Large scale multi-agent simulations for transportation applications*, ETH Zurich, Switzerland and TU Berlin, Germany, 2004.

[10] Raney, B. and Nagel, K. An improved framework for large-scale multi-agent simulations of travel behaviour. In *Proceedings of Swiss Transport Research Conference (STRC)*, str (2004). See `www.strc.ch`.

[11] Raney, B. and Nagel, K. Iterative route planning for modular transportation simulation. Swiss Transport Research Conference (STRC), Monte Verita, Switzerland 2002. http://www.strc.ch. Accessed July 2004.

[12] Rieser, M. *Berechnung von Nachfragematrizen mit VISEM*, Semesterarbeit, Institut für Verkehrsplanung und Transportsysteme (IVT), ETH Zürich, 2004.

[13] PTV AG, VISEM & VISUM, Karlsruhe, `http://www.ptv.de`

[14] Vovsha, P; Petersen, E; Donnelly, R. Microsimulation in Travel Demand Modeling: Lessons Learned from the New York Best Practice Model. In Transportation Research Record: Journal of the Transportation Research Board, No. 1805, TRB, National Research Council, Washington, D.C., 2002, pp. 68-77

[15] M. Balmer, B. Raney, and K. Nagel. Coupling activity-based demand generation to a truly agent-based traffic simulation - activity time allocation. Paper (submitted), Transportation Research Board Annual Meeting, Washington, D.C., 2005

A Example files

A.1 villages.txt

A plain text file, containing zone-id (integer value), zone-name (string value) and x- and y-coordinate (both integer values) of the center of the zone. Optionally, the radius of a circle for the populated area can be given as integer value. The values must be separated with tab-stops. The zone-name must start with a letter and can contain spaces, numbers, parantheses and hyphens. Lines starting with a star (*) will be ignored.

The following regular expression is used to parse the lines:

```
($id, $name, $xcoord, $ycoord, $radius) =
   /^(\d+)\s+(\w+(?:\s*[a-zA-Z0-9_\(\)\-]*)*)\t+(\d+)\s+(\d+)
                                                    \s+(\d+)/;
```

Example:

```
* id     name              x        y        radius
1        Aeugst am Albis   679257   236349
2        Affoltern am Albis 676549  236982
3        Bonstetten        678055   241570
4        Hausen am Albis   683119   232918
5        Hedingen          676396   239040   416
6        Kappel am Albis   681247   231079
7        Knonau            677523   230863   1239
8        Maschwanden       674813   231983
9        Mettmenstetten    677702   233296
10       Obfelden          674502   235299   875
11       Ottenbach         673216   237008
12       Rifferswil        680083   233127
13       Stallikon         679227   243112
14       Wettswil am Albis 678228   243405
21       Adlikon           694831   271059
22       Benken (ZH)       691299   278808
23       Berg am Irchel    687357   268980
24       Buch am Irchel    689244   267305
25       Dachsen           688631   280091
26       Dorf              691008   269802
27       Feuerthalen       690827   282734
28       ...
```

A.2 population.txt

A file containing a matrix of real numbers in plain text. The numbers represent the number of persons in a zone and in a given population group. A header row contains labels for different population groups, whereas a header column contains zone-ids. The intersection cell of header row and header column must be empty. The values must be separated by spaces or tab-stops.

Example:

```
     A             Aa            Aap
1    0.000192164   227.7224408   0.000296154  ...
2    510.8429427   0.000431773   0.000443067  ...
```

```
3    0.000158329   0.00018706    0.000191953  ...
4    0.000386093   0.000456153   0.000468085  ...
5    0.000239504   0.000282964   0.000290366  ...
6    0.000499918   0.000590632   0.000606083  ...
7    0.000740565   0.000874947   0.000897835  ...
8    0.0042928     0.005071691   0.00520428   ...
9    0.000574902   0.000679222   0.00069699   ...
10   0.000643576   1107.842027   0.00076843   ...
11   1819.681396   0.002256751   0.002330244  ...
12   0.001275368   0.001505718   0.001554747  ...
13   0.000502897   0.00059373    0.000613064  ...
14   261.8125894   972.6631053   0.000436399  ...
21   0.000978575   0.001149149   0.001193921  ...
22   0.000300345   193.9833573   0.000489881  ...
23   0.000403986   0.000473775   0.000492886  ...
24   0.000245071   286.5460953   0.001568631  ...
25   0.00024174    350.1249999   0.000390648  ...
26   0.000493452   0.000575928   0.000602041  ...
27   ...
```

A.3 patterns.txt

A list containing activity patterns in the first row and the corresponding percentages in the second row, separated by blanks (tab-stops or spaces).

The following regular expression is used to parse the lines:

```
($pattern, $frequency) = /$\s*(\w+)\s+(\S+)\s*/;
```

Example:

```
ZFZ      27.74
ZAZ      26.41
ZEZ      16.63
ZSZ      12.17
ZAFAZ    3.06
ZFFZ     2.43
ZAEAZ    1.74
ZEFZ     1.57
ZFEFZ    1.08
ZAAZ     1.75
ZEEZ     0.87
ZFEZ     0.79
ZFAZ     0.71
ZAFZ     0.98
ZAEZ     0.78
ZSFZ     0.39
ZEAZ     0.46
ZSSZ     0.20
ZFSZ     0.10
ZASZ     0.07
ZSEZ     0.05
```

A.4 translation.txt

The file must contain two rows, in each one character, separated with spaces or tab-stops. Activity patterns from *patterns.txt* (see A.3) will be translated from the characters in the first row into the corresponding characters in the second row. A specific character must not be in both columns. Capital and small letters are distinguished as different characters.

The following regular expression is used to parse the lines:

```
($from, $to) = /$\s*(\S)\s+(\S)/;
```

Example:

```
Z       h
A       w
E       s
F       l
S       e
```

A.5 villages.xml

This file is the output of *ascii2villages.pl* (see chapter 3.2). It contains a list of zones with their id, their name and the coordinates of the center of the zone in a structured XML-format.

```
<?xml version="1.0" ?>
<villages xml:lang="de-CH">
  <village id="1" name="Aeugst am Albis" x="679257" y="236349" />
  <village id="2" name="Affoltern am Albis" x="676549" y="236982" />
  <village id="3" name="Bonstetten" x="678055" y="241570" />
  <village id="4" name="Hausen am Albis" x="683119" y="232918" />
  <village id="5" name="Hedingen" x="676396" y="239040" radius="416"/>
  <village id="6" name="Kappel am Albis" x="681247" y="231079" />
  <village id="7" name="Knonau" x="677523" y="230863" radius="1239" />
  <village id="8" name="Maschwanden" x="674813" y="231983" />
  <village id="9" name="Mettmenstetten" x="677702" y="233296" />
  <village id="10" name="Obfelden" x="674502" y="235299" radius="875" />
  <village id="11" name="Ottenbach" x="673216" y="237008" />
  <village id="12" name="Rifferswil" x="680083" y="233127" />
  <village id="13" name="Stallikon" x="679227" y="243112" />
  <village id="14" name="Wettswil am Albis" x="678228" y="243405" />
  <village id="21" name="Adlikon" x="694831" y="271059" />
  <village id="22" name="Benken (ZH)" x="691299" y="278808" />
  ...
</villages>
```

A.6 population.xml

In this XML-file, the number of persons is stored which have not yet a day plan assigned. The persons are already distinguished by activity patterns. This makes it easy to chose a random person with a activity pattern. The numbers decrease with every iteration of a main-process.

```
<?xml version="1.0" ?>
<villages xml:lang="de-CH">
```

```
  <village id="1" name="Aeugst am Albis" x="679257" y="236349">
    <pattern chain="hlh" count="193" />
    <pattern chain="hwh" count="184" />
    <pattern chain="hwlwh" count="22" />
    <pattern chain="hslh" count="11" />
    ...
  </village>
  <village id="2" name="Affoltern am Albis" x="676549" y="236982">
    <pattern chain="hlh" count="1288" />
    <pattern chain="hwh" count="1226" />
    <pattern chain="hwlwh" count="143" />
    <pattern chain="hslh" count="73" />
    ...
  </village>
  <village id="3" name="Bonstetten" x="678055" y="241570">
    <pattern chain="hlh" count="486" />
    ...
  </village>
  ...
</villages>
```

A.7 cityplans.xml

A preliminary plans-file containg the attributes `cityid` and `home_city` instead of x- and y-coordinates. Secondary activities have no location assigned yet. No `<leg>`-tags are created between activities, but the leg-mode is stored as attribute of the person.

```
<?xml version="1.0" ?>
<!DOCTYPE plans SYSTEM "plans.dtd">
<plans xml:lang="de-CH">
<person id="1" home_city="1" leg_mode="car">
  <plan pattern="hwh">
    <act type="h" cityid="1" end_time="08:00:00" />
    <act type="w" dur="08:00" cityid="2" primary="true" />
    <act type="h" cityid="1" />
  </plan>
</person>
<person id="2" home_city="1" leg_mode="car">
  <plan pattern="hlslh">
    <act type="h" cityid="1" end_time="08:00:00" />
    <act type="l" dur="02:00" cityid="2" primary="true" />
    <act type="s" dur="01:00" />
    <act type="l" dur="02:00" />
    <act type="h" cityid="1" />
  </plan>
</person>
...
</plans>
```

A.8 pre-plans.xml

A preliminary plans-file, containing the attributes `x100` and `y100` in place of `cityid`. This is the output of *cityplans2plans* with the option `-cityid2xy`

(see chapter 5.1). Secondary activities have the same coordinates as the home-location. <leg>-tags exist between activities.

```
<?xml version="1.0" ?>
<!DOCTYPE plans SYSTEM "file:/full/path/to/plans.dtd">
<plans xml:lang="de-CH">
<person id="1">
  <plan>
    <act type="h" x100="678753" y100="237424" end_time="08:00:00" />
    <leg mode="car" />
    <act type="w" x100="675549" y100="237156" dur="08:00" primary="true" />
    <leg mode="car" />
    <act type="h" x100="678753" y100="237424" />
  </plan>
</person>
<person id="2">
  <plan>
    <act type="h" x100="678309" y100="236441" end_time="08:00:00" />
    <leg mode="car" />
    <act type="l" x100="676764" y100="237153" dur="02:00" primary="true" />
    <leg mode="car" />
    <act type="s" x100="678309" y100="236441" dur="01:00" />
    <leg mode="car" />
    <act type="l" x100="678309" y100="236441" dur="02:00" />
    <leg mode="car" />
    <act type="h" x100="678309" y100="236441" />
  </plan>
  ...
</person>
```

A.9 plans.xml

A complete, MATSIM-plans-file. This file validates against the document type definition, *plans.dtd* (see A.10).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plans SYSTEM "file:plans.dtd">
<plans xml:lang="de-CH">
  <person id="1">
    <plan selected="no">
    <act type="h" x100="678753" y100="237424" end_time="08:00:00"/>
    <leg mode="car"/>
    <act type="w" x100="675549" y100="237156" dur="08:00" primary="true"/>
    <leg mode="car"/>
    <act type="h" x100="678753" y100="237424"/>
  </plan>
</person>
<person id="2">
  <plan selected="no">
    <act type="h" x100="678309" y100="236441" end_time="08:00:00"/>
    <leg mode="car"/>
    <act type="l" x100="676764" y100="237153" dur="02:00" primary="true"/>
    <leg mode="car"/>
    <act type="s" x100="683250" y100="249450" dur="01:00" />
```

```
    <leg mode="car"/>
    <act type="l" x100="683750" y100="247850" dur="02:00" />
    <leg mode="car"/>
    <act type="h" x100="678309" y100="236441"/>
  </plan>
</person>
...
</plans>
```

A.10 plans.dtd

The Document Type Definition for plans files (*plans.xml*, see A.9).

```
<?xml version="1.0" encoding="utf-8"?>

<!ELEMENT plans       (demand|person)*>
<!ATTLIST plans
          xml:lang    NMTOKEN "de-CH"
>

<!ELEMENT demand      (segment)*>
<!ATTLIST demand
          name        CDATA #IMPLIED
>

<!ELEMENT segment     (model)*>
<!ATTLIST segment
          id          CDATA #REQUIRED
          name        CDATA #IMPLIED
>

<!ELEMENT model       (param)*>
<!ATTLIST model
          name        CDATA #REQUIRED
>

<!ELEMENT param       EMPTY>
<!ATTLIST param
          name        CDATA #REQUIRED
          mean        CDATA #REQUIRED
          type        CDATA #IMPLIED
          stddev      CDATA #IMPLIED
>

<!ELEMENT person      (plan)+>
<!ATTLIST person
          id          CDATA #REQUIRED
          seg_id      CDATA #IMPLIED
>
<!-- possible future person attributes: age, gender, income, ... -->

<!ELEMENT plan        (#PCDATA|act|leg)*>
<!ATTLIST plan
          score       CDATA #IMPLIED
```

```
          age         CDATA #IMPLIED
          selected    (yes|no) "no"
>

<!ELEMENT act         EMPTY>
<!ATTLIST act
          type        CDATA #REQUIRED
          x100        CDATA #REQUIRED
          y100        CDATA #REQUIRED
          link        CDATA #IMPLIED
          zone        CDATA #IMPLIED
          start_time  CDATA #IMPLIED
          end_time    CDATA #IMPLIED
          dur         CDATA #IMPLIED
          primary     CDATA #IMPLIED
>

<!ELEMENT leg         (route)?>
<!ATTLIST leg
          num         CDATA #IMPLIED
          mode        CDATA #REQUIRED
          dep_time    CDATA #IMPLIED
          trav_time   CDATA #IMPLIED
>

<!ELEMENT route       (#PCDATA)>
```

A.11 demand-matrix.fma

A file containing a OD-matrix in the ASCII-format written by VISEM [13]. It contains information about the time of the contained trips, the size of the matrix, and the number of trips in the cells of the matrix.

```
$V;d8
* G:\VISEM_work\output\output_Pkw.fma Pkw
* Zeitintervall
     8     9
* Anteil
    1.00
* Anzahl VZellen
   182
* BezirksNummer
     1            2            3            4            5
     6            7            8            9           10
    11           12           13           14           21
    22           23           24           25          ...
*   1
  0.27343810  20.58377457   1.88410282   3.79979420   3.20691514
  1.13177490   1.00508618   0.84048206   2.58526707   4.16495275
  1.89059031   2.67430234   2.60731125   1.27563977     ...
*   2
  20.52584076  3.29715800  15.68585110  10.78690243   1.88618815
  6.06036186  12.18537617  11.14492035  24.58188629  17.51260948
  17.95108032 12.05580044   5.67500639  13.17440796     ...
```

```
*   3
  2.21606994  16.17108154   0.21675503   0.81266505   0.74204993
  0.47230765   1.05071044   0.99901491   2.87136698   4.76392365
  3.44822097   1.42674959   2.59337425   0.17184621     ...
*   4
  7.40430689  16.96325302   1.77726710   0.16218220   3.11874604
  6.06747580   2.82219243   2.63983226   6.06062365   3.89462209
  1.90754700  11.24550915   1.68430972   0.98741281     ...
*   5
  ...
```

A.12 config.xml

The configuration file used for the three main processes, *fma2trips*, *fma2persons* and *fma2plans*. It contains mainly the file paths of input- and output-file for running the calculations.

The configuration file usually contains full file-paths. This imposes problems when the processes need to be run on different machines with different directory structures. Thus the configuration file is created from a template-file by invoking `make config`. The template-file contains only relative paths based on the cvs root-directory.

```
<?xml version="1.0" ?>
<!DOCTYPE config SYSTEM "config.dtd">

<config>

<module name="peopleGenerator">
  <param name="leg_mode" value="car" />
  <param name="input_population" value="dataIN/poplist.xml" />
  <param name="input_plans" value="dataIN/plans.xml" />
  <param name="output_population" value="dataOUT/poplist.xml" />
  <param name="output_plans" value="dataOUT/plans.xml" />
</module> <!-- peopleGenerator -->

<module name="cityid2xy">
  <param name="minX" value="630000" />
  <param name="minY" value="200000" />
  <param name="maxX" value="730000" />
  <param name="maxY" value="300000" />
  <param name="plansDTD" value="file:src/masim/resources/plans.dtd" />
  <param name="input_plans" value="dataIN/plans.xml" />
  <param name="input_villages" value="dataIN/villagesXY.xml" />
  <param name="output_plans" value="dataOUT/plans.xml" />
</module> <!-- cityid2xy -->

</config>
```

A.13 config.dtd

The Document Type Definition for MATSIM [8] configuration files (see A.12 for an example).

```
<?xml version="1.0" encoding="utf-8"?>

<!-- Allow config to be within config so that included files may be -->
<!-- standalone. -->

<!ELEMENT config      (config|include|module)*>
<!ATTLIST config
          xml:lang    NMTOKEN "de-CH"
>

<!ELEMENT module      (param)*>
<!ATTLIST module
          name        CDATA #REQUIRED
>

<!ELEMENT param       EMPTY>
<!ATTLIST param
          name        CDATA #REQUIRED
          value       CDATA #REQUIRED
>

<!ELEMENT include     EMPTY>
<!ATTLIST include
          file        CDATA #REQUIRED
>
```

A.14 settings.xml

The settings in this file are accessed by more than one process. Because the stored information in this file are dependent on the input data rather then the computing process, the informations are not stored in the configuration file (A.12).

```
<?xml version="1.0" ?>
<!DOCTYPE config SYSTEM "config.dtd">

<config>
<module name="primaryActs">
    <param name="hlh" value="l" />
    <param name="hwh" value="w" />
    <param name="hsh" value="s" />
    <param name="heh" value="e" />
    <param name="hwlwh" value="w" />
    <param name="hllh" value="l" />
    <param name="hwswh" value="w" />
    <param name="hslh" value="l" />
    <param name="hlslh" value="l" />
    <param name="hwwh" value="w" />
    <param name="hssh" value="s" />
    <param name="hlsh" value="l" />
    <param name="hlwh" value="w" />
    <param name="hwlh" value="w" />
    <param name="hwsh" value="w" />
    <param name="helh" value="e" />
```

```
    <param name="hswh" value="w" />
    <param name="heeh" value="e" />
    <param name="hleh" value="e" />
    <param name="hweh" value="w" />
    <param name="hesh" value="e" />
</module> <!-- primaryActs -->

<module name="actDurations"> <!-- duration in minutes -->
    <param name="w" value="480" /> <!-- 8h -->
    <param name="e" value="240" /> <!-- 4h -->
    <param name="l" value="240" /> <!-- 4h -->
    <param name="s" value="60" /> <!-- 1h -->
</module> <!-- ActDurations -->
</config>
```

A.15 landuse.xml

An XML-file containing information about the utility of activities in map squares. This file is used by *LocationChoice* (see 5.2) to determine the location of secondary activities.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE landuse SYSTEM "file:/path/to/landuse.dtd">
<landuse>
  <cell x100="641300" y100="213900">
    <activity type="home" capacity="1"/>
  </cell>
  <cell x100="641300" y100="214500">
    <activity type="home" capacity="3"/>
  </cell>
  ...
  <cell x100="641300" y100="218400">
    <activity type="home" capacity="11"/>
    <activity type="work" capacity="3" start_time="07:00" end_time="19:00"/>
  </cell>
  <cell x100="641300" y100="266600">
    <activity type="shop" start_time="18:30" end_time="22:30"/>
  </cell>
  <cell x100="645900" y100="249100">
    <activity type="home" capacity="30"/>
    <activity type="work" capacity="645" start_time="07:00" end_time="19:00"/>
    <activity type="leisure" start_time="18:30" end_time="22:30" capacity="1000"/>
  </cell>
  ...
</landuse>
```

A.16 landuse.dtd

The Document Type Definition for *landuse.xml* (see A.16).

```
<?xml encoding="UTF-8" ?>

<!-- Created by fmarchal on June 26, 2003, 6:32 PM -->
```

```
<!ELEMENT landuse (cell|activity)+>
<!ATTLIST landuse
          xml:lang    NMTOKEN "de-CH"
>

<!ELEMENT cell (activityref|activity)*>
<!ATTLIST cell
        x100    CDATA #REQUIRED
        y100    CDATA #REQUIRED

>

<!ELEMENT activityref EMPTY>
<!ATTLIST activityref
        id      IDREF #REQUIRED
        capacity    CDATA   #IMPLIED
        start_time  CDATA   #IMPLIED
        end_time    CDATA   #IMPLIED
        >

<!ELEMENT activity EMPTY>
<!ATTLIST activity
        id       ID      #IMPLIED
        type     CDATA   #REQUIRED
        capacity CDATA   #IMPLIED
        start_time  CDATA   #IMPLIED
        end_time    CDATA   #IMPLIED
>
```

B SVG – Scalable Vector Graphics

SVG is an XML-based file-format for two-dimensational vector graphics. The SVG file-format is an official standard of the World Wide Web Consortium (`http://www.w3.org`). Most web browsers can display SVG graphics, either natively or by use of a plugin. The graphics even support animations and interactions without loss of quality by integrating proven standards like CSS (Cascaded Style Sheets) and JavaScript.

Because the graphics are vector-based, zooming into images to see details is no problem. Many features known from the PostScript-Language and professional illustration tools (e.g. Bezier-Curves, Splines, special text-formatting, masks and opacity) are part of SVG, allowing to programmatically create stunning graphics often not possible with other graphic libraries. Many GIS-Applications already started to support SVG, either via converters or as a native file format.

One drawback is that large graphics with a lot of details can take quite some time to load and display. But with the advance of faster computers and with the use of intelligent algorithms (like only drawing those elements in the visible area), this problem is no roadblock.

`http://www.w3.org/Graphics/SVG/` The official SVG site from W3C. Contains links to official resources as well as general news about SVG.

`http://www.w3.org/TR/SVG/` The official specifications from W3C. Contains detailed specifications with DTDs and very basic examples.

`http://www.w3schools.com/svg/svg_examples.asp` Some basic examples. Good for learning the basics by looking at examples.

`http://svg.tutorial.aptico.de/` A very good and detailed introduction with a lot of easy-to-follow examples. My personal favorite!