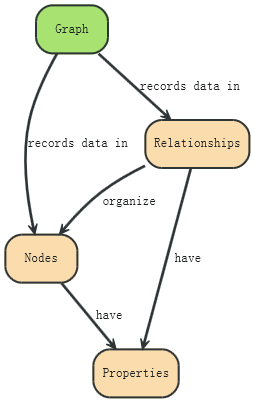
# Neo4J

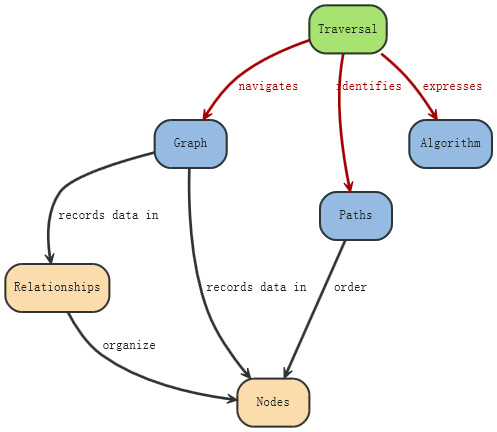
## 图形数据库

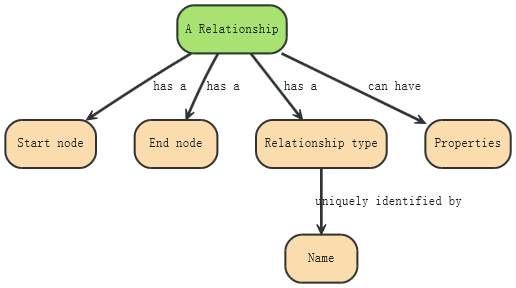
图包含点和连线。“A Graph —records data in→ Nodes —which have→ Properties”图将数据记录在点中，点包含属性。

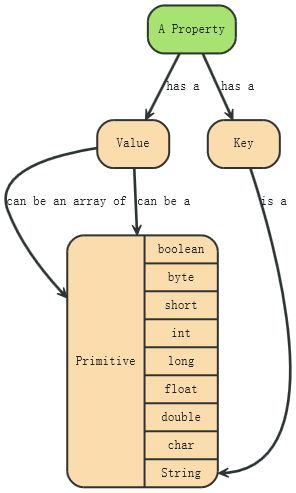


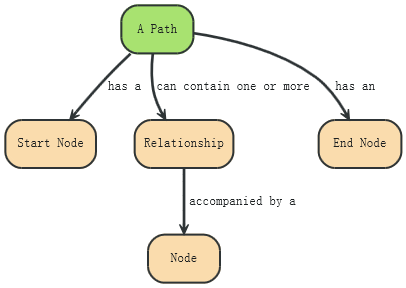
连线组织成图。“Nodes —are organized by→ Relationships —which also have→ Properties”点被连线联系起来，连线也包含属性。

通过遍历查询图。“A Traversal —navigates→ a Graph; it —identifies→ Paths —which order→ Nodes”遍历查询图，搜索出由点组成的路径。

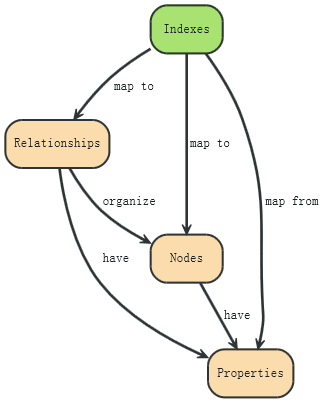




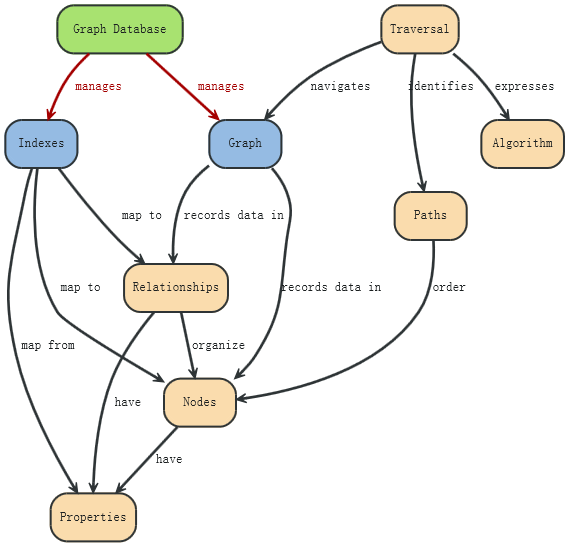




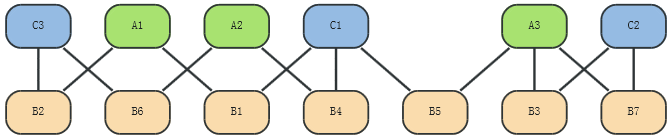
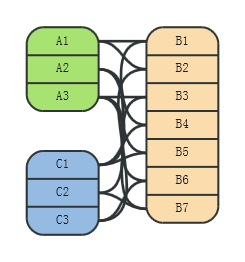
索引查询点和连线。“An Index —maps from→ Properties —to either→ Nodes or Relationships”索引将属性映射到点或连线。



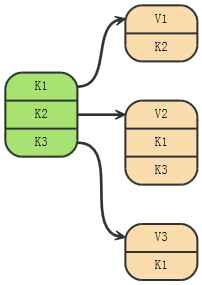
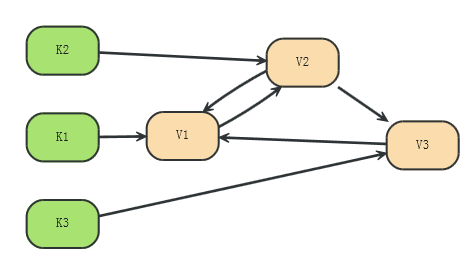
Neo4J是图形数据库。“A Graph Database —manages a→ Graph and —also manages related→ Indexes”图形数据库管理图和索引。



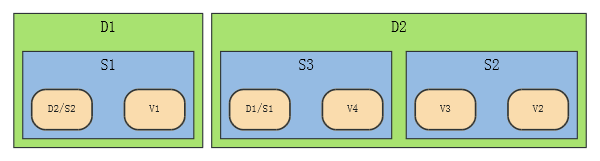
关系型数据库到图形数据库转换



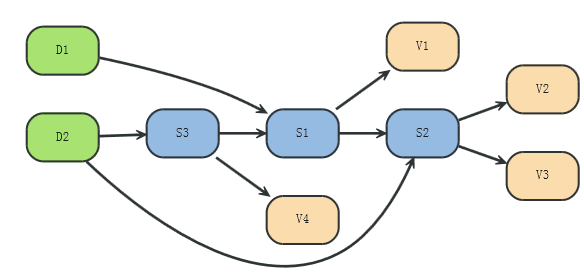
图形数据库描述键值存储方式

图形数据库描述文档结构



D=Document, S=Subdocument, V=Value, D2/S2 = reference to subdocument in (other) document.



## Cypher Query Language

Cyper是一种图形查询语言，通过表达式更新和查询图而不需写复杂的遍历算法。Cyper在逐步成熟中，意味着将来某些语法可能改变，且不像Neo4J其他组件那样经过严格测试。

Cyper由几部分组成：

START：图开始点，通过索引或ID获得。

MATCH：图模式符合。

WHERE：过滤条件。

RETURN：返回值。

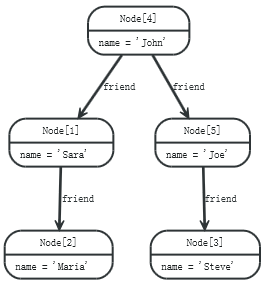
CREATE：生成点和连线。

DELETE：删除点，连线和属性。

SET：设置属性值。

FOREACH：执行更新操作。

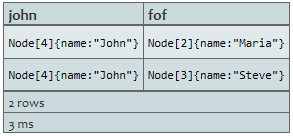
WITH：划分查询成多个部分。



在索引中查找John，然后遍历图找到John的朋友的朋友，返回John和所有找到的朋友。

|  |
| --- |
| START john=node:node\_auto\_index(name = 'John')  MATCH john-[:friend]->()-[:friend]->fof  RETURN john, fof |

返回值：



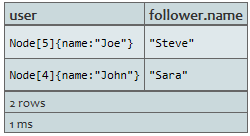
通过ID查找一系列用户，然后遍历图搜索有下属的人的朋友，返回名字中以s开头的朋友

START user=node(5,4,1,2,3)

MATCH user-[:friend]->follower

WHERE follower.name =~ /S.\*/

RETURN user, follower.name



## Neo4j应用

### 新建

解压，Jar包在lib/directory下，Eclipse单击Path->Configure Build Path，Add External JARs，将所有Jar包导入。

生成嵌入式图形数据库，嵌入式图形数据库实例可以由多线程共享，但不能生成多个实例都指向同一数据库。如果DB\_PATH下没有数据库则会新建一个。新建数据库是非常耗时的，因此不要每次都建立新实例。

graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( DB\_PATH );

registerShutdownHook( graphDb );

可用专门线程关闭数据库

private static void registerShutdownHook( final GraphDatabaseService graphDb )

{

// Registers a shutdown hook for the Neo4j instance so that it

// shuts down nicely when the VM exits (even if you "Ctrl-C" the

// running example before it's completed)

Runtime.getRuntime().addShutdownHook( new Thread()

{

@Override

public void run()

{

graphDb.shutdown();

}

} );

}

根据配置打开数据库

GraphDatabaseService graphDb = new GraphDatabaseFactory().

newEmbeddedDatabaseBuilder( "target/database/location" ).

loadPropertiesFromFile( pathToConfig + "neo4j.properties" ).

newGraphDatabase();

所有操作都必须在transaction中执行

Transaction tx = graphDb.beginTx();

try

{

// Mutating operations go here

tx.success();

}

finally

{

tx.finish();

}

生成数据

firstNode = graphDb.createNode();

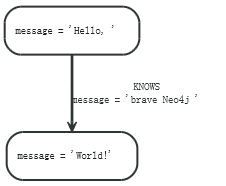
firstNode.setProperty( "message", "Hello, " );

secondNode = graphDb.createNode();

secondNode.setProperty( "message", "World!" );

relationship = firstNode.createRelationshipTo( secondNode, RelTypes.KNOWS );

relationship.setProperty( "message", "brave Neo4j " );



删除数据

firstNode.getSingleRelationship( RelTypes.KNOWS, Direction.OUTGOING ).delete();

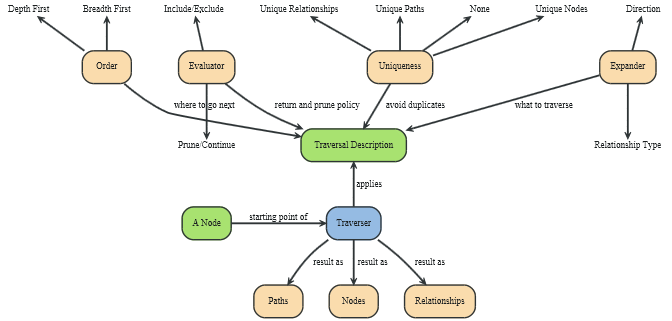
firstNode.delete();

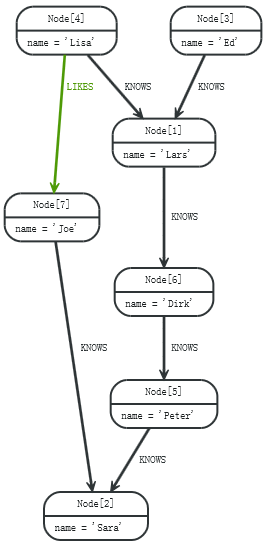
secondNode.delete();

### 遍历

遍历框架包含

1. Expanders：定义遍历对象，特别是连线方向和类型；
2. Order：顺序是深度优先还是广度优先；
3. Uniqueness：是点或连线或路径只经过一次；NONE：可重复访问；NODE\_GLOBAL：点只能访问一次，需要耗费大量内存保存访问过的点；RELATIONSHIP\_GLOBAL：连线只能访问一次，比点耗费的内存还大；NODE\_PATH：点在路径未到达其前不会被访问；RELATIONSHIP\_PATH：连线在路径未到达其前不会被访问；NODE\_RECENT：以一定内存保存最近被访问过的点，内存大小可在TraversalDescription.uniqueness()中设置；RELATIONSHIP\_RECENT：同前。
4. Evaluator：返回值或终止和继续条件；有四种类型：Evaluation.INCLUDE\_AND\_CONTINUE, Evaluation.INCLUDE\_AND\_PRUNE包含但不继续; Evaluation.EXCLUDE\_AND\_CONTINUE; Evaluation.EXCLUDE\_AND\_PRUNE
5. Starting Node：起始点；
6. TraversalDescription：定义和初始化遍历的主要接口；
7. Relationships：如果为空则遍历所有类型的连线，如果设置，只有相应类型会遍历。有including direction和excluding direction两种方法，后一种是指双向。
8. Traverser：调用TraversalDescription.traverse()的结果，代表图中的遍历位置；
9. BranchSelectior：用于选择下一个被访问的分支，Traversal.preorderDepthFirst();Traversal.postorderDepthFirst();Traversal.preoderBreadthFirst();Traversal.postorderBreadthFirst()广度优先比深度优先内存耗费大；
10. PathExpander/RelationshipExpander：





traverse(node)设置起始点为Joe，方法是depthFirst，连线类型是Rels.KNOWS和Rels.LINKES，evaluator为最深5层

for ( Path position : Traversal.description() .depthFirst() .relationships( Rels.KNOWS )

.relationships(Rels.LIKES,Direction.INCOMING ).evaluator(Evaluators.toDepth(5) ).traverse( node )){ output += position + "\n";}

结果：

(7)

(7)<--[LIKES,1]--(4)

(7)<--[LIKES,1]--(4)--[KNOWS,6]-->(1)

(7)<--[LIKES,1]--(4)--[KNOWS,6]-->(1)--[KNOWS,4]-->(6)

(7)<--[LIKES,1]--(4)--[KNOWS,6]-->(1)--[KNOWS,4]-->(6)--[KNOWS,3]-->(5)

(7)<--[LIKES,1]--(4)--[KNOWS,6]-->(1)--[KNOWS,4]-->(6)--[KNOWS,3]-->(5)--[KNOWS,2]-->(2)

(7)<--[LIKES,1]--(4)--[KNOWS,6]-->(1)<--[KNOWS,5]--(3)

只深度遍历一次连线类型为KNOWS，

final TraversalDescription FRIENDS\_TRAVERSAL = Traversal.description().depthFirst()

.relationships( Rels.KNOWS ).uniqueness( Uniqueness.RELATIONSHIP\_GLOBAL );

(7)

(7)--[KNOWS,0]-->(2)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)<--[KNOWS,3]--(6)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)<--[KNOWS,3]--(6)<--[KNOWS,4]--(1)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)<--[KNOWS,3]--(6)<--[KNOWS,4]--(1)<--[KNOWS,5]--(3)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)<--[KNOWS,3]--(6)<--[KNOWS,4]--(1)<--[KNOWS,6]--(4)

在以上基础上遍历深度为3

for ( Path path : FRIENDS\_TRAVERSAL.evaluator( Evaluators.toDepth( 3 ) ).traverse( node ) )

{

output += path + "\n";

}

(7)

(7)--[KNOWS,0]-->(2)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)<--[KNOWS,3]--(6)

从深度2到4遍历

for ( Path path : FRIENDS\_TRAVERSAL.evaluator( Evaluators.fromDepth( 2 ) )

.evaluator( Evaluators.toDepth( 4 ) ).traverse( node ) )

{

output += path + "\n";

}

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)<--[KNOWS,3]--(6)

(7)--[KNOWS,0]-->(2)<--[KNOWS,2]--(5)<--[KNOWS,3]--(6)<--[KNOWS,4]--(1)

获得遍历节点的名称

for ( Node currentNode : FRIENDS\_TRAVERSAL.traverse( node ).nodes() )

{

output += currentNode.getProperty( "name" ) + "\n";

}

Joe

Sara

Peter

Dirk

Lars

Ed

Lisa

获得遍历连线

for ( Relationship relationship : FRIENDS\_TRAVERSAL.traverse( node ).relationships() )

{

output += relationship.getType() + "\n";

}

KNOWS

KNOWS

KNOWS

KNOWS

KNOWS

KNOWS

### 最短路径

Dijkstra能根据权重计算最短路径。

Node startNode = graphDb.createNode();

Node middleNode1 = graphDb.createNode();

Node middleNode2 = graphDb.createNode();

Node middleNode3 = graphDb.createNode();

Node endNode = graphDb.createNode();

createRelationshipsBetween( startNode, middleNode1, endNode );

createRelationshipsBetween( startNode, middleNode2, middleNode3, endNode );

private void createRelationshipsBetween( final Node... nodes )

{

for ( int i = 0; i < nodes.length - 1; i++ )

{

nodes[i].createRelationshipTo( nodes[i+1], ExampleTypes.MY\_TYPE );

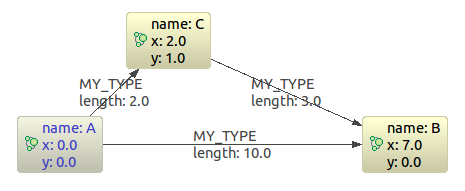
}

}

PathFinder<Path> finder = GraphAlgoFactory.shortestPath(

Traversal.expanderForTypes( ExampleTypes.MY\_TYPE, Direction.OUTGOING ), 15 );

Iterable<Path> paths = finder.findAllPaths( startNode, endNode );

[](http://docs.neo4j.org/chunked/snapshot/images/graphalgo-astar.png)

|  |
| --- |
| Node nodeA = createNode( "name", "A", "x", 0d, "y", 0d );  Node nodeB = createNode( "name", "B", "x", 7d, "y", 0d );  Node nodeC = createNode( "name", "C", "x", 2d, "y", 1d );  Relationship relAB = createRelationship( nodeA, nodeC, "length", 2d );  Relationship relBC = createRelationship( nodeC, nodeB, "length", 3d );  Relationship relAC = createRelationship( nodeA, nodeB, "length", 10d );  EstimateEvaluator<Double> estimateEvaluator = new EstimateEvaluator<Double>()  {  public Double getCost( final Node node, final Node goal )  {  double dx = (Double) node.getProperty( "x" ) - (Double) goal.getProperty( "x" );  double dy = (Double) node.getProperty( "y" ) - (Double) goal.getProperty( "y" );  double result = Math.sqrt( Math.pow( dx, 2 ) + Math.pow( dy, 2 ) );  return result;  }  };  PathFinder<WeightedPath> astar = GraphAlgoFactory.aStar(  Traversal.expanderForAllTypes(),  CommonEvaluators.doubleCostEvaluator( "length" ), estimateEvaluator );  WeightedPath path = astar.findSinglePath( nodeA, nodeB ); |

# MITSIMLab

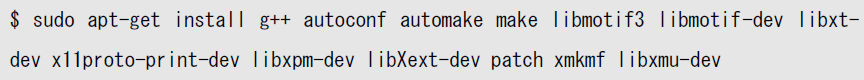
## Ubuntu安装（未成功）

MITSIMLab只能基于Linux，Redhat 7.3版本编译，可以跑在新版本Linux上，包括Fedora Core和Ubuntu。

### 安装前准备工作

参考官网提供的安装手册Install and Compile\_10\_v3.pdf。

1. 安装GNOME： sudo apt-get install ubuntu-desktop
2. 安装PVM： System->Administration->Synaptic Package Manager 输入pvm和libpvm3，应用Apply
3. $gcc –version，适用于4.4.1版本，新一点的版本貌似也无影响
4. 安装软件包:



1. 安装libstdc++-libc6.2-2.so.3，原下载地址已经无效，如果不安装，在启动MITSIMLab时会报错，网上很多解决方法也都过时。

libstdc++-libc6.2-2.so.3 属于 libstdc++2.10-glibc2.2 包，将提供libstdc++2.10-glibc2.2的源加入sources.list。

用gedit打开/etc/apt/sources.list，在最下面添加

deb <http://archive.debian.org/debian> woody main

deb-src <http://archive.debian.org/debian> woody main

$ sudo apt-get update

$ sudo apt-get install libstdc++2.10-glibc2.2

其他解决方法：<http://rpmfind.net/>搜索compat-libstdc++。链接地址：

<ftp://rpmfind.net/linux/redhat-archive/9/en/os/i386/RedHat/RPMS/compat-libstdc++-7.3-2.96.118.i386.rpm>下载下来是个rpm安装文件

### 安装MITSIMLab

1. 从http://sourceforge.net/projects/mitsim/下载MITSIMLab.tar.gz，直接在google搜MITSIMLab.tar.gz下载

$ tar zxvf MITSIMLab.tar.gz

$ cd MITSIMLab/

1. MITSIMLab下的目录组成：
2. bin：有ad和Linux两个目录，包含可执行程序；
3. GUI\_libs.tar：GUI\_Libs压缩文件；
4. pvmprofile
5. .pvm\_hosts
6. view：/data/brunnsviken目录下包含输入文件的例子；
7. Manual.pdf：用户手册
8. $ tar zxvf GUI\_Libs.tar.gz

$ cd GUI\_Libs

1. $ sudo –s输入密码

$ sudo –s ./Install

如果报”! You have to be root to run this script”错，则$ sudo gedit ./Install，将#!/bin/sh改为#!/bin/bash

### 配置MITSIMLab

1. 设置动态链接库。

$ sudo gedit /etc/ld.so.conf，在最后添加/usr/local/lib，$ sudo /sbin/ldconfig $ exit

1. 设置环境变量。

$ gedit ~/.bashrc，根据MITSIM解压位置，在最后添加

export XENVIRONMENT=~/software/MITSIMLab/bin/ad/xmitsim.ad

export PATH=$PATH:~/software/MITSIMLab/bin/Linux

export SIMLAB\_Linux=~/software/MITSIMLab/bin/Linux

export SIMLAB\_DAT=~/software/MITSIMLab/view/data/brunnsviken

$ source .bashrc

1. PVM负责MITSIMLab和TMS的通信，将pvm\_hosts文件从MITSIMLab下拷贝到家目录下

$ cp ~/software/MITSIMLab/.pvm\_hosts ~/

在最后添加一行，主机名通过/etc/hostname查看，默认为ubuntu

<主机名> ep=$SIMLAB\_Linux wd=$SIMLAB\_DAT

### 运行MITSIMLab

1. 修改MITSIMLab/view/data/brunnsviken目录下的master.mitsim，master.tms和master.smc文件

[Default Parameter Directary] ="~/software/MITSIMLab/view/data/Common"

[Input Directory] = "~/software /MITSIMLab/view/data/brunnsviken"

[Output Direct] = "~/software /MITSIMLab/view/data/brunnsviken/Output"

[Working Directory] = "~/software /MITSIMLab/view/data/brunnsviken/Output"

1. 删除原来存在的pvm文件。$ rm -fr /tmp/pvm\*

运行pvm。 $ pvm ~/.pvm\_hosts

确定主机服务器被添加到pvm。pvm>conf

让pvm在后台上运行。pvm>quit

1. xmitsim –m master.mitsim(带图形界面)

mitsim –m master.mitsim(不带图形界面)

**问题一：**

Error: can't open display "0.0"或xhost unable to open display 0.0

root下执行xhost +

## CentOS安装（成功）

### 安装前准备工作

**主要问题：**

error while loading shared libraries: libXm.so.3: cannot open shared object file: No such file or directory没有安装图形界面

$ yum install libXp-devel

$ yum install xorg-x11-xbitmaps

$ yum install openmotif

$ yum install pvm

root到/usr/lib下查看是否有libXm.so.4

$ ln –s /usr/lib/libXm.so.4 /usr/lib.libXm.so.3

### 安装MITSIMLab

1. 从http://sourceforge.net/projects/mitsim/下载MITSIMLab.tar.gz，直接在google搜MITSIMLab.tar.gz下载

$ tar zxvf MITSIMLab.tar.gz

$ cd MITSIMLab/

1. MITSIMLab下的目录组成：
2. bin：有ad和Linux两个目录，包含可执行程序；
3. GUI\_libs.tar：GUI\_Libs压缩文件；
4. pvmprofile
5. .pvm\_hosts
6. view：/data/brunnsviken目录下包含输入文件的例子；
7. Manual.pdf：用户手册
8. $ tar zxvf GUI\_Libs.tar.gz

$ cd GUI\_Libs

1. 切换到root身份执行

$ ./Install

注意是否报错，一般都因未安装openmotif-2.3.0.tar.gz

安装openmotif-2.3.0.tar.gz过程：

# tar -xzvf openmotif-2.3.0.tar.gz

# cd openmotif-2.3.0

# vim INSTALL.configure

# ./configure

# make

# make check

# make install

安装过程中出现“X11/extensions/Print.h: No such file or directory”的解决方法

这是因为系统中缺少libXp-devel $ yum install libXp-devel

安装过程中出现“X11/bitmaps/gray: No such file or directory”的解决方法

这是因为系统中缺少xorg-x11-xbitmaps（x11/xbitmaps (Ubuntu) and xorg-x11-xbitmaps (Fedora Core 5).）$ yum install xorg-x11-xbitmaps

### 配置MITSIMLab

1. 设置动态链接库。

$ vi /etc/ld.so.conf，在最后添加/usr/local/lib，$ ldconfig

1. 设置环境变量。

$ vi ~/.bashrc，根据MITSIM解压位置，在最后添加

export XENVIRONMENT=~/software/MITSIMLab/bin/ad/xmitsim.ad

export PATH=$PATH:~/software/MITSIMLab/bin/Linux

export SIMLAB\_Linux=~/software/MITSIMLab/bin/Linux

export SIMLAB\_DAT=~/software/MITSIMLab/view/data/brunnsviken

$ source .bashrc

1. PVM负责MITSIMLab和TMS的通信，将pvm\_hosts文件从MITSIMLab下拷贝到家目录下

$ cp ~/software/MITSIMLab/.pvm\_hosts ~/

设置临时主机名和查看主机名

$ hostname centos $ hostname

设置永久主机名

1. $ vi /etc/hosts

将127.0.0.1 localhost.localdomain localhost改为127.0.0.1 centos localhost

1. $ vi /etc/sysconfig/network

将HOSTNAME=localhost.localdomain改为HOSTNAME=centos

在最后添加一行，默认为centos

<主机名即centos> ep=$SIMLAB\_Linux wd=$SIMLAB\_DAT

### 运行MITSIMLab

1. 修改MITSIMLab/view/data/brunnsviken目录下的master.mitsim，master.tms和master.smc文件

[Default Parameter Directary] ="~/software/MITSIMLab/view/data/Common"

[Input Directory] = "~/software /MITSIMLab/view/data/brunnsviken"

[Output Direct] = "~/software /MITSIMLab/view/data/brunnsviken/Output"

[Working Directory] = "~/software /MITSIMLab/view/data/brunnsviken/Output"

1. xmitsim –m master.mitsim(带图形界面)

mitsim –m master.mitsim(不带图形界面)

## 文件类型

都是ASCII类型文件。

MITSIMLab需要1）监控设备的数据，包括控制设备的控制逻辑；2）待评估系统的执行环境。

MITSIMLab的输入文件包含以下几类：

### 输入文件

#### master.mitsim

定义整体仿真参数，包括1）输入输出目录；2）输入文件名称；3）输出文件名称和格式；4）仿真开始时间和步长；5）输出参数；6）仿真参数。

**/\***

**\* MITSIMLab master file**

**\*/**

**[Title] = "Modified Central Artery Network"**

**[Default Parameter Directory] = "/export/home/mit/DTA-Mitsim/data/"**

**[Input Directory] = "/export/home/mit/DTA-Mitsim/data/"**

**[Output Directory] = "/export/home/mit/DTA-Mitsim/data/Output/"**

**[Working Directory] = "/tmp/"**

输入文件为参数文件paralib.dat，网络文件networkrun.dat，旅程表demand.dat

**[Parameter File] = "paralib.dat"**

**[Network Database File] = "networkrun.dat"**

**[Trip Table File] = "/export/home/mit/DTA-Mitsim/data/demand.dat"**

**[Vehicle Table File] = ""**

**[State Dump File] = ""**

**[GDS Files] = {**

**% Filename MinScale MaxScale**

**}**

与最短路径计算相关，**Time variant path calculation**指根据时间而不是平均值计算最短路径，**Calculate shortest path periodically**对于未有预定路线的驾驶员定期重新计算最短路径，**Update path table travel time periodically**每辆车强制更新路径行驶时间，包括预先设定路线的车辆，**Use existing (cached) shortest path table** 使用存储的最短路径树，只在大多数车辆预先设定行驶路线时使用，**Updated travel time used for pre-trip plan**与引导车相关，如果设置，则引导出根据当前更新的行驶时间做决定，如果没设，引导车只在开始驾驶后使用当前更新的行驶时间。

0x109 # SP flags根据选项以十六进制相加组成，即包含{**Time variant path calculation, Use existing (cached) shortest path table,Updated travel time used for pre-trip plan**}。

**[Link Travel Times Input File] = {**

**"linktime.dat" # Historical travel time**

**"update.dat" # Updated travel time**

0x109 # SP flags

**% 0x001 Time variant path calculation**

**% 0x002 Calculate shortest path periodically**

**% 0x004 Update path table travel time periodically**

**% 0x008 Use existing (cached) shortest path table**

**% 0x100 Updated travel time used for pre-trip plan**

**}**

MITSIMLab的输入文件，Incident File包含意外的时间和严重程度，path.dat包含起始点和终止点的可能路径。

**[Incident File] = "incident.dat"**

**[Path Table File] = "path.dat"**

其他输出文件：

**[MOE Specification File] = ""**

**[MOE Output File] = "moe.out"**

**[Network State Tag] = "i3dm"**

**[Segment Statistics File] = "segstats.out"**

**[Segment Travel Times File] = "segtime.out"**

**[LinkFlowTravelTimes Output File] = "lft\_i.out"**

**[Link Travel Times Output File] = "linktime\_mitsim.out"**

**[Vehicle File] = "vehicle.out"**

**[Vehicle Trajectory File] = "trajectory.out"**

**[Vehicle Path Record File] = "pathrec.out"**

**[Departure Record File] = "dep.out"**

**[Queue File] = "queue.out"**

**[Point Sensor File] = "sensor.out"**

**[VRC Sensor File] = "vrc.out"**

起始和终止时间，步长：

**[Start Time] = 07:00:00**

**[Stop Time] = 08:12:00**

**[Step Size] = 0.2**

以下是各种步长，最重要的是Segment Data Sampling Step Size（定义仿真数据的采样频率，即向segstats.out输出的频率），Segment Data Report Step Size（每多少秒计算仿真数据的平均值），Point Sensor Step Size（定义传感器报告采集到的车辆数据的频率），这三个都以秒为单位。Sensor Output Flags定义是否生成输出文件。

**[Segment Data Sampling Step Size] = 30**

**[Segment Data Report Step Size] = 300**

**[Point Sensor Step Size] = 120**

**[Point Sensor Step Sizes] = { 1 30 }**

**[Sensor Output Flags] = { 0 1 }**

**[Area Sensor Step Size] = 60**

**[Animation Step Size] = 0.1**

**[Segment Color Step Size] = 15**

**[Console Message Step Size] = 60**

MOE Step Size定义数据收集和平均的时间间隔，MOE OD Pairs里的OD对指明哪些OD对需要报告最大速度和距离。

**[MOE Step Size] = 600**

**[MOE OD Pairs] = {**

**}**

指明需要输入的文件有哪些，以16进制相加。

**[Output] = 0x13357**

% 0x00001 = Vehicle log

**% 0x00002 = Sensor readings**

**% 0x00004 = VRC readings**

**% 0x00010 = Link travel times**

**% 0x00020 = Segment travel times**

**% 0x00040 = Segment statistics**

**% 0x00080 = Queue statistics**

**% 0x00100 = Travel time tables**

**% 0x00200 = Vehicle path records**

**% 0x00400 = Vehicle departure record**

**% 0x00800 = Vehicle trajectories**

**% 0x01000 = Output rectangular text**

**% 0x02000 = No comments**

**% 0x10000 = State 3D**

下面是设置MITSIMLab显示网络元素和车辆的参数。

**[Segments] = 3**

**% 0 = Link type**

**% 1 = Density**

**%** 2 = Speed

**% 3 = Flow**

**[Signals] = 0x0**

**% 0x01 = Traffic signals**

**% 0x02 = Portal signals**

**% 0x04 = Variable speed limit signs**

**% 0x08 = Variable message signs**

**% 0x10 = Lane use signs**

**% 0x20 = Ramp meters**

**[Sensor Types] = 0x0**

**% 0x1 = Loop detectors**

**% 0x2 = VRC sensors**

**% 0x4 = Area sensors**

**[Sensor Color Code] = 3**

**% 0 = Count**

**% 1 = Flow**

**% 2 = Speed**

**% 3 = Occupancy**

**[Vehicles] = 4**

**% 0 = None**

**% 1 = Vehicle type**

**% 2 = Information availability**

**% 3 = Turning movement**

**% 4 = Driver behavior group**

**% 5 = Lane use**

**[Vehicle Shade Params] = {**

**0 # Shade**

**86400 # Outstanding time in a segment**

**86400 # Outstanding time in the network**

**}**

通用参数，Verbose控制终端的文本消息的显示，1显示，0跳过。Nice控制处理器资源配置的优先权，0最高，1允许其他进程并行运行。

**# [Verbose] = 1**

**# [Nice] = 1**

#### master.tms

包含控制车辆运行所需的信息，如信号时长，信号控制逻辑和参数的输入文件。

**/\***

**\* TMS master file**

**\*/**

**[Title] = "Modified Central Artery Network"**

**[Default Parameter Directory] = "/export/home/mit/Mitsim/data/"**

**[Input Directory] = "/export/home/mit/Mitsim/data/"**

**[Output Directory] = "/export/home/mit/Mitsim/data/Output/"**

**[Working Directory] = "/tmp/"**

网络文件为networkrun.dat，控制参数文件为ctrlpara.data，控制逻辑文件为ctrllogic.dat，其他与交通控制相关参数。

**[Network Database File] = "networkrun.dat"**

**[GDS Files] = {**

**% Filename MinScale MaxScale**

**}**

**[Parameter File] = "ctrlpara.dat"**

**[Control Logic File] = "ctrllogic.dat"**

不同信号的控制逻辑，目前保存在ctrllogic.dat中。

**[Signal Plan File] = ""**

**[Control Logic] = 0**

**% 0 = None**

**% 1 = A1 incident response**

**% 2 = Gating logic**

定义运行模式，0根据历史数据仿真，1根据即时数据仿真，2根据预测数据仿真。

**[Information] = 2**

**% 0 = Historical data**

**% 1 = Real time measurement**

**% 2 = Prediction**

定义仿真时间参数，起始和结束时间，步长。

**[Start Time] = 07:00:00**

**[Stop Time] = 08:12:00**

**[Step Size] = 0.1**

与ATIS的设计相关的参数，诱导生成，驾驶员行为传播。Rolling Step Size定义ATIS生成诱导的频率。Rolling Length定义预报步长。NumOfDTAIterations指明每次诱导生成期间ATIS迭代次数。DTAComputationalDelay指明诱导生成后对驾驶员产生影响的时间延迟。

**[RollingStepSize] = 600**

**[RollingLength] = 1800**

**[NumOfDTAIterations] = 2**

**[DTAComputationalDelay] = 110**

以下定义路网界面显示参数。

**[Console Message Step Size] = 60**

**[Segments] = 3**

**% 0 = Direction**

**% 1 = Link type**

**% 2 = Density**

**% 3 = Speed**

**% 4 = Flow**

**[Signals] = 0x0**

**% 0x01 = Traffic signals**

**% 0x02 = Portal signals**

**% 0x04 = Variable speed limit signs**

**% 0x08 = Variable message signs**

**% 0x10 = Lane use signs**

**% 0x20 = Ramp meters**

**[Sensor Types] = 0x0**

**% 0x1 = Loop detectors**

**% 0x2 = AVI sensors**

**% 0x4 = VRC sensors**

**% 0x8 = Area sensors**

**[Sensor Types] = 0x0**

**% 0x1 = Loop detectors**

**% 0x2 = AVI sensors**

**% 0x4 = VRC sensors**

**% 0x8 = Area sensors**

**[Sensor Color Code] = 3**

**% 0 = Count**

**% 1 = Flow**

**% 2 = Speed**

**% 3 = Occupancy**

**# [Randomize] = 0**

**# [Verbose] = 1**

**# [Nice] = 1**

#### master.smc

控制所有组件的整体运行，链接master.mitsim和master.tms文件。定义输入输出目录，主机设置和显示设置。

**# Simlab master file**

**[Title] = "Central Artery Network"**

**[Input Directory] = "/export/home/mit/Mitsim/data/"**

**[Output Directory] = "/export/home/mit/Mitsim/data/output/"**

**[MITSIMLab] = {**

**"master.mitsim" # master file**

**"dtaev" # host**

**"dtaev.sdi.utk.edu:0.0" # display**

**}**

**[TMS] = {**

**"master.tms" # master file**

**"dtaev" # host**

**"" # display**

**}**

**[MesoTS] = {**

**"master.meso" # master file**

**"dtaev" # host**

**"" # display**

**}**

**# [Misc] = {**

**# "master.mdi" # master file**

**# "dtaev" # host**

**# "" # display**

**# }**

**# [Verbose] = 1**

**# [Nice] = 1**

#### Paralib.dat

定义仿真模型参数。

第一部分包含限制参数和GUI表格设置。

**/\***

**\* This file contains the parameters used by MITSIMLab**

**\*/**

**# Following parameters transfer units in the British system into**

**# meteric system.**

**[Native Length to Meter] = 0.3048**

**[Native Speed to Meters per Second] = 0.4470**

**[Native Density to Vehicles per Kilometer] = 0.6214**

**[Native Flow to Vehicles per Hour] = 1.0000**

**[Native Travel Time to Minute] = 1.0000**

**[Native Demand to Vehicles per Hour] = 1.0000**

**# LEGEND UNITS**

**[Density Label] = "Density (vpm)"**

**[Speed Label] = "Speed (mph)"**

**[Flow Label] = "Flow (vph)"**

**[Occupancy Label] = "Occupancy (%)"**

**# THRESHOLDS FOR VIEWING GRAPHICAL OBJECTS (optional)**

**# Show the objects when pixels per meter is greater than the**

**# value provided here.**

**[Resolution] = {**

**1 # lanes marks**

**2 # lane type, vehicles, sensors and signals**

**6 # vehicle lane change indicators**

**10 # vehicle labels**

**}**

**[FontSizes] = {**

**100 120 140 160 180**

**}**

下面定义路径选择模型的参数。模型基于C-Logit，MITSIM使用以下参数在路径选择模型中，行驶时间（一般采用最短路径的行驶时间），共用因子（以纠正路径有重叠时违反IIA属性的错误），高速路偏差和违规惩罚。车辆可能有或无诱导，或既定和动态计算的路径。

驾驶员选择某条路径的可能性取决于逻辑选择模型，驾驶员在多条路径中选择哪条取决于行驶时间，其基于历史数据或实时数据，和是否有诱导。

违规惩罚加入相应路径的行驶时间。参数Fraction定义有诱导和无诱导车辆的分数。 是有无诱导车辆的行驶时间系数。

第一列是每种类型车辆的比例，第二列是所选路径的行驶时间的系数。

**[Route Choice] = {**

**% Fraction Beta**

**1.0 -5.0**

**0.0 -5.0**

**}**

当两条路径有共同部分时，驾驶员不该考虑选择哪条。Commanalit Factor是应用到路径上的惩罚系数，以纠正该错误。

**[Commonality Factor] = -1.0**

驾驶员喜欢行驶在入口少的道路，如高速路。因此在连接时间和最短路径计算中应用高速路因子。因子应大于等于1.

**[Freeway Bias] = 1.0 # Travel time factor**

路径生成模型中，没有预先制定路径的车辆在每个交叉口都要生成路径，因此在连接处离开需应用改变惩罚系数。路径交换模型中，设定路径的车辆可能在中间点处换成其他路径，因此在更改原定路径时应用改变惩罚系数。

**[Diversion Penalty] = {**

**5.0 # in route generation model (minutes)**

**1.0 # in route switching model (minutes)**

**}**

驾驶员不会考虑所有可行路径。如果行驶时间大于最短路径行驶时间乘以Valid Path Factor的路径，驾驶员将不会考虑。

**[Valid Path Factor] = 3.0 # compare to shortest path**

没有预设路径的驾驶员根据基本路径选择模型选择下一条路。RLF（rational link factor）定义哪些与终点相连的下游连接会考虑，以避免重复，循环路径。0最少限制，所有与终点相连的连接都考虑；1最多限制，只有让车辆离终点更近的连接才考虑。RLF小于5相当于RLF=0。

**[Rational Link Factor] = 0.5**

将新和旧行驶时间组合更新

Updated = (1-r)\*old + r\*new where r=[Path Alpha]

**[Path Alpha] = 0.75**

控制驾驶员对新交通环境的反应时间和车辆状态，如加速，减速，保持速度或停止。

**[Update Step Sizes] = { # normal distributions (seconds)**

**% mean stdev lower upper**

**0.5 0.0 0.5 0.5 # decelerating**

**1.0 0.0 1.0 1.0 # accelerating**

**1.0 0.0 1.0 1.0 # uniform speed**

**0.5 0.0 0.5 0.5 # stopped vehicle**

**}**

25页

1. Ctrlpara.dat

交通管理仿真相关的仿真参数。

1. Ctrllogic.dat

包含意外事故应急相关的参数定义。

1. Network Database

网络定义，包含点，连线，车道等网络组成。

1. Demand file

不同时间下的OD点对。

1. Path File

OD对的路径。

### 输出文件

1. Moe.out

包含最大车头速，最大尾速，内车道间隙。

1. Segstats.out

仿真各段数据，包含仿真时间间隔，每段进入和离开的车数，平均密度，速度和行驶时间，每条车道的速度和密度。

1. Segtime.out

每段流量和行驶时间。

1. Lft\_i.out

连接处的流量和行驶时间。

1. Linktime.out

连接处的行驶时间。

1. Vehicle.out

车辆起始地，目的地，出发和到达时间，里程，速度。

1. Trajectory.out

包含所有车辆的位置，用于获得车辆行驶路径。

1. Pathrec.out

包含连接处的进入和离开数据。

1. Dep.out

包含车辆的出发记录，由出发时间，车辆ID，起始点，目的地，车辆类型和路径组成。

1. Queue.out

仿真时获得的排队长度。

1. Sensor.out

传感器获得的交通量。

1. Vrc.out

点对点传感器记录下的车辆信息，每条记录与单独车辆相应。

### 公交过境系统

公交过境系统还需包含以下输入文件：

1. Route.dat

定义网络构成，如公交站点，路线等。

1. Schedule.dat

定义个人路线，设置特定路线和达到公交站点时间。

1. Run.dat

指定在Schedule.dat中分配的路线上的公交行驶次序。

1. Bus.dat

指定在Run.dat中分配的公交，和公交类型，出发时间（即进入网络提供服务的时间）。

1. Passenger.dat

指明乘客达到率和在每个站点间的公交载客率。

1. Conditional.dat

包含优先策略的条件阈值。

公交过境系统还可包含以下输出文件：

1. Transtrj.out

包含公交ID，路线ID，旅程ID，总计划偏差，当前时刻连续公交间的车头时距，每时间步长下的载客量。

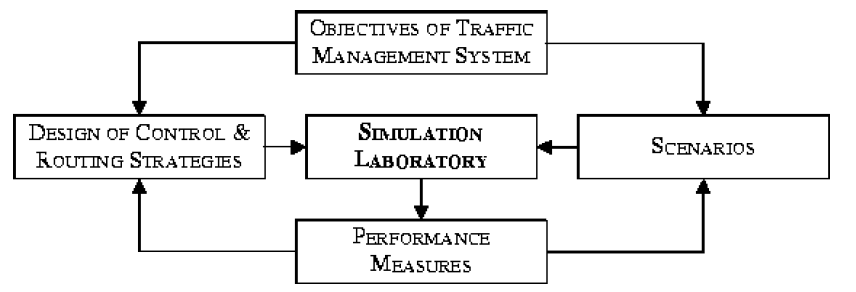
1. Busstop.out

包含公交达到每个站点的时间，包括公交站点ID，公交ID，路线ID，与计划偏差，当前公交与前一公交的车头时距，靠站时间，上车乘客数，车上乘客数，下车乘客数。

## MITSIMLab模型

### 整体框架

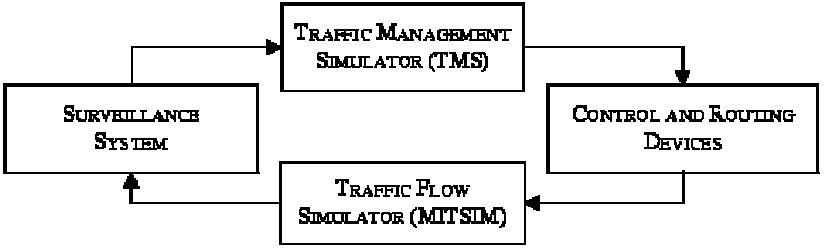
MITSIMLab本身资料很少零散，用户手册未教如何使用，开发文档无。



整体设计和评价框架

MITSIMLab用于提供实时遥感数据模拟ITS交通管制，接受交通信号控制并在模拟网络中显示，计算效率。

MITSIMLab由TMS和MITSIM两部分组成，TMS模拟路径诱导和信号控制，由MITSIM模拟并检测其效率。MITSIM模拟单个车辆在路网上的行为和驾驶员对路径诱导和信号控制的反应。



MITSIMLab的路径选择模型基于C-Logit。

### MITSIM

MITSIM尽可能精确的模拟交通流，以获得车辆对实时控制和路径选择策略的反应，由监控系统收集信息提供给TMS。

1. Network representation

**Nodes：**代表交叉口或OD点，由类型(type，intersection|origin/destination)，ID，名称(name)组成。

**Links：**多个部分组成的路段，由类型（type，freeway|ramp|urban street|tunnel），ID，起始点，终止点，包含部分组成。

**Segments**：

车道数，等级，曲率，设计速度等地理特征所描述的路段部分，由速度极限，设计速度，等级，几何形状，ID，包含车道组成。

**Lanes**：

最低等级元素，从两方面描述每条车道：ID，车道变换和车道使用规则，车道变换规则决定哪条车道允许变换，车道使用规则决定什么类型车辆运行使用车道。

**Lane-connections**：

车道连接处，每个交叉口连接处显式编码，Segments由车道连接处连接。

1. Traffic surveillance & control

**Surveillance sensors**：

监控器提取交通流，纵向位置相同的传感器组成同一传感器站，传感器站由传感器类型，传感器任务，检测地区长度，纵向位置和包含的探测器组成。各类传感器误差由TMS的监控系统模块参数决定。

不同类型的传感器包括：

1. 单点传感器提取车辆数，占用，速度，前进点，如环形探测器（loop detector）；
2. 单点传感器提取单个车辆信息，如车辆类型，载重；
3. 点对点传感器提取行驶时间，如探测车（probe vechicles）；
4. 区域传感器如雷达探测器，摄像头等。

**Control devices：**

控制设备包括交叉口控制，匝道控制，主干道控制（e.g. Lane Use Signals (LUS), Variable Speed Limit Signs(VSLS), Variable Message Signs (VMS),portal signals at tunnel entrances），路径诱导设备。

控制设备范围可以是道路范围或车道范围，道路范围如VSLS，VMS控制所有车辆，LUS只控制特定车道交通。控制设备由位置，类型，初始状态，是否可见组成。

**Toll plazas：**

收费站由许多收费亭组成，由是否可见，各收费亭编码，各收费亭位置，每个收费亭参数组成。收费亭参数包括车道使用优先编码，默认状态（free，open，closed），速度限制，收费延迟。车道使用优先编码决定哪种类型车辆允许使用收费亭（如ETC），延迟设为负指数分布。

1. Travel demand

MITSIM的输入为基于时间的OD表，高峰期时间间隔段，平峰期长。车辆类型包括：车辆等级（高级轿车，低级轿车，公交，卡车，拖车），车道使用优先（HOV和ETC），有无诱导驾驶。每对OD有离开率和标准方程，0-1的泊松分布系数决定多少比例的车辆确定离开，代表拥堵程度。

1. Vehicle characteristics

**Maximum acceleration rate：**

最大加速比根据道路等级和车辆速度方程计算，默认值基于FREsim使用手册[FHWA 94][FHWA 80]。

**Normal deceleration rate**

正常减速比是在无紧急情况下的平缓减速和计算停车距离，默认值基于[Institute of Transport Engineers 82]。

**Desired speed**

理想速度很少使用，因为驾驶员看到限速牌或进入有限速的路段时会更新。更新规则基于车辆最大速度，理想速度，路段限速。

**Speed distribution across lanes**

每条车道根据它的设计速度和路段数有最大速度，默认值基于[FWHA 80]。

**Target speed**

目标速度是驾驶员试图获得的速度，由驾驶员理想速度和车道最大速度决定，如果后者限制了目标速度，则驾驶员考虑换道。

1. Vehicle routing

行驶时间有2种：历史和实时时间。实时时间周期性更新并传给TMS，更新信息传到有诱导的车辆，车辆根据其选择剩余路径。车辆也根据VMS提示以一定比率反应。路径选择模型获得驾驶员决定和对交通信息的反应。

1. Vehicle movements (car-following, lane changing)

车辆行为由与其他车辆的交互，网络对信号控制的反应，驾驶员特点如理想速度和车道使用偏好等共同决定，最终形成加速，减速，变道等行为，包括跟驰，变道和事件反应三种微观行为。

**Car following**

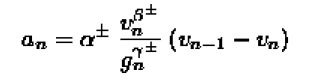
跟驰模型包括合适的间距，前车速度，当几条车道合并时将出现的情况或其他车道加插等。

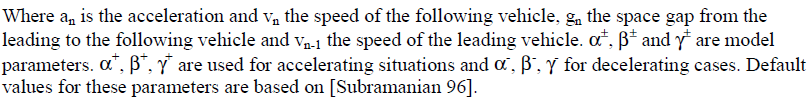
车辆行为有3种：自由行驶，跟驰，紧急刹车。

如果前方距离高于预设阈值hupper，车辆无需与前车发生交互则自由行驶。当速度较低时，车辆将根据最大加速比达到目标速度；当速度超过目标速度则根据正常减速比达到目标速度。

如果前方距离低于预设阈值hlower，车辆紧急刹车，采用合适减速比避免撞车直到前方距离大于hlower。

如果前路低于预设阈值hupper，高于预设阈值hlower，车辆跟驰，加速比根据Herman的通用跟驰模型计算。





**Merging**

当多条上游车道合并为一条时，车辆需作出不同反应，包括2种并道。优先并道包括从坡道到高速路或从辅路到主路。非优先并道是下游收费站或车道减少（如3条并2条）。优先并道中，车辆需要等待一个可接受的主路车间隔，然后加速合并或减速让路。非优先并道中，意味着减小跟驰距离，可能2辆车在合并区域共用同一车道。结果是在合并时车辆几乎紧贴另一辆，然后突然以最大加速比加速离开合并区域。

**Event responding**

驾驶员需对其他驾驶员行为，上游事件如交通信号和标志，下条道路连接预期，其他车辆转入相同车道等作出反应。

Traffic signals and signs：每个控制设备有一项参数定义多远距离内驾驶员能看到（基于地形和天气等），在视距内才计算车速和加速比。如果是减速或停止信号，则用所需的减速比，否则保持原速。红灯转绿灯时，启动延迟根据车辆在排队中的位置计算。以上每项交通规则有一定的服从比率，只有服从车辆执行这些规则。

Incidents：部分或全部车道堵塞，使得车辆减速到一定速度（一般是对面或临近车道）。当事故发生，车辆不能变道通过，而是利用正常刹车距离和合适的减速。

Anticipation of connection to a downstream link：当车辆需要变道继续行驶时，如果无法变道，车辆将减速以避免错过出口。

Courtesy yielding：避让，当驾驶员减速为其他进入自己车道的车辆让路。进入车辆将成为前车，采用合适减速获得理想的跟车距离。

**Lane changing**

变道分为三步：1检测车道变换是否必要；2选择合适车道；3选择可接受车距，执行变道。

1变道决定：有强制变道和酌情变道。强制变道基于这些原因：下条路径在不同车道，遇到事故，避免进入限制车道，对LUS和VMS作出反应。除此之外都是酌情变道。强制变道需根据到出口的距离和拥堵程度作出决定。酌情变道根据冲动因子，速度差别因子，当前和旁边车道的交通情况作出。

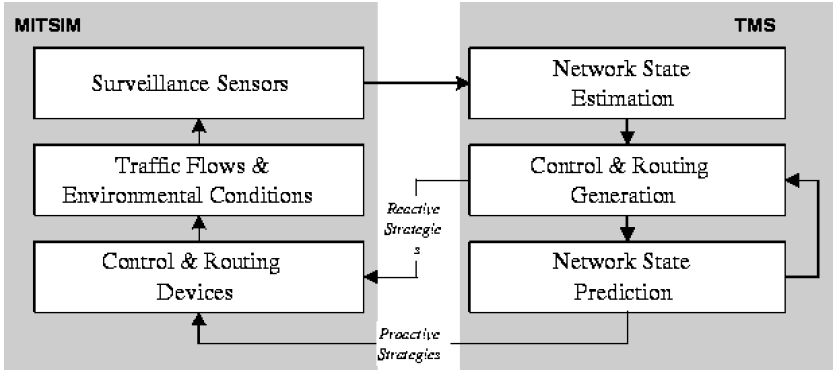
2车道选择：酌情变道根据交通情况，变道规则，车道使用权，车道方向等。强制变道只考虑下个连接。

3可接受车距：车辆检测目标车道的车距，当检测到足够距离则立刻变道。

**Nosing and yielding**

在拥堵交通中很难找到合适车距，强制变道的车辆会尽可能挤进目标车道（nosing），根据到出口距离，变道数和等待时间。被插入的车辆会自动避让。

### TMS



TMS分析交通情况（Network State Estimation），生成控制和路径信息（Control&Routing Generation），预测未来交通情况（Network State Prediction）。

TMS包括交叉口控制（信号，等待和停止信号TS），匝道控制（匝道合并，速度限制），主路控制（车道使用标志LUS，可变速度限制标志VSLS，可变信息标志VMS）几种设备，由static，pretimed，traffic adaptive和metering四种信号控制器。根据设定逻辑互相变换，例如高峰期用pretimed控制，平峰期用adaptive自适应控制。

## 界面开发

MITSIMLab界面利用Motif开发，

<http://blog.csdn.net/mathlover/article/details/416298>

图表显示使用SciPlot控件，2D或3D

数据显示使用XbaeMatrix，类似电子表格

界面显示使用Xmt

# Transims

## 概述

TRANSIMS（TRansportation ANalysis and SIMulation System）是基于元胞自动机微观区域交通分析工具集。基于虚拟人口和活动的模拟单个出行者和不同出行模式的新方法。

目前TRANSIMS可处理大于100,000道路和大于30,000,00个出行者。

* 模拟由调查数据获得的虚拟个人在24小时内的活动；
* 根据高细节的道路和网络，跟踪并分析个人每秒位置；
* 利用路径模块分析出行者24小时的道路延迟。

TRANSIMS输出：

* 细节快照数据：每个出行者的位置，车辆，过境车辆；
* 交通流数据：拥堵，交叉口排队等
* 道路延迟

## 安装使用

### Win7安装

安装TortoiseSVN，Transims4.0安装Visual Studio 2005 Profeesional，Transims5.0安装2010版本。

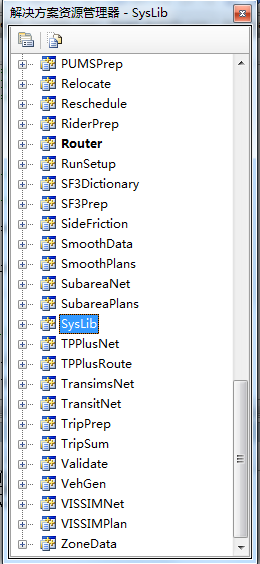
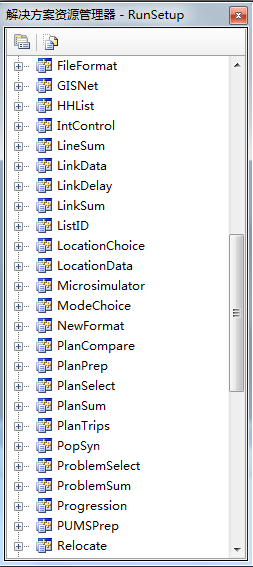
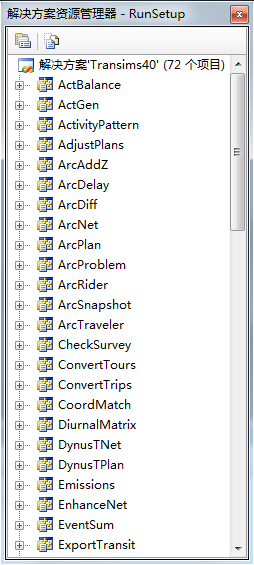
新建Transims40目录，进入目录右键弹出菜单中选择TortoiseSVN Checkout，

输入URL，<https://transims.svn.sourceforge.net/svnroot/transims/version4/trunk/src/Transims40>

单击OK，下载trunk目录下的综合开发版本，branch下面是分支测试版，tag是不同稳定版本。

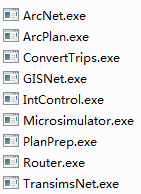
进入目录双击“Transim40.sln”，编译执行。

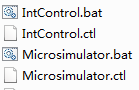
### 使用手册

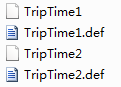
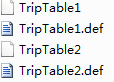


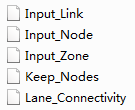
TRANSIMS并没有界面，而是由一系列可执行文件组成不同的流程，利用其它工具或软件显示结果数据。其中最重要的是SysLib，其中定义了TRANSIMS通用类。

每个数据集中有control，demand，network，results，bin几个目录。

Bin包含流程所需的可执行文件；

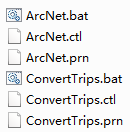
control包含控制文件和调用可执行文件的批处理文件；

demand包含出行表和昼夜分布曲线；

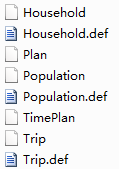
network包含网络和节点文件。

执行bat后

1. control内生成程序输出的prn文件；

bat文件内容为..\bin\ArcNet.exe ArcNet.ctl

1. Demand生成家庭数据和出行计划；



1. Network包含全部网络文件，其中生成arcview目录为矢量文件；
2. Results包含仿真输出

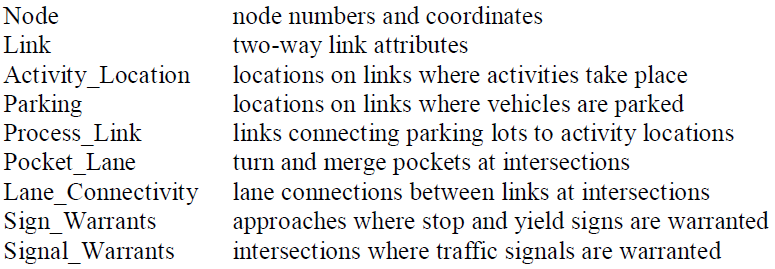
#### Case 1

1. **Network Conversion**

用TransimsNet.bat将network里的点和线转换为路网，提供TransimsNet.ctl。

控制文件由一系列键值对组成，一行一个，中间用tab键隔开，键大写，值小写，多个键值，则每个键后面加数值（如POCKET\_RANGE\_FOR\_FACILITY\_4），顺序随意。

输入为Input\_Link，Input\_Node，Input\_Zone，Keep\_Nodes；

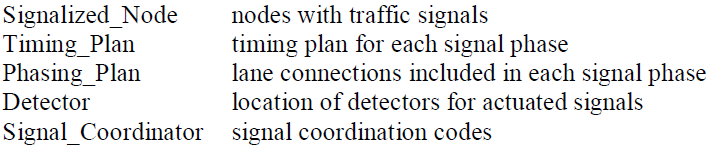
输出为

1. **Intersection Traffice Controls**

必须生成交叉口信号控制才算路网转换完成。利用TransimsNet生成的Sign\_Warrants和Signal\_Warrants，调用IntControl.bat生成信号控制，提供IntControl.ctl。

输入为Sign\_Warrants，Signal\_Warrants。

输出为



如果显示“Program Terminated Due to Errors”，可查看IntControl.prn，程序自动在Unsignalized\_Node文件中增加“no control”。

1. **ArcView Network Files**

用ArcNet.bat将TRANSIMS网络文件转换为ArcView矢量文件，供ArcView，ArcGIS显示。

结果在network/arcview下，包含二进制矢量（.shp）和索引（.shx）文件，dBase文件（.dbf），为每个dBase文件提供定义文件（.def），提供给子程序以文件的字段和数据类型信息。

1. **Trip Table Conversion**

Demand目录下有四个出行OD表（TripTable1）和四个出行时间分布表（TripTime1）。用ConvertTrip.bat将其转换为Trip，Household，Population，Vehicle。

1. **Travel Plans**

Router.bat根据Trip利用最近距离算法生成Plan文件，Plan指明活动时间和时长，在不同活动中的时间间隔。如果无法找到路径，则会写入results/Route\_Problem文件中。

1. **Plan Sorting**

Router建立家庭出行计划后，由于出行是随机指定出行时间，微观仿真会根据出发时间处理出行，因此需要对出行计划按照时间进行排序。

用PlanPrep.bat排序计划。

1. **Microsimulation**

利用元胞自动机按秒地仿真出行计划，跟踪个人和车辆运动情况，Microsimulation确定车辆冲突，执行信号周期，交通吞吐量和速度。

输出为Problem File，Problem Link Files，Traveler Files，Snapshot Files，Link Delay Summary Files，Performance Summary Files，Event Files，System Event Files。

1. **Batch Execution**

可执行NetPrep.bat和RunAll.bat完成全部操作，NetPrep.bat包含前三步，RunAll.bat包含后四步。

#### Case 2

1. **GIS Network Conversion**

GISNet.bat将ArcView矢量数据转换为TRANSIMS格式。

输出为Input\_Node，Input\_Link，Input\_Shape，是TransimsNet的输入。

提供GISNet.ctl文件，其中包括其他几种文件：

User Program Script

GISNet利用control/Script.txt将GIS文件转为TransimsNet所需数据。Script.txt使GIS文件中的字段与TRANSIMS link文件字段相对应。返回值1指明GISNet要保存link记录。

Link.ID = GIS.LINK

Link.NODEA = GIS.ANODE

Link.NODEB = GIS.BNODE

Link.PERMLANESA = GIS.LANES\_BA

Link.PERMLANESB = GIS.LANES\_AB

Link.TWOWAYTURN = "F"

Link.CAPACITYA = GIS.CAPACITY\_B

Link.CAPACITYB = GIS.CAPACITY\_A

Link.SPEEDLMTA = GIS.SPEED\_BA

Link.SPEEDLMTB = GIS.SPEED\_AB

Link.FREESPDA = GIS.SPEED\_BA

Link.FREESPDB = GIS.SPEED\_AB

Link.FUNCTCLASS = GIS.CLASS

Link.VEHICLE = GIS.LIMIT

Link.NOTES = GIS.NOTES

RETURN (1)

END

Coordinate Conversion

如果GIS矢量文件不是UTM坐标，则利用control文件的四个值将Washington DC的国家平面坐标转换为UTM坐标。ArcNet也有类似四个值将UTM坐标转回国家平面坐标。

1. **Create the TRANSIMS Network**

TransimsNet.bat生成其他路网信息。

1. **ArcView Network Files**

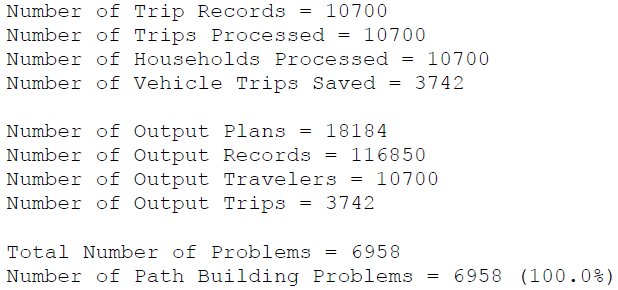
为确定转换是否精确，可将TRANSIMS矢量文件与原始矢量和遥感图片叠加，需要先为新矢量文件建立空间坐标系，ArcToolbox用于定义文件坐标系。

1. **Trip Table Conversion**

ConvertTrips.bat用于将出行转换为TRANSIMS的出行，家庭，人口，车辆文件。

1. **Travel Plans**

Router生成出行计划，但不一定成功建立所有计划，需查看Router.prn。



10700个出行，只有3742个成功，6958个由于建立路径原因失败，

1. **ArcView Plan Files**

用ArcPlan建立矢量记录从而确定问题路径。提供许多选择计划方法，如根据出行者，出行时间，连线或点顺序等。

查看ArcPlan图层的属性表，有三项记录：走路到停车场，两停车场间的开车，走路到目的地。

1. **Plan Sorting**

出行计划完善后，执行PlanPrep.bat生成TimePlan。

1. **Microsimulation**

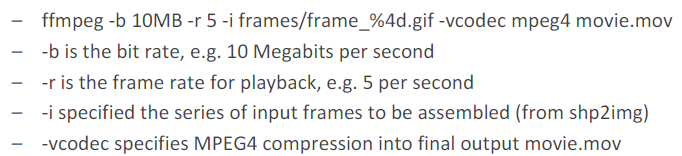
Microsimulation.bat检查输出结果，如果没有在trip conversion和routing时修正错误，则实际交通量将小于预期。

### 界面显示

ArcSnapshot生成用于显示动画的矢量文件。

UMN MapServer提供shp2img，根据ArcSnapshot生成的矢量文件显示动画。

Ffmpeg生成MPEG4的影片，用QuickTime播放



NEXTA，TRANSIMS仿真数据可视化工具

Metropolis，Python开发，3D导航功能

### 工具集

1. 路网转换

TransimsNet：生成符合TRANSIMS网络特征的路网；

IntControl：利用TransimsNet生成的warrants生成交通标志和详细交通信号；

EnhanceNet：为link和connectivity表增加方位和类型字段。

1. 输出GIS图层

ArcNet：生成高精度的GIS矢量文件，供ArcGIS可视化显示；

ArcPlan：Routing后显示个人计划和组计划；

ArcProblem：显示位置问题的类型，包括Zero Node Problem，Access Problem，Circuity Problem等；

ArcDelay：显示连线延迟，容量和时间间隔下的道路连接参数。

1. GIS路网编辑

GISNet：读取ArcGIS点和连线矢量文件，转换为标准TRANSIMS网络文件。

1. 出行转换

ConvertTrips：利用出行表和昼夜分布推算出个人出行；生成虚拟车辆，家庭和出行者；分配车辆类型至出行表；估计出行时间；生成出行记录。

SmoothData：读取昼夜分布文件平滑，避免时间段间出行量剧烈变化。

1. 分区

HHList：从大量家庭中随机抽取一系列家庭用于分区；

1. 子区仿真

SubareaNet：利用GIS矢量文件剪出一块子网，重写成新路网；

SubareaPlans：出行分析工具，决定是否达到子网；通过子网的出行被提取到新的计划文件中，记录进出子网地点作为起始和结束位置。

LinkDelay：合并子网和整个路网的道路延迟。

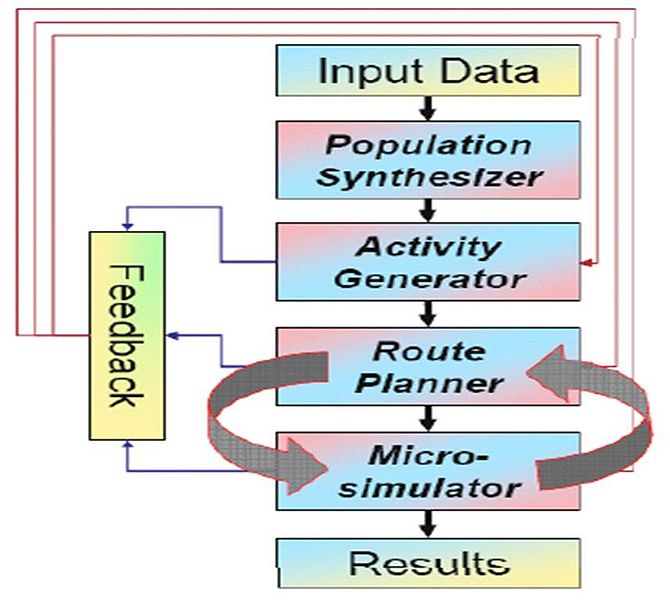
1. 反馈和评价

PlanSelect：根据出行时间，持续时间，V/C比，坐标，车辆类型，子区形状选择重新规划线路的家庭；

ProblemSelect：根据问题线路，问题类型选择重新规划线路的家庭；

PlanCompare：生成收敛数据和分布表；对比计划，选择不同出行时间的计划。

## 系统结构

[](http://upload.wikimedia.org/wikipedia/commons/7/77/Transims.jpg)

首先生成路网，获得运输计划，将交通载入网络，采用纳什均衡迭代。

然后有两种方法：1、根据人口普查数据，活动调查，执行Population Synthesizer和Activity Generator，生成Activities；2、根据出行表，昼夜分布，执行Trip Converter生成Trips。

接着利用Route Planner从Activities或Trips中获得出行计划，在Microsimulator中测试出行计划。

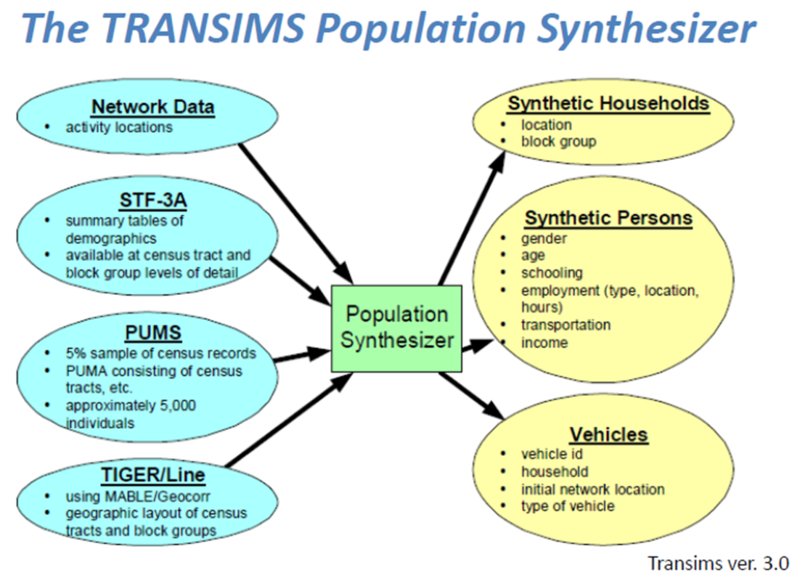
最后在Router和Microsimulator中反复反馈和迭代，直至达到某种目标。

### Input Data

* 交通网络（道路，交叉口，信号，过境线路和计划，土地利用数据，区域信息）
* 道路网和计划
* 运输网和计划
* 人口普查数据
* 家庭出行调查
* 流动出行者和出行
* 车辆特点和原型

道路网可从商业导航或城市规划局获得，必须拓扑正确（道路连续），运输网必须与道路网一致，公交跟随交通流动，因此仿真结果可能与公交时刻表不同。

### Population Synthesizer

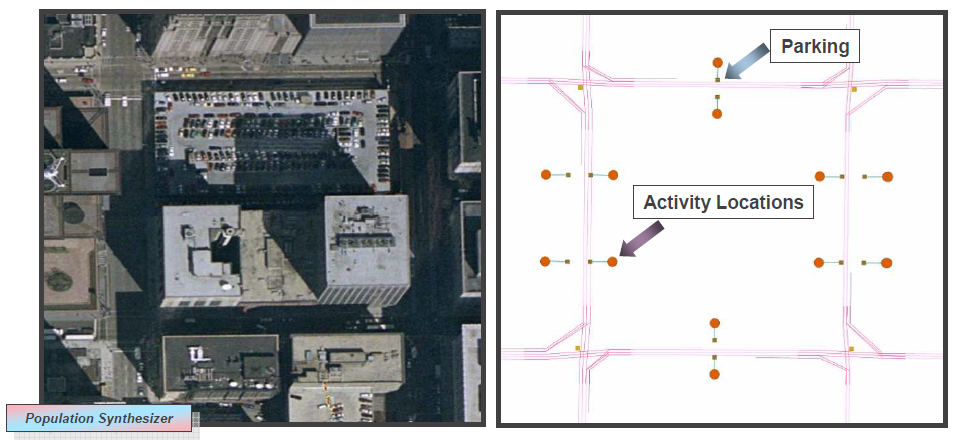
[](http://upload.wikimedia.org/wikipedia/commons/a/a3/Transims_population_synthesizer.png)

虚拟人口

* 人口特征与真实情况相符；
* 家庭分布与区域人口分布接近；
* 家庭位置决定出行的起始和结束点。

虚拟人口作用：

* 从人口普查生成虚拟家庭；
* 生成家庭人口特点（收入，成员等）；
* 每个家庭在网络中的接入位置（活动位置）；
* 车辆配置。



Population Synthesizer示例

美国人口普查数据分为两种：

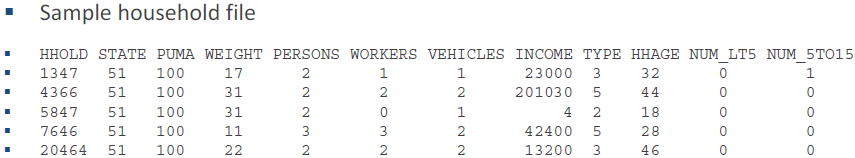
SF3是综合数据，相对较小区域，大概4000家庭。

PUMS是5%采样率的实际调查数据，覆盖大区域，大概1000,000家庭。

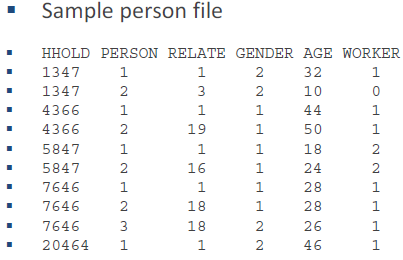
PopSyn需要的输入数据包括Zone Data File（ZoneDataTool生成），PUMS Household File（PUMSPrep生成），PUMS Population File（PUMSPrep），Activity Location File，Process Link File（TransimsNet生成），Vehicle Type Distribution File。利用工具有PUMSPrep，SF3Prep，ZoneData，LocationData。

调查问卷分为家庭问卷和个人问卷。

家庭问卷包括HHOLD（家庭ID），STATE（国家缩写或FIPS编码），PUMA，WEIGHT（调查膨胀系数），PERSONS（家庭成员数），WORKERS（工作人数），VEHICLES（车辆数），INCOME（收入），TYPE（家庭类型），HHAGE（当家年龄），NUM\_LT5（小于5岁的孩子数），NUM\_5TO15（5至15岁孩子数）。

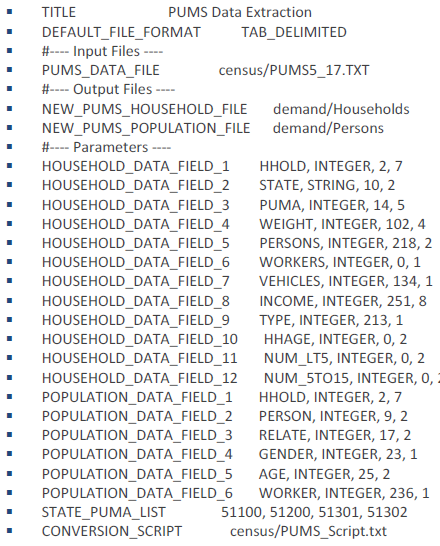


个人问卷包括HHOLD（家庭ID），PERSON（家庭人数），RELATE（家庭中角色），GENDER（性别），AGE，WORKER（是否工作）

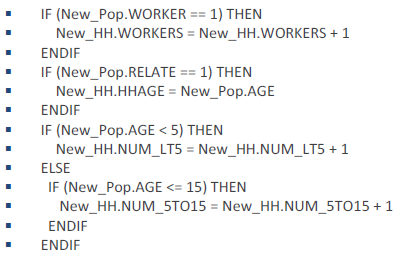


PUMSPrep从PUMS中提取出household和persons文件。

PUMSPrep控制文件格式：

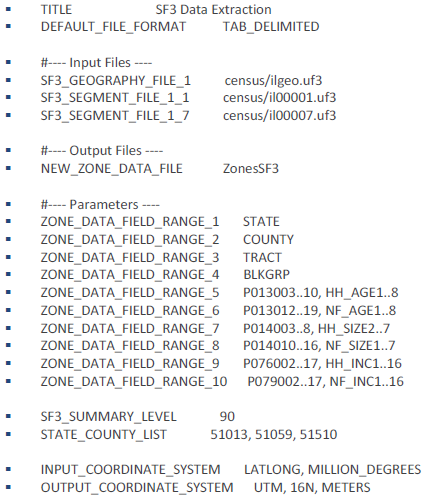


PUMSPrep脚本，类似SQL

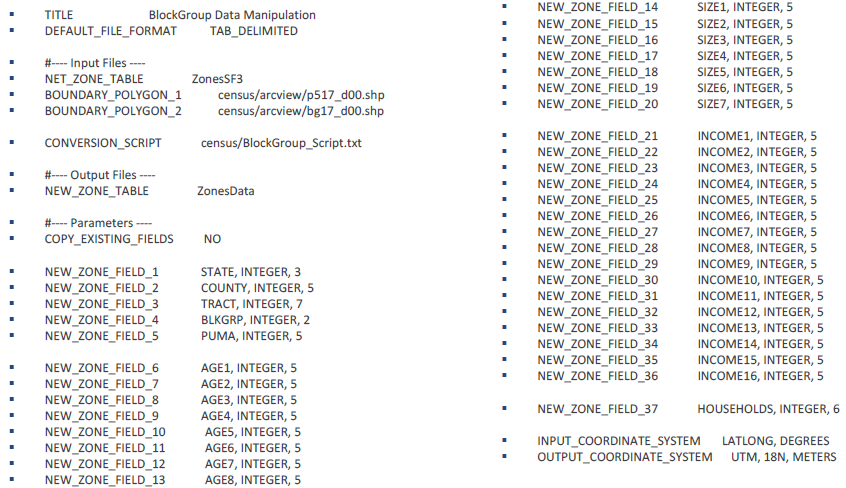


SF3Prep从SF3中提取household和persons文件。

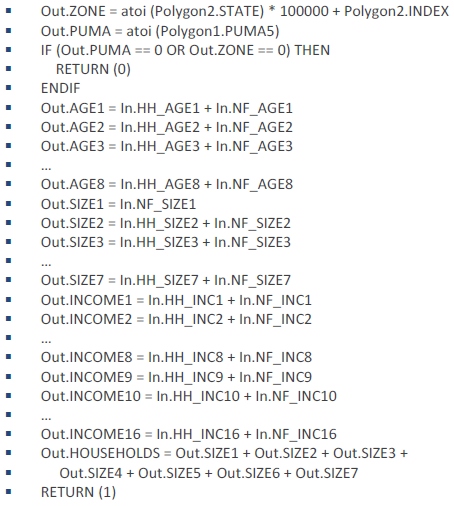
SF3Prep控制文件格式：



ZoneData Tool工具



ZoneData脚本，生成zoneID，将家庭数据与非家庭数据合并，计算总家庭数等



难题是普查数据的推演可能无法反映当前时刻，需要额外的土地利用数据进行家庭活动位置分配。

TRANSIMS虚拟人口采用IPF算法

Deming和Stephan（1940）最早提出IPF，1996年Beckman提出两阶段IPF。

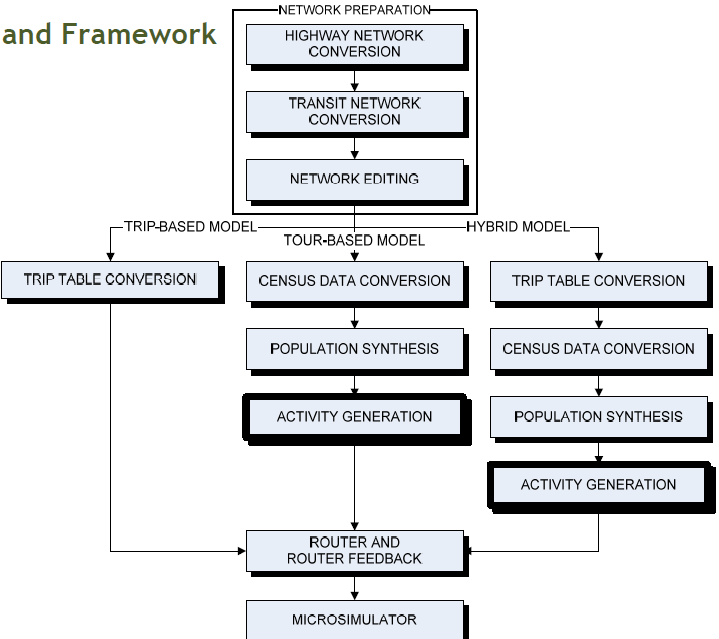
### Activity Generator

Activity geneater决定能够被载入路网的出行需求。

使用上步生成的虚拟人口为家庭中的每个人分配交通行为。

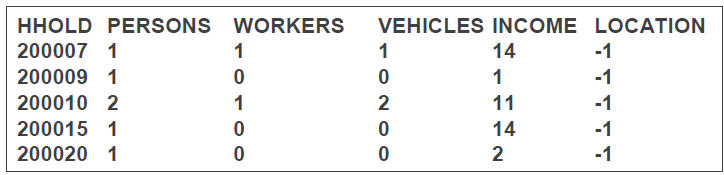
主要输入数据是活动调查，一般通过电话访问一整天的活动行程，比如7点至8点早餐，8点至9点开车上班，9点至13点上班……

ActGen负责分配每个家庭成员的活动，决定活动的起始和结束时间，选择活动地点，定义出行方式。分配活动地点基于位置选择模型，该模型根据活动目的和出行方式作出选择，默认使用可能位置与当前位置的直线距离，再加上吸引因子，选择活动地点。

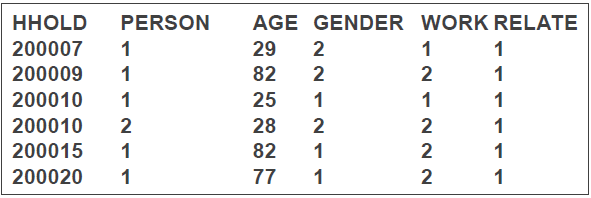


调查数据分为Survey household file，survey population file，survey activity file，survey weights file。

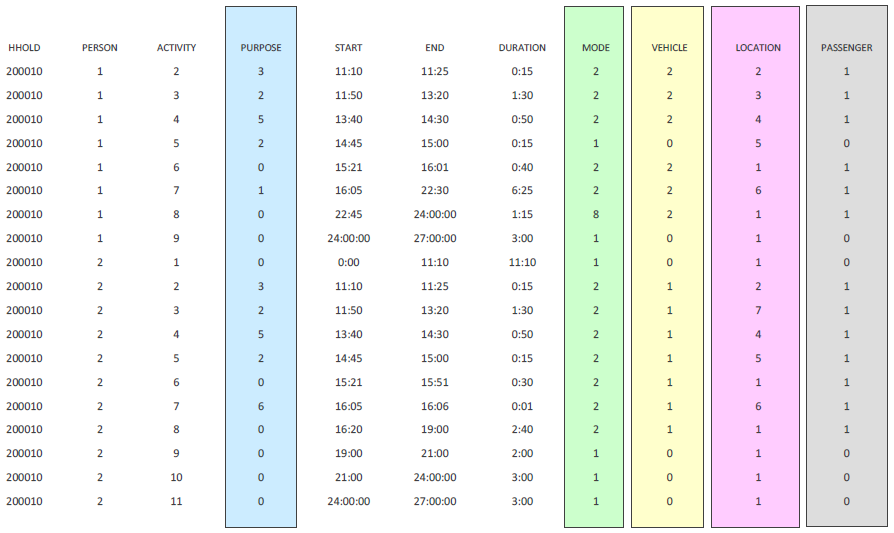
Survey household file，PopSyn会为LOCATION生成活动地点。

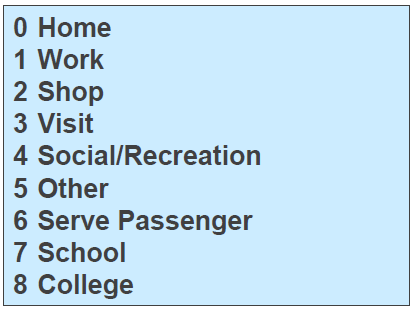
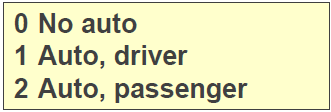


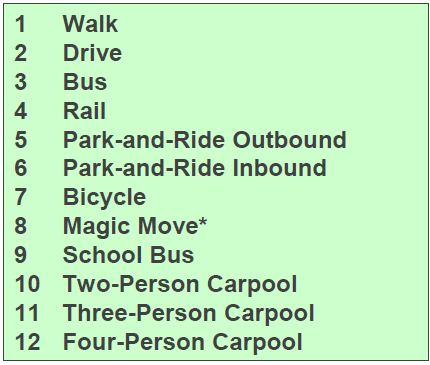
Survey population file，RELATE是1990 US 人口普查缩写，指家庭成员关系。



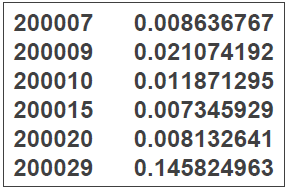
Survey activity file，Passenger除驾驶员外的乘客数，Location家庭地址所属的活动地址。



出行目的车辆类型

出行方式

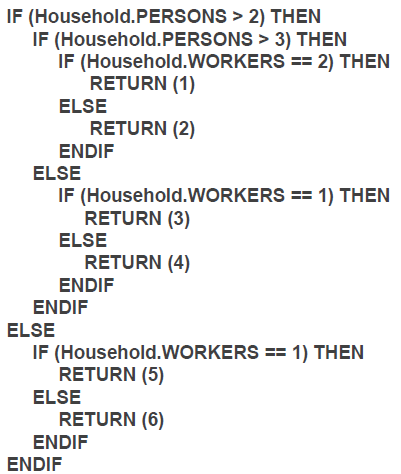
Survey weights file，调查家庭的类型占全部类型的比例。



用统一家庭类型定义全部调查家庭，根据survey weights随机选择某个家庭，分配活动。

Household Type Scripts

以下是家庭类型分配的简单示例，根据家庭人数和工作人数划分为6类。



ActGen为每个家庭成员分配最匹配的成员类型，根据年龄，工作状态，性别，然后根据survey activity file分配活动时间，出行方式等。

调查数据获得困难且昂贵，对整个研究区域具有代表性，由于人口增长或结构变化，出行行为变化很大，导致调查容易过时。

General activity assignment process利用社会经济数据将虚拟家庭与相应的被调查家庭匹配。

对调查记录采用随机偏差以避免重复。

Destination choice model根据虚拟人口分配家庭位置，活动包括上班，上学，购物，娱乐等，基于区域信息。

Transportation mode根据调查所得。

根据人口调查将生成每个家庭的一系列出行活动表。分为家庭和个人活动，包括活动重要性，活动时间，必须出行的时间间隔。地点（例如家庭地址，工作地点和学校地址）会提供给强制活动。购物等地址不会指定由出行者选择。

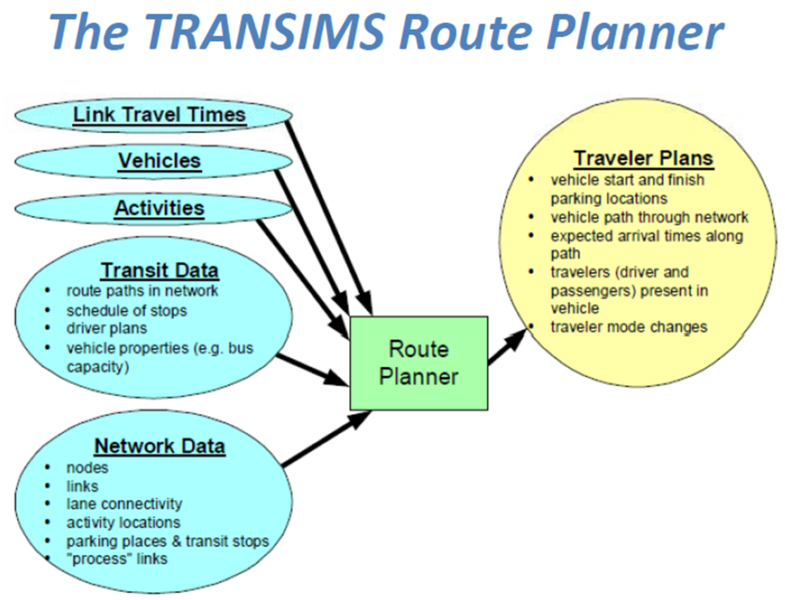
### Route Planner

读入个人活动，决定最快路径。家庭出行允许共乘，比如送孩子上学。出行方式是活动一部分，router不选择出行方式，但根据出发时间提供最佳出行方式。算法根据时间优化，道路不同时间的延迟不同。

根据著名的交通分配方法BPR+估计道路延迟。Router决定最优线路生成精确出行计划。

Microsimulator跟踪出行计划检测车辆间互动，从而生成新的道路延迟取代由Router生成的。

Router和Microsimulator反复迭代直到平衡。

[](http://upload.wikimedia.org/wikipedia/commons/a/a3/Transims_route_planner.png)

### Microsimulator

采用元胞自动机分析车辆间互动。

生成所有出行者和车辆任意时刻的位置。

计算新的道路延迟，选择需重新规划线路的出行。

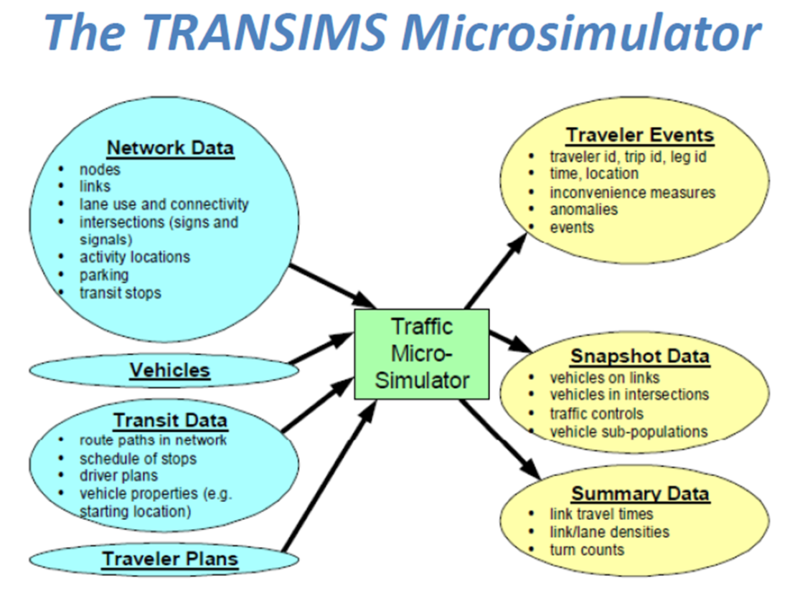
Walk legs：走路不会显式模拟，指明起始点，到达终止点

Driving Legs：起始和终止都在停车场，有特定车辆，车辆需在场或出行者等待车辆，沿指定路线，需指明是否需搭载其他人。

Bus Legs：考虑上下车延迟，载客量和资源限制，乘客上车后直到目的地才下车，根据时刻表行驶，需要可接受的公交线路信息。

元胞自动机：

道路划分为网格，每个网格一个车道宽，7.5米长（用户可指定），测量位置和速度，车辆速度为16.8,33.6,50.3,67.1,83.9mph，4.2版本实现交叉口用网格完全建模，车辆跟驰模型

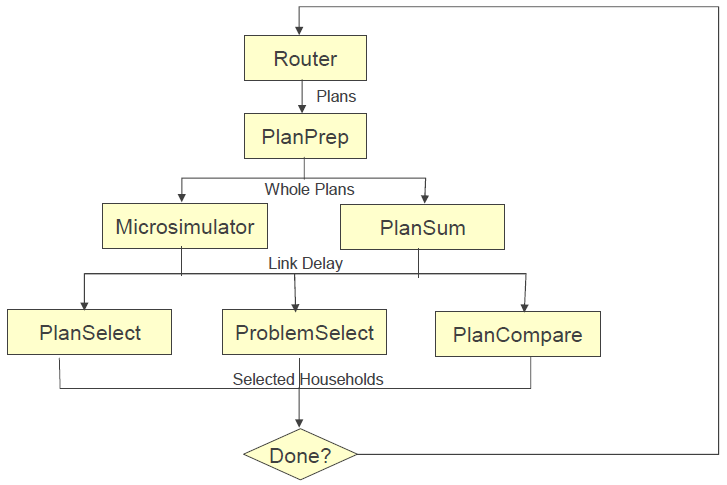
[](http://upload.wikimedia.org/wikipedia/commons/6/60/Transims_microsimulator.png)

### Feedback

反馈应用在平衡计算在路径选择和微观仿真间的迭代。

一些线路不可行：1、长于调查数据；2、指定出行方式下不可行；3、通过反馈发现不可行线路，返回给活动生成以重新选择合适的替代方案；4、重新评估新的出行方案。

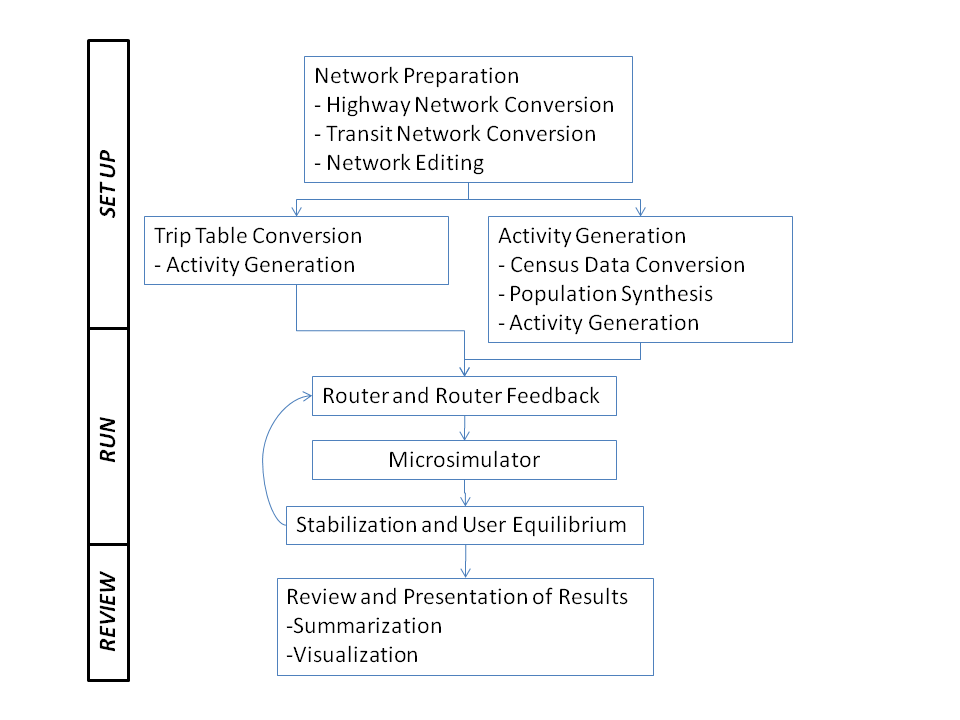
用于重新规划线路的工具包括PlanSelect，ProblemSelect，PlanCompare。



### Results

相对其他传统分析工具，TRANSIMS生成虚拟结果。微观仿真给出高细节数据，可提取每个出行者任意时间的位置并可视化。可视化工具包括fourDscape和Balfour，ArcGIS等GIS工具，Google Earth，NASA World Wind，NCSA，NEXTA。

## 处理流程



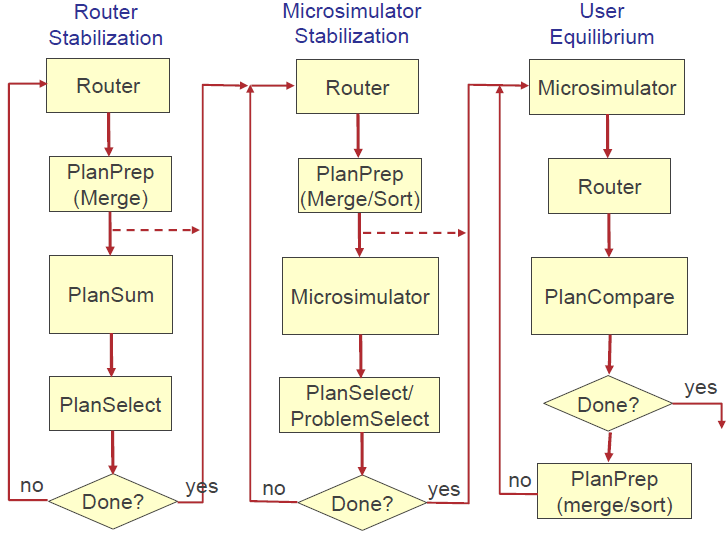
左边分支使用trip tables，右边分支使用activities。

1. **Setting up the model**

**预先需要Network Preparation（准备网络），Trip Table Conversion（出行表转换），Activity Generation（活动生成），Population Synthesizer（虚拟人口）**

1. **Running the model**

**不同的目标：Router stabilization（路径稳定，在仿真前使交通分配更符合逻辑），Microsimulator stabilization（仿真稳定，调试网络和地址仿真问题）和User equilibrium（用户均衡，均衡路径和出行时间），System Equilibrium（稳定道路流量和速度）。**



1. **Reviewing the results**

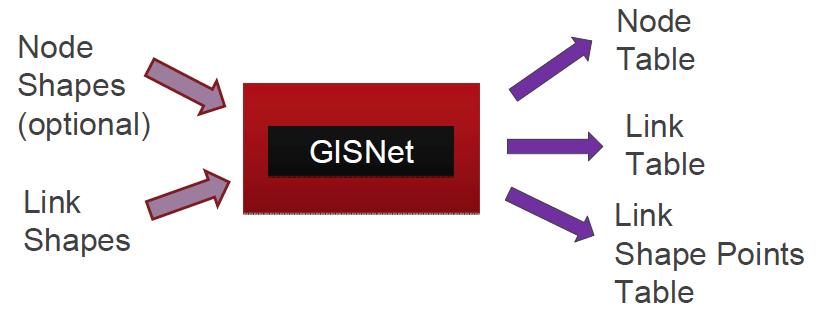
**两种方式：Summarizing the Outputs（总结输出），Visualization（可视化）**

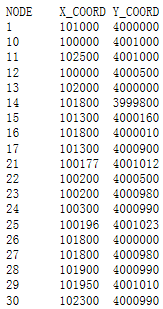
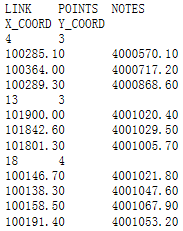
### Network Preparation

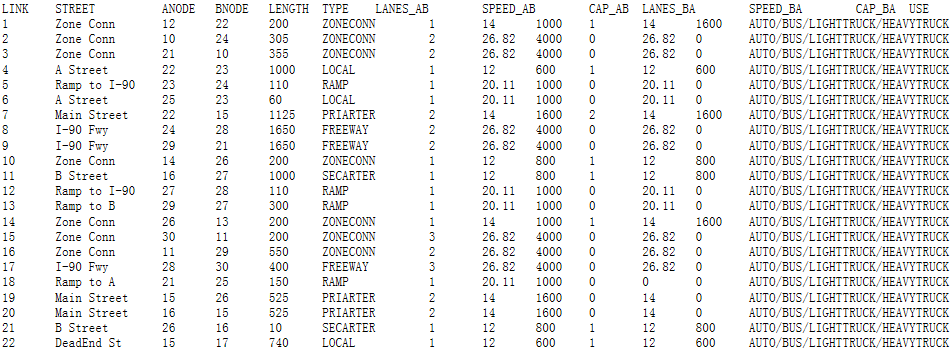
1. Initial Network Inputs

输入数据包括由UTM坐标定义的点（nodes），包含起始/终止点、长度、各方向车道数、速度、容量等信息的道路（links），其他信息包含在shape文件中。

GISNet将ESRI矢量格式转为TRANSIMS格式。



nodes示例shape文件示例



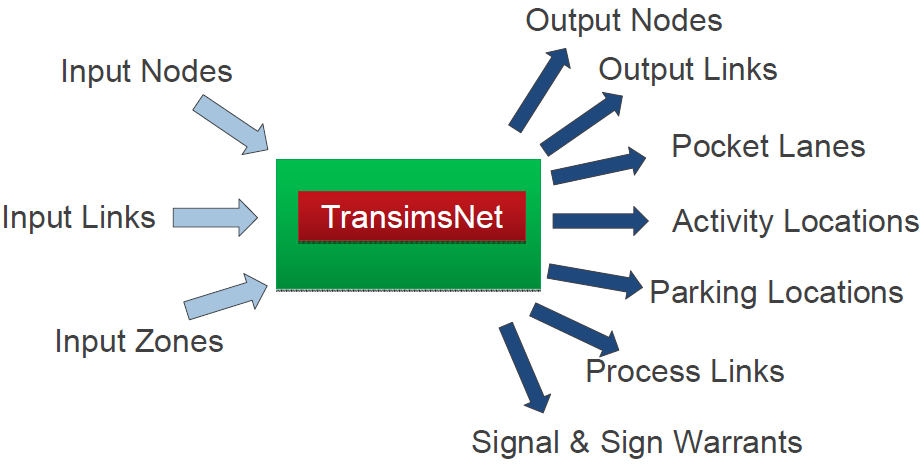
links示例

1. Network Input Corrections and Adjustments

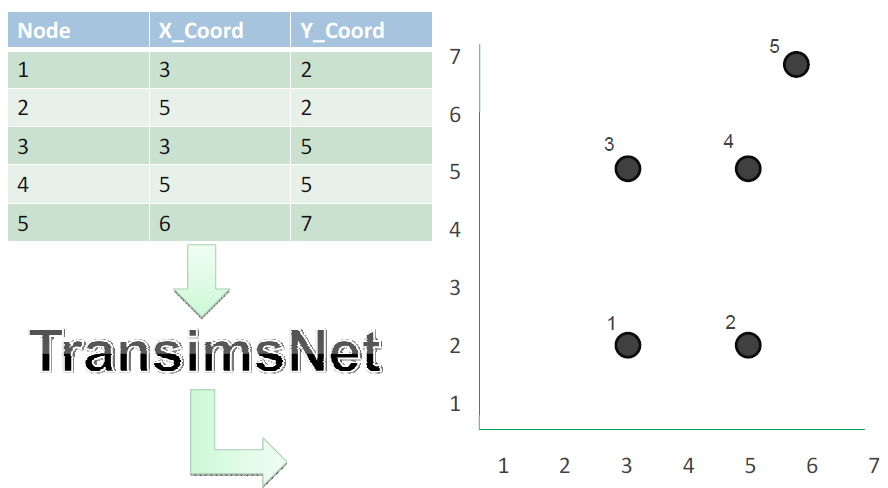
执行网络转换前，先纠正错误，1、删除不需要的道路和点；2、增加所需的道路和点；3、更正车道数和速度；4、编辑极小长度的道路；5、如果信号同时显示两个方向，则转为两个独立信号；6、更正错误形状信息。在上面示例中，点16和26为同一点，删除点16，更新nodes和links。

1. Generate TRANSIMS Network

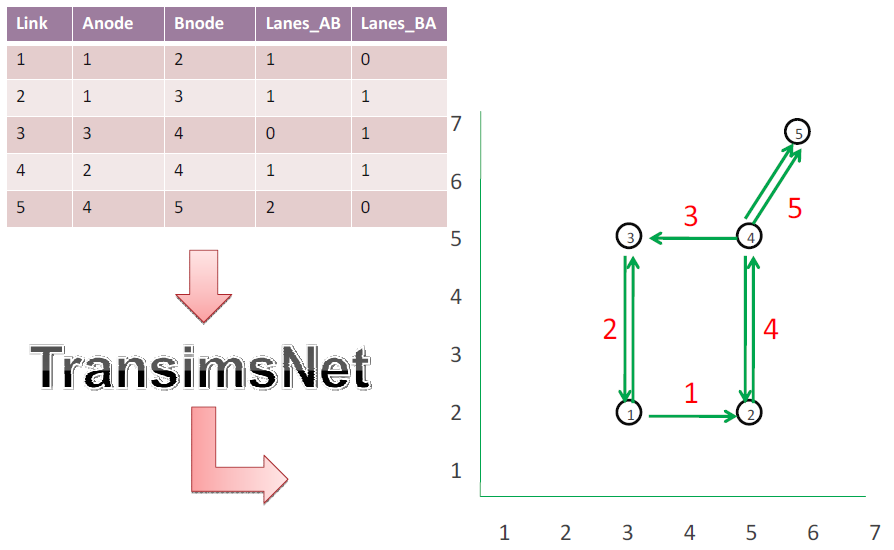
TransimsNet将路网转换为TRANSIMS使用的路网。需要的输入包括Input\_Node、Inupt\_Link、Zone、Turn Prohibition、Shape文件和control文件。



Input Node Table：点被放置到路网中交叉口，分支，车道变化处，终点的位置。

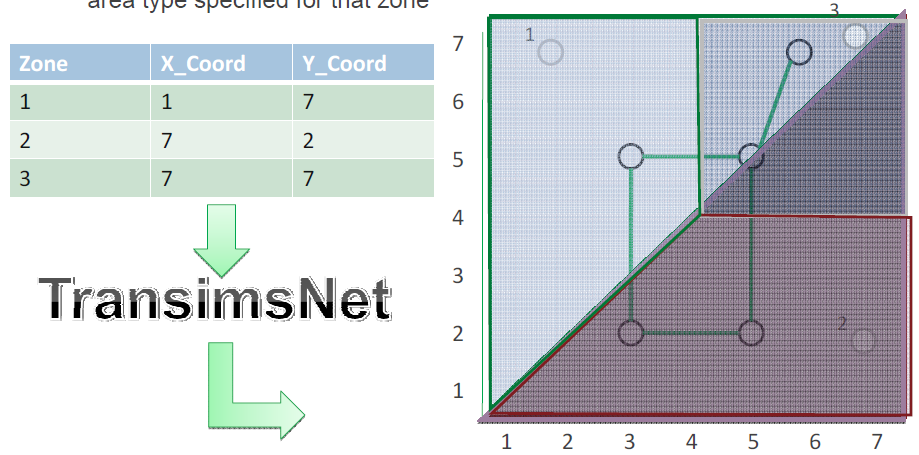


Input Links Table：连线代表道路，人行道，铁路等，连接两点，每个点可连接多条连线，有多种形式，如轻轨。包含两端点A和B，A到B和B到A的车道数，自由速度，道路等级（高速，主干道，次干道，匝道等），什么类型车辆可使用……



Output Links：TransimsNet增加的属性包括ID，道路长度（根据点和形状计算），A和B方向交叉口障碍，A到B和B到A的方位角，双向车道数，车速限制，A到B和B到A的车流容量。

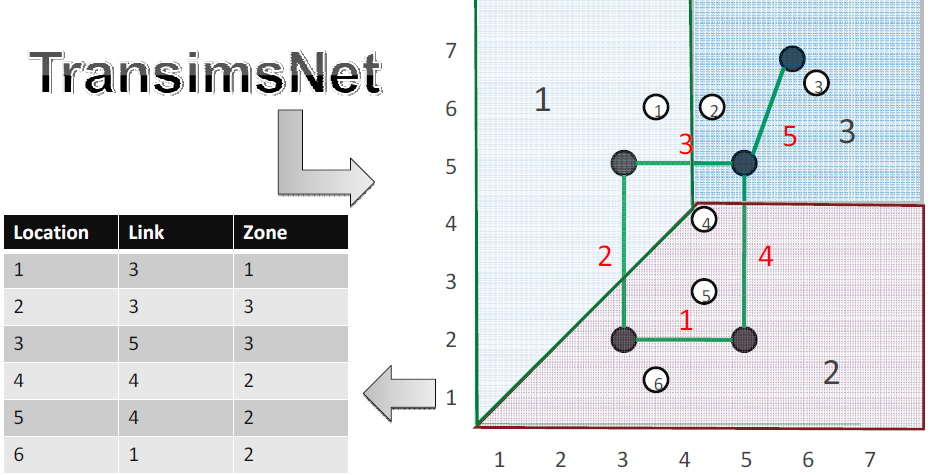
Input Zones Table：指明中心点位置和区域类型。



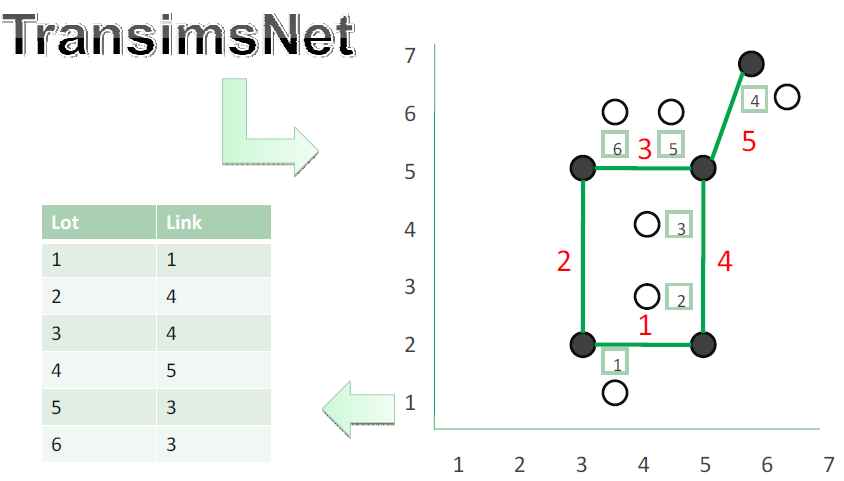
Shape Point Table：提供连线形状以跟踪道路各部分，包含xy坐标。

Keep Nodes Table：TransimsNet将不需要的点删除，移至新的shape point table中。

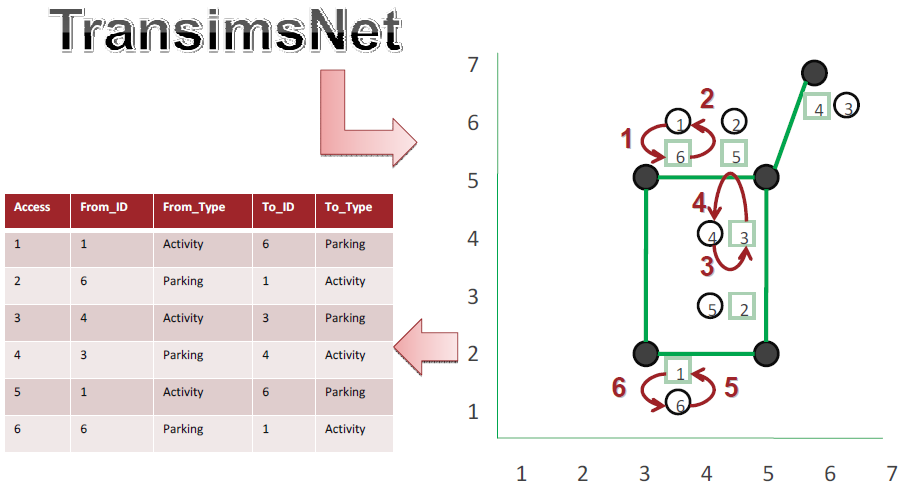
Activity Locations：代表出行的起终点。



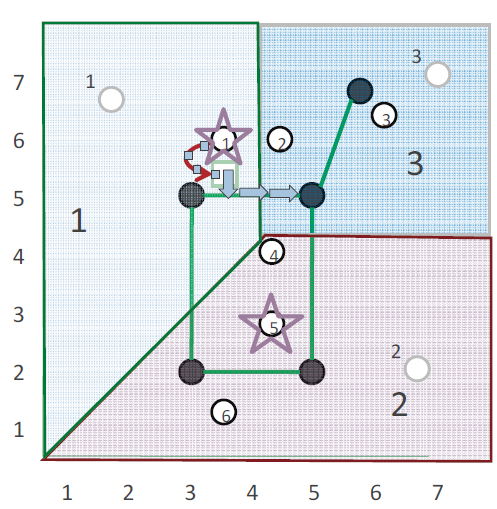
Parking Locations：根据停车条件为道路提供停车场，并与活动范围关联。



Process Links：可认为是在停车位置，活动位置间无目的的移动。

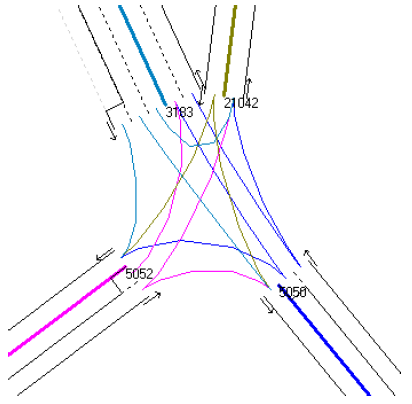


区域中心点决定活动位于哪个zone内，活动位置作为步行者的起点，沿着process link到停车场上车，车辆从停车场到路上，沿着道路到目的地的停车场。

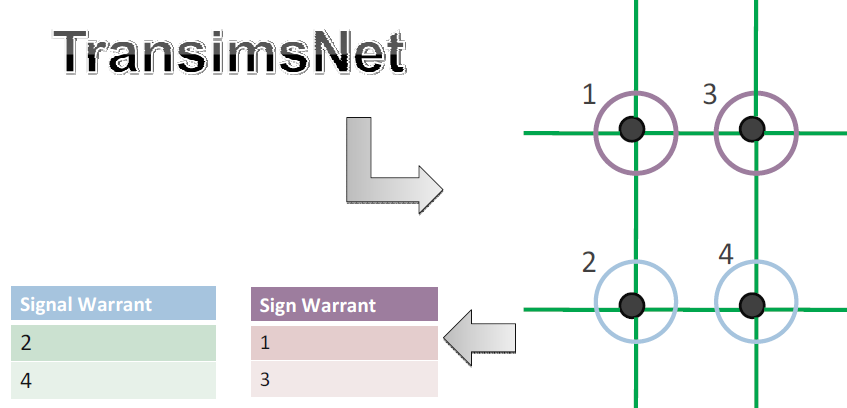


Pocket Lanes：包含点ID，所在线ID，起始位置，车道数，类型（T=turn pocket， P=pull-out pocket，M=merge pocket），长度。

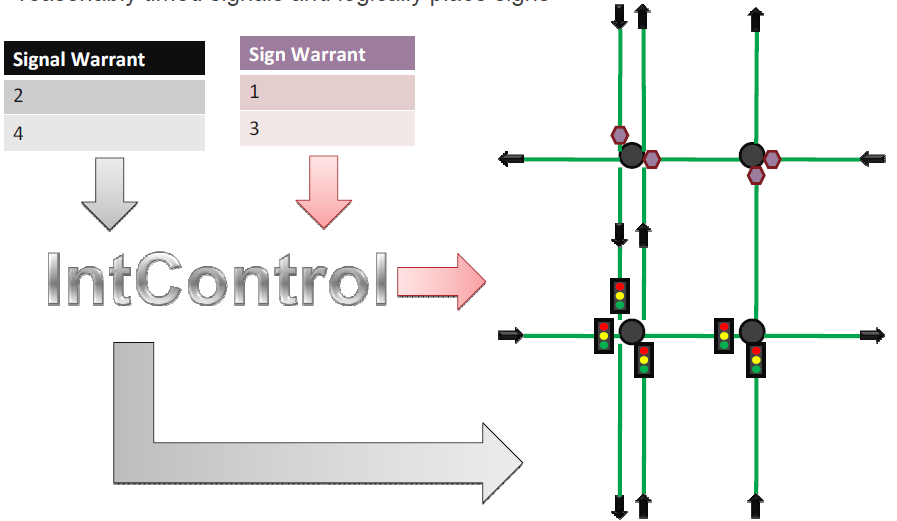
Lane Connectivity：指明在交叉口所有允许的运行途径，包含车道到达的点ID，进入道路ID，离开道路ID，进出道路的车道数。



Signal/Sign Warrants：定义信号和标志最终设置的位置。



IntControl：根据warrant tables和路网属性计算合理的信号周期和标志。



Traffic Signals：包含Signalized node table，Phasing plan table，Timing plan table，Detector table，Signal Coordinator table。

Lane use table：指明某种类型车辆在某个时段可使用指定车道，包括道路ID，车道ID（0指全部车道），车道到达的点ID，限制车辆类型，限制类型（大型车，自行车，自动档车，卡车，公交，铁路），限制起始结束时间。

Trun prohibition table：某种类型车辆在某个时段不能转向，包括点ID，定义转弯点的进出道路ID，限制车辆类型，限制类型（大型车，自行车，自动档车，卡车，公交，铁路），限制起始结束时间。

TransitNet Utility：

Route Header File：包含线路ID，名称，方式，时间周期，起始时间，频率。

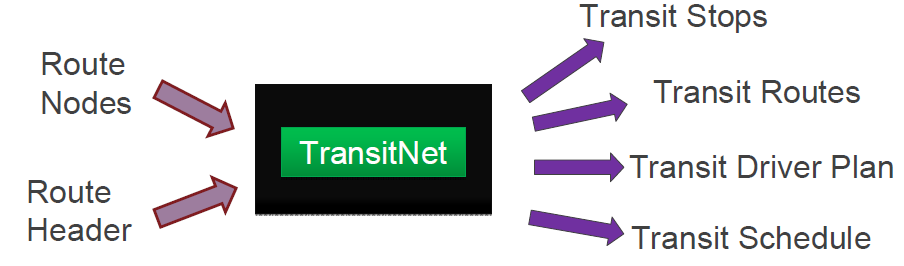
Route Nodes File：包含线路ID，组成线路的点，停车（0不停车）。

Transit Stops：乘客上车和下车地点，连接其他process links。

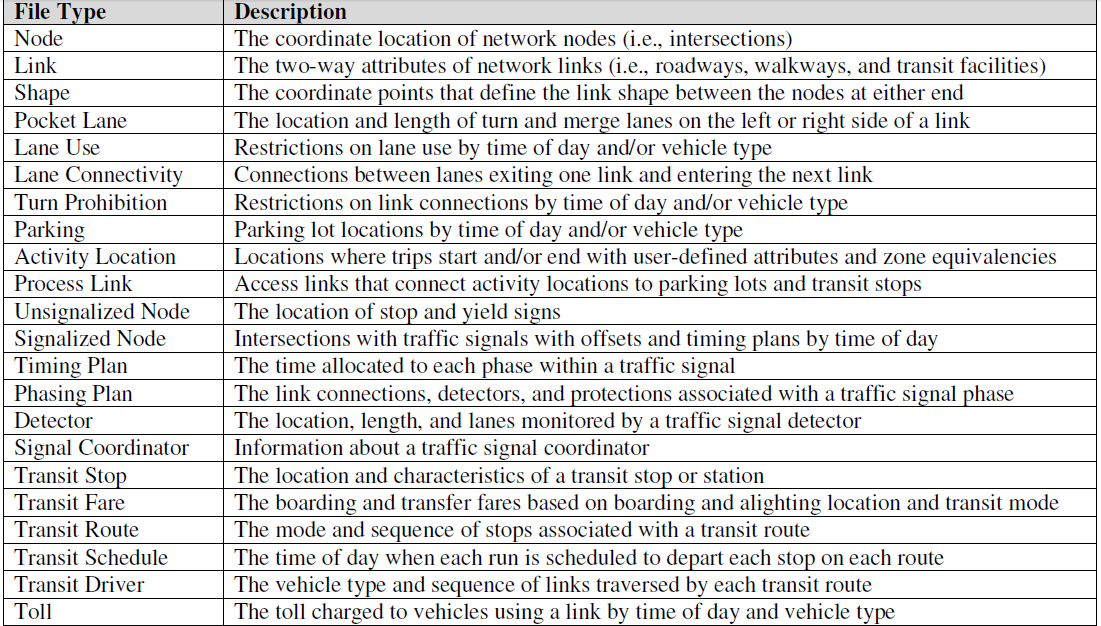
Transit Routes：指明线路中的一系列停车位置，和其他驾驶信息。

Transit driver plan：组成线路的点。

Transit schedule和transit zone tables：每个停车点的出发和达到时刻，区域行驶时间（可选）。



Network Data Tables包含一系列网络表格，并定义各种字段。



1. Lane Connectivity and Signal Warrant Corrections

修正lane\_connectivity文件；修改信号标志。

1. Intersection Control Generation

IntControl用于生成信号交叉口文件，输入包括由TransimsNet生成的Node，Link，Lane\_Connectivity，Pocket\_Lane，Actvity\_Location，Sign\_Warrant，Signal\_Warrant。

1. Subarea Definition

如果进行子区微观仿真，SubAreaNet用于生成子区网络，输入为定义子区的矢量文件。

1. GIS Conversion and Inspection

ArcNet用于从TRANSIMS网络中生成矢量文件，用于GIS。

### Trip Table Conversion

Trip-table包括trip table，trip time和activity location等文件。Trip table定义出行起始和终止区域。Trip time定义不同时间的出行分布。Activity location将区域与多个活动位置关联。

1. Modify input trip tables from the four-step model

从四阶段模型中生成trip table，特别是将出行转换为整数，及删除域内出行。

1. Make any needed modifications to the activity locations

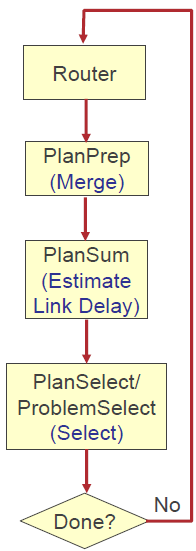
1、用LocationData将出行分配到相应区域；2、找出未分配到出行的区域，用reassign\_zone.py重新分配部分出行。

1. Run ConvertTrips

Convert Trips将trip tables和trip time转换为Trip，Household，Population和Vehicle文件。这些文件包含个人出行，作为TRANSIMS Router的输入。

### Router stabilization

目标是解决网络问题，仿真前改善出行计划使交通分布更合理，反复运行Router生成网络Volume-to-Capacity(V/C)比，为microsimulator提供初始交通流估计。一般5至60次迭代可获得稳定的结果。每次迭代步骤为：



1. 运行Router，输入为网络和出行。第一次迭代后，输入家庭重新规划线路和出行时间，输出包括问题出行（无法确定路线的）和计划（点间或线间出行）。可分区。
2. 运行PlanPrep，合并最近出行路线和前一次。可分区。
3. 可选的，运行PlanTrips，调整限制终点出行的起始时间。
4. 运行PlanSum根据当前计划生成新道路延迟link\_delays。由于线路延迟基于全部计划，因此无法分区。
5. 运行PlanSelect选择下次迭代需重新规划线路的出行。特别是高V/C比的出行线路。

### Microsimulator stabilization

Router迭代稳定后，运行microsimulator，子区仿真取代PlanSum计算新link\_delays。Microsimulator尝试执行由router生成的出行计划，但一些计划由于在非常拥堵线路上导致不可能，microsimulator标记出这些出行，然后删除它们，从而减少问题出行到可接受范围（低于1%），一般需要5至20+次迭代。

1. 运行Router，输入包含网络，出行，家庭和拥堵时间。输出包含问题出行和计划。
2. 运行PlanPrep合并Router最近运行的计划和前一次运行结果。
3. 运行SubareaPlans提取包含子区点和道路的计划。
4. 运行PlanPrep根据出发时间排序计划（为Microsimulator）。
5. 运行Microsimulator尝试执行子区计划，产生输出，包含子区道路延迟。
6. 运行PlanSum生成新的不在子区的道路延迟。
7. 运行LinkDelay合并Microsimulator和PlanSum生成的道路延迟。
8. 运行ProblemSelect选择下次迭代中需重新规划线路的计划。特别是Microsimulator无法执行的机会。
9. 运行ProblemSelect标记在ROUTER中失败的出行家庭。
10. 增加标记家庭到家庭表格中。

### User equilibrium

目标是获得用户均衡，即改变个人出行无法提高旅行时间。一般需要5至20+次迭代。Router每次都要重新规划线路，耗时长。

1. 运行Router，同上。
2. 运行PlanCompare比较当前和上一次的出行时间和线路。
3. 运行PlanPrep合并当前和上一次的出行计划。
4. 运行SubareaPlans提取包含子区点和道路的计划。
5. 运行PlanPrep根据出发时间排序计划（为Microsimulator）。
6. 运行Microsimulator尝试执行子区计划，产生输出，包含子区道路延迟。
7. 运行PlanSum生成新的不在子区的道路延迟。
8. 运行LinkDelay合并Microsimulator和PlanSum生成的道路延迟。

### Summarizing & Visualization

Microsimulator提供以下输出文件：

|  |  |
| --- | --- |
| **Program** | **Description** |
| EventSum | Summarize and report data from Event files |
| LineSum | Summmarize and report data from transit ridership files |
| LinkSum | Summmarize and report data from LinkDelay files |
| TripSum | Summmarize and report data from Trip or Activity files |
| Validate | Compare TRANSIMS flows to a file of observed traffic or transit counts |

TRANSIMS提供可用于GIS的矢量文件：

|  |  |
| --- | --- |
| **Program** | **Description** |
| ArcDelay | Create ArcView shapefiles from Link Delay files |
| ArcDiff | Combine two ArcDelay files and calculate the difference |
| ArcNet | Create ArcView shapefiles from network tables |
| ArcPlan | Create ArcView shapefiles from Plan files |
| ArcProblem | Create ArcView shapefiles from Problem files |
| ArcRider | Create ArcView shapefiles from Ridership files |
| ArcSnapshot | Create ArcView shapefiles from Snapshot or Occupancy files |