系统设计说明书

目录

1 引言

1.1 编写目的

编写该项说明书的主要目的是对该物流配送软件的总体设计所产生的功能模块进行过程描述，包括程序的策划、实现、测试三个方面。 我组主要基于类库 MFC 开发物流配送最优路径规划模拟系统，基本解决了物流公司从仓库出发依次向各客户配送货物再返回仓库所存在的最短路线、最优路线的静态规划、动态规划问题。

1.2 背景说明

软件系统名称：物流配送最优路径规划模拟系统

软件开发平台：VC++6.0(Microsoft Visual C++6.0 中文版)、Macromedia Firework 8、Adobe Photoshop

类库技术支持：MFC(Microsoft Foundation Classes)

开发语言：C\C++

运行平台：Windows 系统平台

1.3 定义及说明

对文档中的特殊名词的简单定义及说明，如表 1-1:

表 1-1 名词定义及说明

名称	定义
输入区	用户需要手动输入、选择的区域
状态区	根据输入信息及行进信息最后反馈的区域
地图区	用户需要手动标记仓库、客户、堵车路段的区域
推荐货车配置	写有小车型号、功率、自重、耗油量的对话框
演算公式	写有状态栏各种指标结果的计算公式的对话框
使用说明	写有详细的使用说明信息的对话框
需求量（吨）	客户需要货物的总吨数，影响平均行驶速度（额定功率÷质量）
燃油费（元/公里）	和行驶里程数共同影响配送成本（里程数 x 燃油费）

其余费（元）	配送过程中需要的其他费用
比例尺	图上每像素：实际千米数=30：1
总里程（km）	从仓库出发依次配送到各个客户后再返回仓库的总距离
配送成本（元）	（里程数 x 燃油费+时间 x 每小时损失费+其余费）的计算结果
预期时长（h）	估计全程耗时，（各路段里程/各路段速度）再求和的结果
速度（km/h）	由汽车当前所行驶在的道路路况、平均行驶速度、波动值决定
路况（流畅、拥挤）	由汽车当前所行驶在的道路路况决定
方案选择	可以选择不同种配送方案
最短路径	搜索最短里程数的路线，只以里程数最小优先，忽视堵车路段
最短时间	首先按最短路线搜索，后随用户设置的堵车路段以时间最短优先
静态规划	汽车未出发前就事先设置好的路况，寻找一条最短里程的路
动态规划	汽车出发后在行进途中设置路况，寻找一条可绕开堵车路段的路
寻路按钮	输入信息合法、标记合法后按该按钮进行初次的路线规划
暂停按钮	行进暂停，用户可以利用此按钮动态标注堵车路段
启动按钮	行进继续，系统会根据刚才用户标记的堵车路段重新规划理想路线
重置按钮	清空标记的所有地点与路段，用户可以重新进行输入与选择
重找按钮	随机算法的缺陷导致每次不一定能找到最短路线，用户按下后系统可以向着更短里程的方向重新搜索路线
收敛重找模式	重找得到的结果一定比上一次搜索得到的结果短（实际用）
非收敛重找模式	重找得到的结果是经过一次算法得到的随机值（测试算法用）
测试按钮	输入信息合法、标记合法后按该按钮进行手动测试，用户可以朝着想要前进的方向，双击道路，汽车会自动按用户的思路行进
重测按钮	测试时点错路线，用户可按下此按钮重新来过
结点	道路与道路之间的交叉点或在道路旁边的客户、仓库地点
标记点	可以按右键进行仓库、客户标记的深蓝色点
仓库点	右键点击“标记仓库”后出现的黑白相间的旗帜
客户点	右键点击“标记客户”后出现的橙色的旗帜
ID	结点、道路的标号（具体请见测试文档）
A 星算法	一种静态路网中求解最短路径的有效算法
OPEN 链表	用来添加已经遍历过的相邻结点的链表
CLOSE 链表	用来添加已经扩展过的中心结点的链表
TRAFFIC 链表	用来添加遍历到堵车结点的链表
堵车结点	一条堵车道路的两端的结点
中心结点	扩展出其周围有相邻道路的结点的结点
相邻结点	由中心结点扩展而出的结点
父结点	同中心结点类似，但该结点一定是到子结点最近的结点
子结点	一个指向父结点的结点
复活结点	因 OPEN 链表已没有任何中心结点可以扩展，而扩展 TRAFFIC 中的堵车结点的结点
估价函数 f(n)	到一个结点的路程与该结点到终点的预计路程之和，f(n)=g(n)+h(n)
已走路程 g(n)	从起始点到一个结点所走过的路程
启发路程 h(n)	预计从一个结点到终止点大概需要走过的路程
回溯路线	调用 A 星算法后每个结点指向了其父结点，从终止点依次追溯至起

	始点找到最短路线的过程
改进模拟退火算法	一种经改进的模拟物理学中固体退火原理的算法
遗传算法	一种模拟自然进化过程的算法
数组顺序	用户标记完仓库和客户后动态产生一个数组，该数组的首尾存放仓库的 ID，中间以一种顺序有且仅存放一次不同客户的 ID
父亲数组	未经下一次打乱顺序的当前数组
儿子数组	经过父亲数组打乱顺序而得的数组
变异	模拟退火算法中以一定概率接受路程稍长的数组顺序，将其作为父亲数组，通过打乱顺序可能得到比舍弃的数组更优的儿子数组
记忆	接受稍长数组的时候记住旧数组的过程
重升温	经过一次退火过程后找到的最短路程大于文件中记录的历史最小值时，需要重新进行退火过程
最终路线链表	经过改进模拟退火算法后得到的最终路线的链表，该链表内部依次存有先后到达结点的 ID，通过此 ID 可以画出行进路线

1.4 参考资料

[1] 盖伊曼德若利(意).软件工程基础[M]. 中国电力出版社 .2006

[2] 佚名 .深入 A*算法.http://dev.gameres.com/Program/Abstract/a8first_2.htm

[3] 康立山、谢云等.计算方法丛书——非数值并行算法[M]. 科学出版社, 1994 - 244 页.2007

[4] 王小平、曹立明.遗传算法——理论、应用与软件实现[M].西安交通大学出版社

[5] 钟敏.A*算法估价函数的特性分析[J].武汉工程职业技术学院学报, 2006 年 02 期

[6] 田明星.路径规划在车辆导航系统中的应用研究[J]. 北京交通大学.2009

2 程序系统的结构

2.1 全程模块

介绍本软件的设计及开发过程的介绍与分工，如表 2-1。

表 2-1 系统开发过程及描述

模块名称	开发负责人	功能描述
市场调查	A	对市场应用价值、需求度等信息的调查与统计。增强系统的价值性。
资料归纳	A	通过查阅资料，归纳总结物流配送需要的指标，及计算公式与计算方法。增强系统的实际准确性。
文档编写	B	编写需求规格书、系统设计说明书、测试文档，让软件的编写游刃有余。增强了编写的有条理性。
界面与图形	C	设计简洁清晰界面，并利用 Photoshop 与 Firework 处理

		各种图样。增强系统的美观性。
消息响应	D	各控件包括按钮、鼠标右键、鼠标移动、鼠标双击、定时器、弹出式菜单需要响应的动作。增强人机交互性。
算法设计	E	A 星算法、改进模拟退火算法、遗传算法三者结合实现最短路径、动态规划。增强了系统的智能性。
后期优化	F	添加选中状态、音效、使用说明等功能。增强了观赏性。
测试数据	G	演示坐标，注明坐标，通过手动测算得到的数据与系统状态栏的数据进行对比。增强了说服性。
视频制作	H	宣传视频的制作与处理

2.2 软件的总体构架(如图 2-1)

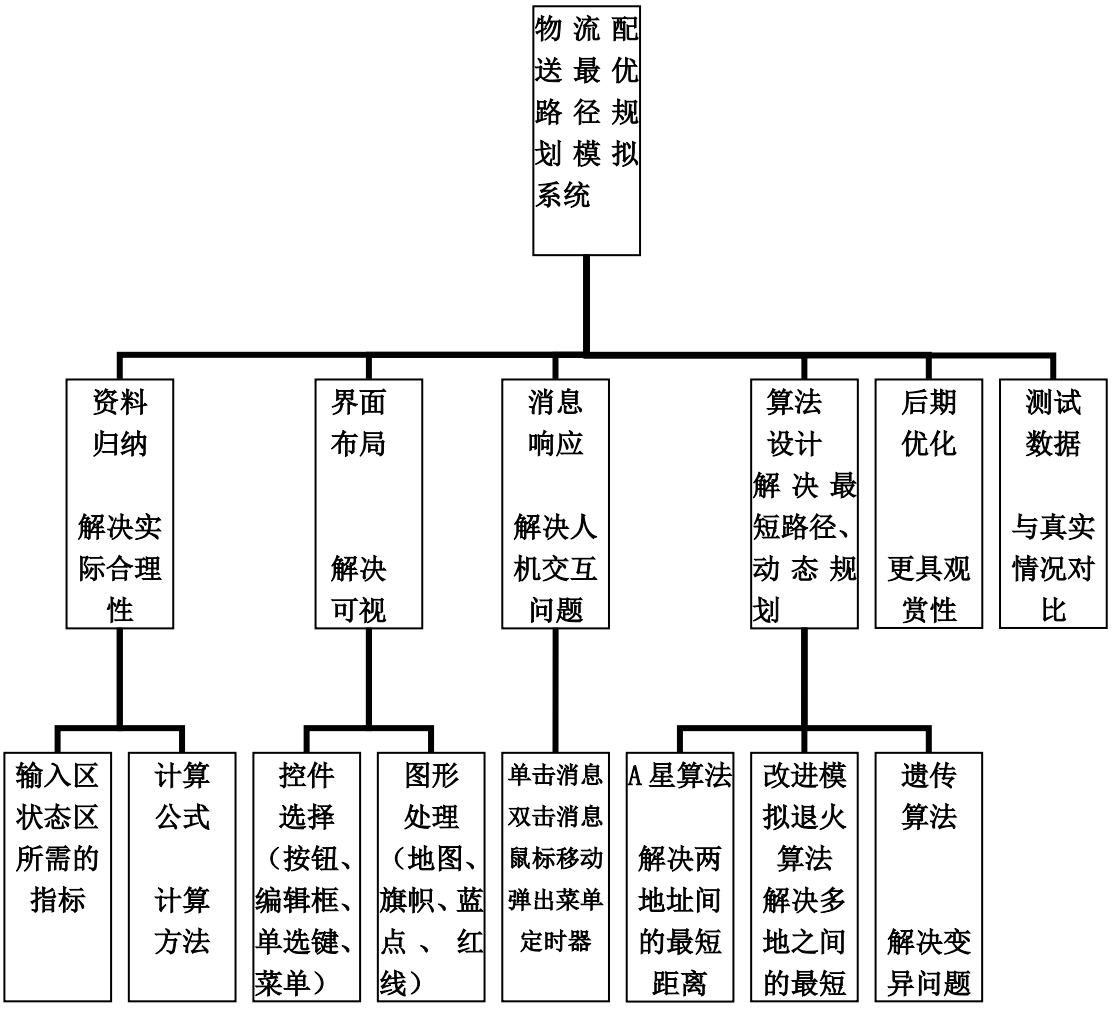


图 2-1 软件的总体构架

3 程序设计说明

3.1 程序头文件及标识符说明

（1）头文件

表 3-1 头文件说明

头文件	主要使用函数	作用
<math.h>	sqrt() pow() fabs()	A 星算法中求两点间的距离
<stdlib.h>	srand() rand()	改进模拟退火算法中随机概率
<time.h>	time()	获取系统时间打乱随机种子
"HelpDlg.h"	HelpDlg 对话框类	提供使用说明对话框类
"winmm.lib"	PlaySound() Beep()	音效设置

（2）宏定义

表 3-2 宏定义说明

宏名	值	含义
UNKNOWN	0x0000	未知地点是仓库还是客户、未知父结点
OUTDOOR	0x1000	鼠标落在对话框不可选中的无意义位置
ONBLUE	0x2000	鼠标落在蓝点上
ONROAD	0x3000	鼠标落在道路上
STOREFLAG	0x4000	右键选择了标记仓库，表示该位置是仓库
CLIENTFLAG	0x5000	右键选择了标记客户，表示该位置是客户
HIDEFLAG	0x6000	已经配送到的客户暂时隐藏，防止重复配送
SMOOTH	0x7000	右键选择了流畅，表示该道路是流畅路况
CROWD	0x8000	右键选择了拥挤，表示该道路是拥挤路况

（3）自定义类

表 3-3 自定义类说明

类名	类作用	成员类型	成员名	成员作用
NODE	储存 1~71 个结点信息。其中 ID 为 1~49 的结点是道路与道路的交叉点，51~71 的结点是可以被标记的客户仓库结点	int	x	该结点横坐标
		int	y	该结点纵坐标
		int	aroundid[9]	该结点相邻结点
		int	roadid[6]	该结点相连道路
		int	id	该结点 ID
		int	choose	该结点被选择情况（蓝点）
		int	father	该结点父结点
		BOOL	running	该结点所在道路是否在行进中
		BOOL	been	该结点是否已配送到（蓝点）
ROAD	储存 1~75 条道路的信息	int	id	该道路 ID
		int	traffic	该道路路况
		BOOL	running	该道路是否在行进中
		BOOL	then	该道路是否被动态添加堵车事件
ASTAR 链表类	A 星算法时需要用到的链表类	int	id	该结点 ID
		int	center	该结点所来源的中心结点
		double	g	起点到该结点需要走过的路程
		double	price	该结点到终点的估价值
		ASTAR	*next	指向下一个 ASTAR 类的指针

ROUTE 链表类	表示路线的先后顺序的链表类	int	id	该结点 ID
		ROUTE	*next	指向下一个 ROUTE 类的指针

注：结点和道路的具体信息请参见测试文档。

（4）全局变量

表 3-4 全局变量说明

类型	名称	作用
ROUTE	*Head	指向最终那条路线头结点的指针
ROUTE	*Start	开辟内存给新结点的指针
ROUTE	*Link	依次拜访最终路线的结点的指针
ROUTE	*link,*lin	在其他函数中动态改变最终路线链表用
CPoint	Point1	右键点击处的坐标在其他函数中使用
CPoint	Point2	图样的坐标传入其他图像函数中显示
CDC	memDC1,*pDC1	设备上下文对象的类，调用函数用
CBitmap	bmp1	位图类，存截图
CRect	rt1	矩形类，装截图
BOOL	timer	定时器是否开启
BOOL	ing	汽车是否在行进途中
BOOL	end	汽车是否回到仓库
BOOL	crowd	中心结点与其相邻结点是否拥挤
BOOL	adjust	是否动态规划过
BOOL	test	是否点击测试按钮
int	touch	选中的是蓝点或道路或无意义区域
int	only	仓库只能有一个
int	plan	所选择方案（未选择、最短路径、最短时间）
int	speed	定时器响应间隔即行进速度
int	lastnode	手动测试时上次所到达的结点 ID
double	xi	当前红线头部所在横坐标，在定时器中累加
double	yi	当前红线头部所在纵坐标，在定时器中累加
double	sonx	Head 表中下一次将要到达的结点横坐标
double	sony	Head 表中下一次将要到达的结点纵坐标
double	D	总里程，由 Head 路线总长和比例尺计算得到
double	O	汽车每公里耗油升，由需求量决定卡车规格不同
double	P	汽车额定功率，由需求量决定卡车规格不同
double	M	汽车质量，需求量与卡车质量总和
double	V	行驶速度
double	S	配送成本
double	T	预计时间
CMenu	m_Menu	弹出式菜单对象，便于调用 CMenu 类中函数
CPen	pen1	画笔对象，红色画笔画行进红线
CPen	pen2	画笔对象，蓝色画笔画标记点
CPen	pen3	画笔对象，浅蓝画笔画选中点

（5）控件关联变量

表 3-5 控件关联变量说明

类型	名称	关联控件	作用
CString	m_NeedEdit	编辑框	利用 UpdateData(TRUE)获取需求量
CString	m_FuelEdit	编辑框	利用 UpdateData(TRUE)获取燃油费
CString	m_OtherEdit	编辑框	利用 UpdateData(TRUE)获取其余费
int	m_MinRadio	单选按钮	利用 UpdateData(TRUE)获取按钮选择
CString	m_DistanceStr	静态文本	利用 UpdateData(FALSE)显示总里程
CString	m_PrimeStr	静态文本	利用 UpdateData(FALSE)显示配送成本
CString	m_WholeStr	静态文本	利用 UpdateData(FALSE)显示预计时间
CString	m_SpeedStr	静态文本	利用 UpdateData(FALSE)显示速度
CString	m_TrafficStr	静态文本	利用 UpdateData(FALSE)显示路况

（6） 消息响应函数

表 3-6 消息响应函数说明

返回类型	名称	响应消息	所属类
BOOL	OnInitDialog()	初始化	CMyDlg
void	OnHelpMenu()	帮助菜单按下	CMyDlg
void	OnOkButton()	确定按钮按下	CMyDlg
void	OnStopButton()	暂停/启动按钮按下	CMyDlg
void	OnReplayButton()	重找按钮按下	CMyDlg
void	OnResetButton()	重置按钮按下	CMyDlg
void	OnTestButton()	测试按钮按下	CMyDlg
void	OnStoreMenu()	标记仓库菜单按下	CMyDlg
void	OnClientMenu()	标记客户菜单按下	CMyDlg
void	OnSmoothMenu()	流畅菜单按下	CMyDlg
void	OnCrowdMenu()	拥挤菜单按下	CMyDlg
void	OnRButtonDown()	右键按下	CMyDlg
void	OnMouseMove()	鼠标移动	CMyDlg
void	OnLButtonDblClk()	左键双击	CMyDlg
void	OnEnterButton()	我知道了按钮按下	CHelpDlg

（7）普通函数

表 3-7 普通函数说明

返回类型	名称	所属类	作用
int*	Array()	无	由标注地点动态分配数组
int*	Product(int *,int)	无	将数组顺序变异后产生新顺序
ROUTE*	FindRoute(int *,int *)	无	A 星算法产生两标记点间的最短路线
void	GoBack(int)	无	从终点起由其指向的父结点回溯路线
ROUTE*	FindCircle(int *)	无	以形参中的数组顺序找巡回路线
ROUTE*	Cool(int *,int)	无	退火与遗传算法结合筛选最短路线
BOOL	Recool(ROUTE *,int)	无	检测是否比文件中记录的最短路线短

BOOL	CheckBlank(CString)	无	检测填入信息是否为空
BOOL	CheckNum(CString)	无	检测数字是否非法
BOOL	CheckSurvive(int)	无	检测堵车是否无路可走
BOOL	CheckTraffic(int ,int)	无	检测两个结点共有道路是否拥堵
BOOL	CheckNode(int ,int)	无	检测一结点周围是否存在另一结点
void	ChangeCircle()	无	动态规划新最终路线
void	FreeAstar(ASTAR *)	无	释放 ASTAR 链表
void	FreeRoute(ROUTE *)	无	释放 ROUTE 链表
void	AddRunning(int ,int)	无	由两结点确定道路并修改为行进中
void	ChooseA(double)	无	由输入区的信息确定常量
void	CalculateTime()	无	计算预期时间
void	CalculateHead()	无	计算 Head 链表总里程
double	CalculateDis(int ,int)	无	计算两结点的距离
double	CalculateSpeed(int ,int)	无	由两结点确定道路再计算速度
void	DarkBlue(int ,int)	CMyDlg	画深蓝点
void	LightBlue(int ,int)	CMyDlg	画浅蓝点
void	AllBlue()	CMyDlg	画所有深蓝点
void	StoreFlag()	CMyDlg	仓库旗图样
void	ClientFlag()	CMyDlg	客户旗图样
void	BeenFlag()	CMyDlg	红勾图样
void	ShowBeen()	CMyDlg	筛选已经到过的客户
void	CutScreen()	CMyDlg	截图
void	PutScreen()	CMyDlg	显示截图
void	ChangeAnytime(int ,int)	CMyDlg	由两点确定道路速度与路况随时更新
void	DrawLine()	CMyDlg	画行进红线
void	Map()	CMyDlg	地图图样
void	Picture(int ,int ,int ,int)	CMyDlg	道路重绘
void	OnPicture(int ,int ,int ,int)	CMyDlg	选中道路
void	TrafficFlag(int)	CMyDlg	重绘标记了拥堵的道路
void	Statistic()	CMyDlg	显示总里程、总成本、时间、速度、路况
BOOL	JoinTable(int)	CMyDlg	检测用户双击的道路是否合法，合法则添加到 Head 最终路线链表

3.2 功能（IPO 图）

（1）求最短路径

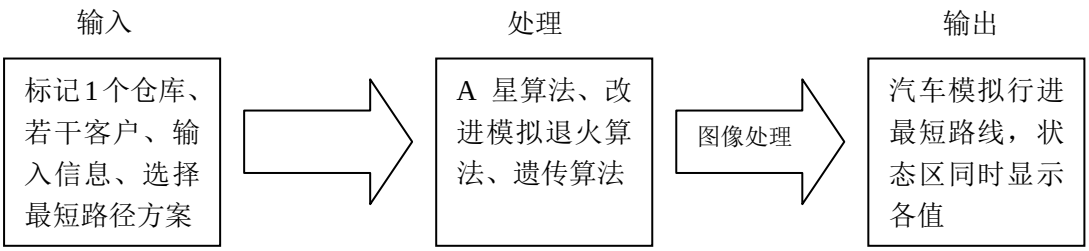


图 3-1 最短路径实现流程

（2）动态规划

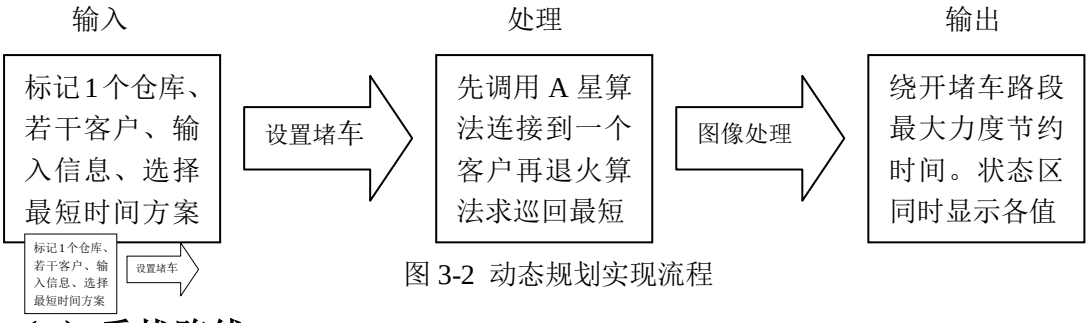


图 3-2 动态规划实现流程

（3）重找路线

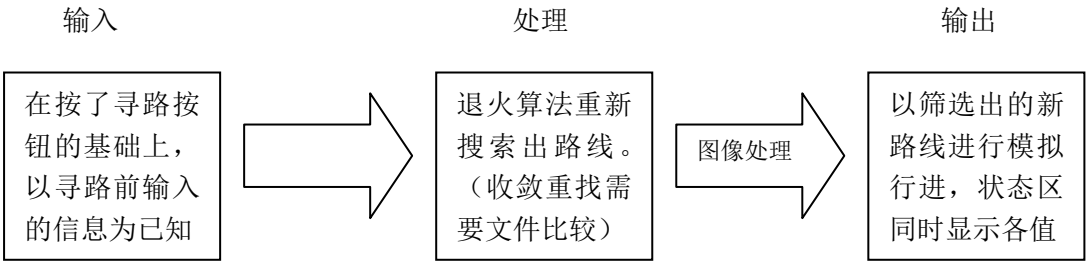


图 3-3 重找路线实现流程

（4）手动测试

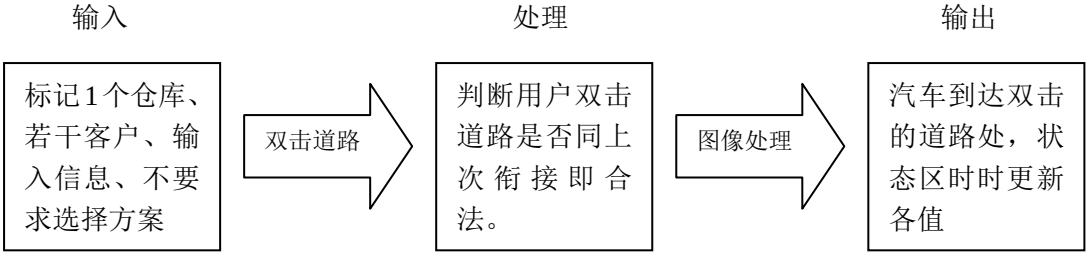


图 3-4 手动测试实现流程

（5）暂停/启动

图 3-5 暂停/启动实现流程

（6）重置

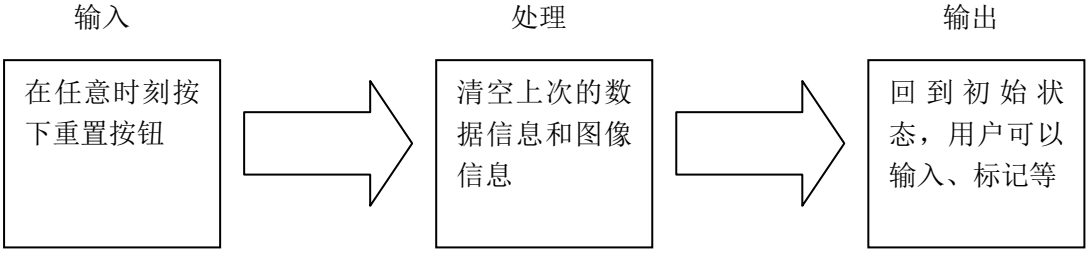


图 3-6 重置实现流程

3.3 性能

（1）精度：该程序的数据基本由 int 和 double 两类型组成，因此精度在 15~16 位。

（2）灵活性：该程序所需人工录入的数据只有地图信息。包括结点 ID,结点相邻结点 ID,结点相连道路 ID,道路 ID。简单易行，灵活性较强。

（3）可替换性：该程序只需改变地图，改变地图数据，及少部分代码，即可实现新地图的产生。

3.4 输入项

图 3-7 输入项图示

表 3-8 各种货车属性表

货车规格	额定功率（千瓦）	每公里耗油（元/公里）	自重（吨）
小型	120	0.08	2
中型	160	0.12	5
较大型	200	0.16	8

（1）需求量（吨）（精度：>10 位小数）：

计算公式：速度=额定功率÷（需求量+货车质量）+波动值

同时因为需求量的不同，影响所选货车规格不同。其中需求量 0~4 吨选择 2 吨自重的小型货车，4~8 吨选择自重 5 吨的中型货车，8~12 吨选择自重 8 吨的较大型货车。利用<stido.h>下的 atof()函数进行 CString 到 double 的转换参与计算。

CString	m_NeedEdit	编辑框	利用 UpdateData(TRUE)获取需求量

（2）燃油费（元/升）（精度：>10 位小数）：

计算公式：配送成本=燃油费 X 每公里耗油量 X 总里程+时间损失费+其余费

CString	m_FuelEdit	编辑框	利用 UpdateData(TRUE)获取燃油费

（3）其余费（元）（精确到个位）：

计算公式：配送成本=燃油费 X 每公里耗油量 X 总里程+时间损失费+其余费

所谓其余费是指在整个配送过程所要产生的费用，例如路况费用、服务费用、通行费用、意外保险费用、参与配送流程员工薪水等费用的总和

CString	m_OtherEdit	编辑框	利用 UpdateData(TRUE)获取其余费

（4）方案选择：

将两个单选按钮合成一个组，然后为他们关联变量 m_MinRadio 来表征选择的方案。

其中-1 表示未选择，0 表示选择最短路径方案，1 表示选择最短时间方案。

int	m_MinRadio	单选按钮	利用 UpdateData(TRUE)获取按钮选择

3.5 输出项

图 3-8 输出项图示

（1）总里程（千米）（精度：2 位小数）

计算公式：总里程=Head->id÷比例尺

其中 Head->id 是 int 型的整型值，模拟的是图上的设备坐标像素值。而比例尺是“图上每像素：实际千米数=30：1”。总里程的含义是指从仓库出发依次配送到各个客户后再返回仓库的总千米数。

double	D	总里程，由 Head 路线总长和比例尺计算得到

（2）配送成本（元）（精确到个位）

计算公式：配送成本=燃油费 X 每公里耗油量 X 总里程+预计时间 X 每小时损失费+其余费

其中每小时损失费是指在实际配送过程中由于未能在预期的时间给客户配送到，所遗留下来的服务质量价值费用，是一个虚构的权值。

double	S	配送成本

（3）预期时长（小时）（精度：1 位小数）

计算公式：预期时长=∑（各路段里程数÷各路段对应速度）

double	T	预计时长

（4）速度（千米/时）（精度：2 位小数）

计算公式：速度=额定功率÷（需求量+货车质量）+波动值

其中平均速度大致是由额定功率、载重、路况决定。而波动值只

由路况决定，流畅时，波动值在-7~7km/h 间波动；拥挤时，速度波动值在-1.4~1.4km/h 间波动。

double	P	汽车额定功率，由需求量决定卡车规格不同
double	M	汽车质量，需求量与卡车质量总和
double	V	行驶速度

（5）路况（流畅、拥挤）

根据速度的不同显示不同的路况信息。

CString	m_TrafficStr	静态文本	利用 UpdateData(FALSE)显示路况

（6）像素坐标

当用户的鼠标移动到地图区后，右上角出现以对话框左上角为原点，向右为 x 轴，向下为 y 轴的以像素为单位的坐标值。

（7）模拟行进（红线移动）

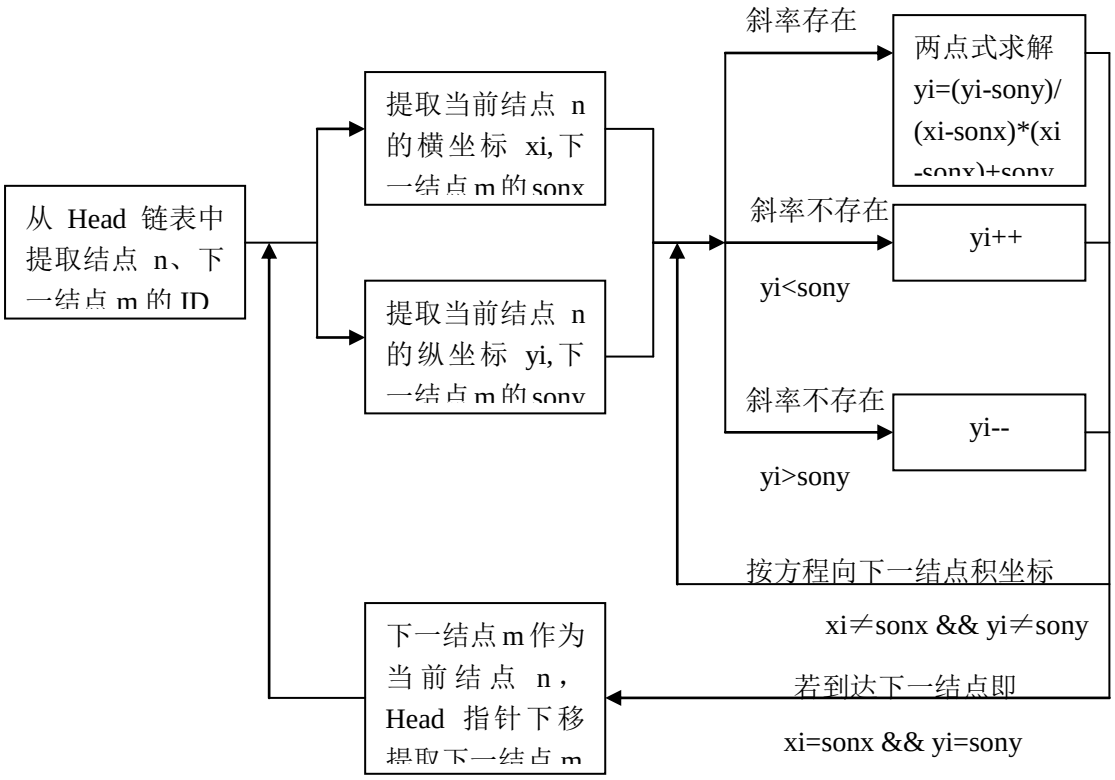


图 3-9　模拟行进图示

此画红线的函数单独写于 DrawLine()函数中，下面是伪代码:

```
画当前所在点 xi,yi
if(xi < sonx) //横坐标从左边逼近
{
    xi++;
    yi=((yi-sony)/(xi-sonx))*(xi-sonx)+sony;
}
if(xi > sonx) //横坐标从右边逼近
{
    xi--;
    yi=((yi-sony)/(xi-sonx))*(xi-sonx)+sony;
}
else    //斜率不存在
{
    if(yi < sony) //纵坐标从下面逼近
   {
        yi++;
    }
    else if(yi > sony) //纵坐标从上面逼近
    {
        yi--;
    }
```

} 画下一时刻的点

（8） 图形及音效

（1）所用的到图片包括：地图（百度地图）、选中地图（Firework8 处理）、仓库旗帜（Photoshop 透明处理，显示时用到 TransparentBlt() 函数）、客户旗帜、配送到客户旗帜、图标。具体请见 res 文件夹。

（2）音效的产生包含了#pragma comment(lib,"winmm.lib")多媒体库，然后用 PlaySound()实现异步播放配送到客户的音效“Been.wav”。其次还用了<windows.h>头文件中的 Beep()蜂鸣函数产生右键选择时的音效。

3.6 算法

（1）改进 A 星算法——解决任意起点与终点间的最短路线问题。

①算法简介

A 星算法是一种静态路网中求解最短路最有效的方法。A 星通过改变它自己行为的能力基于启发式代价函数，能够使程序运行得更快。由于 A 星算法并非随机算法，因而得到的最短路线每次均是是一个确定值。

②算法核心

计算公式：f（n）=g（n）+h（n）

其中 f(n)是从初始点经由结点 n 到目标点的估价函数，g(n)是

状态空间中从初始结点到 n 结点的实际代价，h(n)是从 n 到目标结点最佳路径的估价。因此，算法快慢与否和找到最优解的与否很大程度上取决于所用的 h(n)估价函数。

这样以来，就可以对地图上的结点进行扩展。其方法是建立 ASTAR 类的名为 OPEN 表（用于装扩展出的结点，未作为中心结点的结点），CLOSE 表（用于装已经作过中心结点的结点）。从仓库开始，选取 OPEN 表中 f(n)最小的结点作为中心结点（当 f(n)相同时，比较他们的 h(n)大小）扩展。算出该点所有相邻结点的启发值 f(n)。还有最重要的是，将这些相邻结点中的成员变量 father 全部指向该中心结点（当然如果某个相邻结点在之前的搜索过程中已经指向了另外的中心结点，那么就要比较两种情况下的 f(n)大小，选取较小者指向），最后把所有相邻结点的信息加入到 OPEN 表中，同时从 CLOSE 表中删除该中心结点。而这样的查找的结束条件是，当遇到目标节点退出。

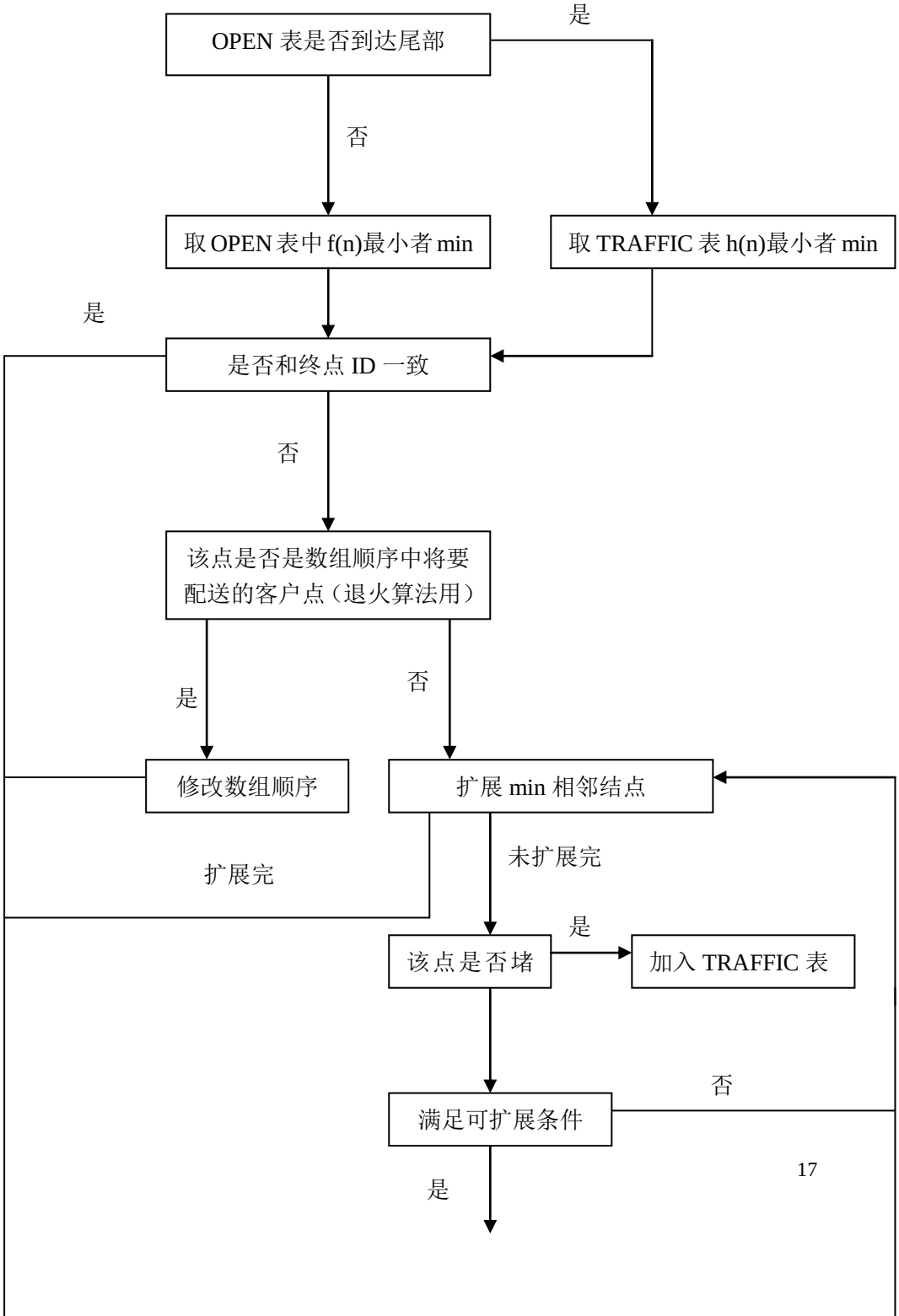
这样以来，通过目标结点中的 father 就能推知其父结点的 ID，通过其父结点中的 father 就能继续推知更上层的父结点的 father 的 ID。这样一直回溯到起点为止。而这个回溯过程则需要利用递归的思想。

③改进方面（主要是针对堵车问题）

我组对该算法的改进方面在于加入了 TRAFFIC 表，此表加入中心结点扩展出的所有拥堵的结点。而其用途在于解决当 OPEN 表到达表尾。（即当前扩展出的所有结点已经遍历完毕，换句形象的话说就是堵车路段阻断了所有汽车能够绕开的道路让其“无路可走”）那么这

时就只有从之前加入 TRAFFIC 表中的堵车结点中取 h（n）最小的结点作为中心结点扩展。（之所以选择比较他们的 h（n）而非 f（n）的原因就在于动态规划时，汽车目前已经到达了一个离终点较近的地方，如果比较 f（n）很可能路线会返回再重走一遍刚才的路，这样就不能达到目的）

④流程逻辑（如图 3-10）



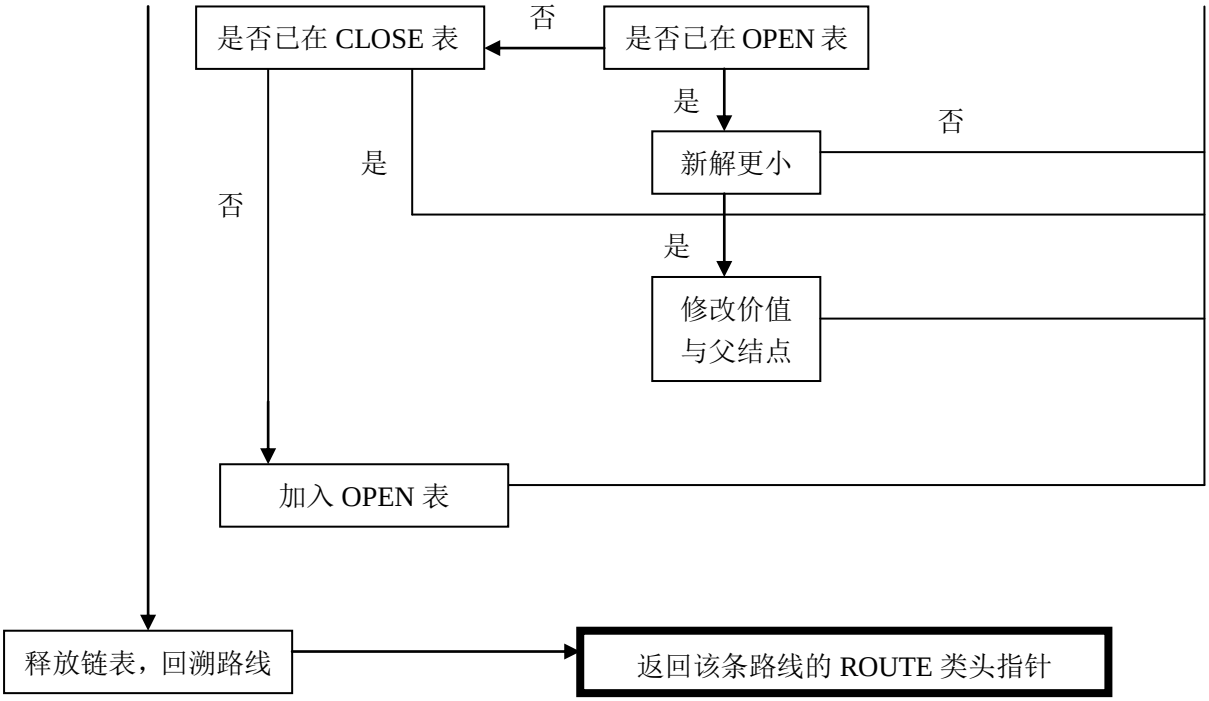


图 3-10　流程逻辑图示

⑤具体实现（伪代码）

```
创建三个链表分别为 OPEN 表，CLOSE 表，TRAFFIC 表
OPEN 表中首先存入仓库结点的 ID
while(TRUE)
{
    if(OPEN 表达到表尾)
    {
        在 TRAFFIC 表中找出 h(n)最小的结点作为中心结点 min
    }
    找出 OPEN 表中 f(n)最小的结点作为中心结点 min
    if(min ==  客户)
    {
```

```
        加入 CLOSE 表中
        退出循环
    }

    while(未到数组的结束标志)
    {
        if(起点和终点客户间存在其他顺路客户)
        {
            修改数组顺序
            退出循环
        }
    }

    while（min（中心结点）的相邻结点未扩展完）
    {
        if(该相邻结点属于堵车结点  &&  该结点不在 CLOSE 表中)
        {
            将该结点加入到 TRAFFIC 表中
        }

        if(最短路线方案
        ||  最短时间方案&&该结点不堵车
```

```
|| min 结点无路可走
||  该结点无路可走
||  该结点就是需配送客户)
{
    if(该结点已经在 OPEN 表中)
    {
        计算新的 f(n)
        if(新 f(n) <  旧 f(n))
        {
            修改 f(n)
            修改其父结点
        }
        取下一个结点
    }

    if(该结点在 CLOSE 表中)
    {
        取下一个结点
    }

    if(该结点既没在 OPEN 表也没在 CLOSE 表)
   {
```

```
                计算 f(n)、g(n)
                加入 OPEN 表
            }
        }
        取下一个结点
    }
    释放 OPEN、CLOSE 表
    回溯路线
    返回 ROUTE 路线头指针
```

（2）改进模拟退火算法、遗传算法结合——解决仓库与多个客户地址间巡回最短路线问题。

①算法简介

模拟退火算法来源于固体退火原理，将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

遗传算法是一类借鉴生物界的进化规律（适者生存，优胜劣汰遗传机制）演化而来的随机化搜索方法。

我组结合两者的优点，互补两者的缺点。以模拟退火算法为主，遗传算法为辅。

表 3-9 两种算法涉及的术语

名称	标识符	原型含义	影响
起始温度	T	循环变量的初值	T 越高、T_min 越小、r 越大（0<r<1）迭代次数越多，找到最优解的概率越大，但搜索速度变慢
温度下限	T_min	退出循环的条件	
退火速度	r	退出循环的速度	
能量差	dE	新旧解的路程差	dE<0 接受新解,dE>0 以一定几率接受新解
变异差	de	新、记忆解路程差	de<0 则变异机会增 1，否则减 1
变异	Product()	调换数组顺序	数组变化规则越符合概率统计、离散规律越高效、易得到最优解
淘汰	stand	变异机会	让恶性变异适可而止，增强效率
重升温	times	同文件比较次数	让搜索适可而止，增加速度
旧解	old	上一次路线链表	同新解作差得能量差
新解	fresh	当前路线链表	同旧解作差得能量差
记忆解	remember	暂时舍弃的父亲路线链表	同新解作差得变异偏移量。且当新数组有恶性变异趋势时，将记忆的链表赋予
当前数组	order	当前数组顺序	可以作为形参传递给给 Cool()模拟退火算法函数产生对应路线链表
记忆数组	order1	变异前数组顺序	

②算法核心

退火算法需要一个 while()循环，其终止条件是经过数次迭代之后达到最低温度。那么初始化起始温度、温度下限、退火速度就成了算法的关键。比如起始温度越高则找到最优解的概率就越大，但搜索速度相应变慢。因此，控制好初始化条件是这个算法的难点与重点。我组通过多次对找到最短路径概率的统计最终决定以 T=5000;T_min=10;r=0.99;为初始条件。

在循环开始处，需要用利用遗传算法中的变异（Product()函数）产生新解（即新的数组顺序），然后新解同旧解生成 dE、记忆解同新解作差生成 de。若 dE、de 均说明 fresh 更短，则将 fresh 与文件中历史最短解比较，小则写入文件。其次，若 de 小于零，说明新解的路程大于记忆解的，那么以概率统计的角度，大致说明该新解有朝着恶性变异的方向的趋势，更容易被淘汰即 stand++，减少其变异次数。

达到 5 次则赋予记忆解，终止恶性变异。但如果 de 大于零，则证明可能是优良变异，于是 stand--。这就是遗传算法的融入，也是本算法的改进方面。

以上只是准备工作，下面演示核心算法的过程。首先，判断 dE 大小，如果新解比旧解小，那么接受新解；若比旧解大，那么以狄利克雷函数 exp(-dE/T)所得的概率接受这个可能“看似较短”的新解，然后通过 Product()函数变异数组中配送客户的先后顺序可能就会得到比之前的旧解还要短的新解。

这里，Product()函数也是这个算法的精髓所在，其打乱次序的方式直接影响了最后的结果。本程序采用两种打乱方案，每种方案在不同的地形处使用。

【方案一】：经过统计，该方案适合利用在客户地址分布比较松散、遥远的场合。首先设当前客户数为 n，再产生 1~n 的两随机数 p、q：

（例：原始顺序为（51，55，57，69，56，53，71））

（1）：若 p<q，将 p、q 之间逆转。

例：p=53,q=55。变换后的顺序为（51，53，56，69，57，55，71）

（2）：若 p>q，将 q、p 以外逆转。

例：p=57,q=56。变换后的顺序为（55，51，57，69，71，53，56）

（3）：若 p=q，结合库函数 rand()将数组随机打乱。

【方案二】：经过统计，该方案适合利用在客户地址分布比较紧密、临近的场合。结合库函数 rand()将数组随机打乱。

最后起始温度 T 乘以降温的速度 r 得到当前温度。到这里，就完成了一个循环。当退出循环后，则要通过文件读写把刚才所找到的那个最后的解同文件中记录的历史最短的解进行比较，如果比其小，则不进行重升温；若比其大，则要返回重新退火一次，最多退火 5 次，然后输出结果。设置上限是避免迂回搜索。

③改进方面

根据概率统计合理初始化、设计了合理的状态产生函数即根据客户点坐标的分布形式最终决定的 Product()打乱数组的方式，避免了盲目的不适宜搜索过程导致算法的效率低下且成功率低下、设计合适的终止法则例如重升温、增加了记忆功能、增加了补充搜索过程、设计了合适的终止法则避免迂回搜索、增加了同文件中历史最短择优筛选过程。

④流程逻辑（如图 3-11）

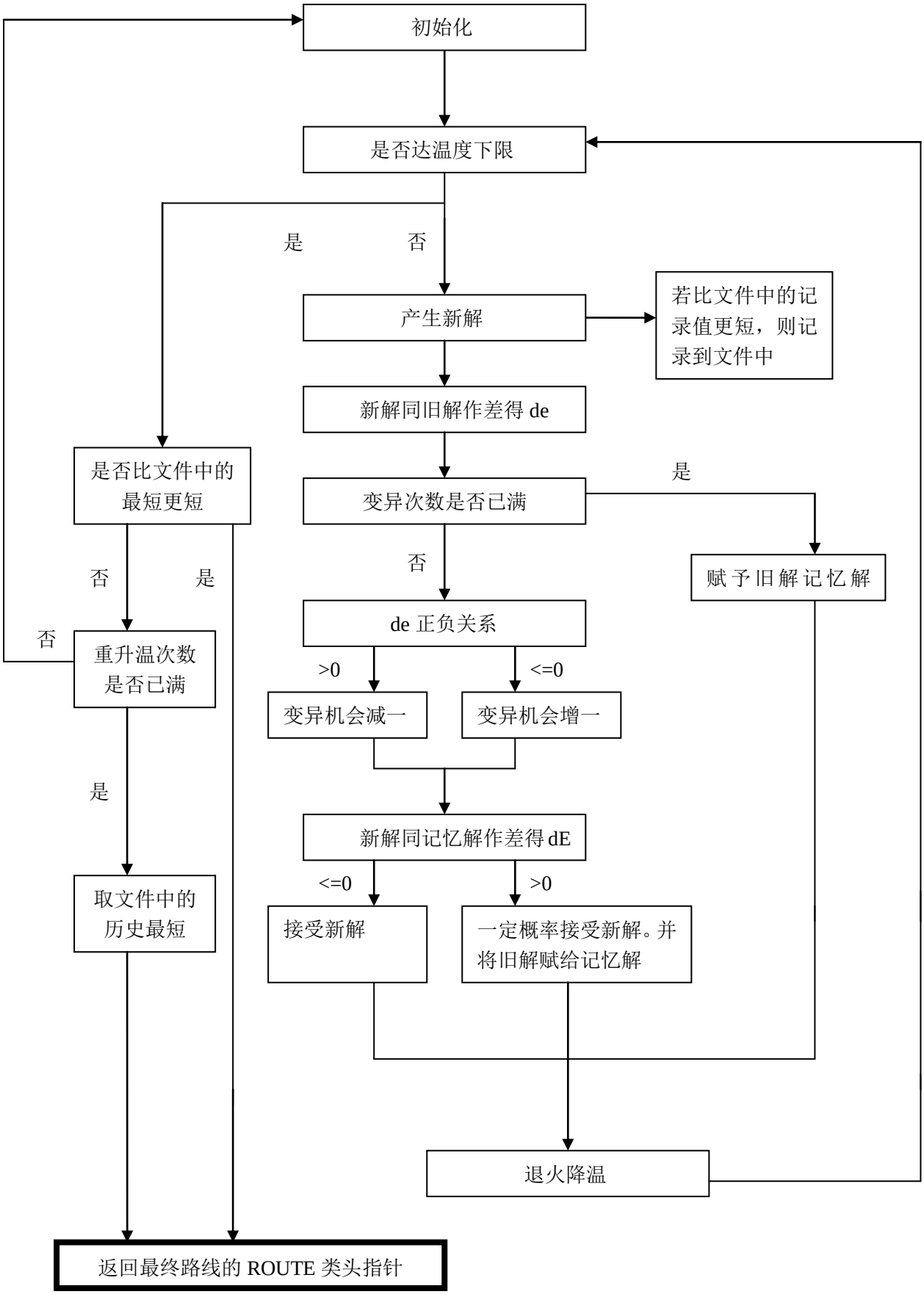

初始化
是否达温度下限
是
否
产生新解
若比文件中的记录值更短，则记录到文件中
新解同旧解作差得 de
是否比文件中的最短更短
变异次数是否已满
是
否
是
否
赋予旧解记忆解
否
de 正负关系
重升温次数是否已满
>0
<=0
变异机会减一
变异机会增一
是
新解同记忆解作差得 dE
取文件中的历史最短
<=0
>0
接受新解
一定概率接受新解。并将旧解赋给记忆解
退火降温
返回最终路线的 ROUTE 类头指针

图 3-11　流程逻辑图

⑤具体实现（伪代码）

```
各物理变量、新、旧、记忆解初始化
while(未达温度下限)
{
    产生新解
    dE=新解-旧解
    de=新解-记忆解
    if(dE < 0 && de < 0)
    {
        新解同文件中的记录值比较
        if(新解更小)  以新解覆盖文件中的记录值
    }
    if(累计恶性变异次数 <  最大次数)
    {
        if(de > 0) //新解更小
            累计恶性变异次数减 1
        else
            累计恶性变异次数加 1

        if(dE <= 0) //新解更小
        {
            接受新解
```

```
            新解数组变异
        }
        else
        {
            if(狄利克雷函数计算所得概率值在 > 0~1 的随机小数)
            {
                记住旧解
                接受新解
            }
            新解数组变异
         }
    }
    else //终止恶性变异的数组
    {
        变异机会重新赋予
        赋予旧解记忆解
    }
    降温退火
}
```

（3）动态规划

①情景描述

汽车在行进途中，当遇到前方堵车，能绕开堵车路段而动态规划出一条新的最短路径以节约时间（我组只考虑绕开堵车路段，因为汽车堵车时的平均速度一般在 1m/s 左右，根据离散数学与概率统计依据，只要道路长度达到一定，不绕开道路所用时间与绕开道路所用时间之比成指数型增长。换言之，绕开道路可能比不绕开而选择堵车路段中路段长度最短的那条所用的时间更短）。

②算法核心

为解决此问题，我组总结出一种算法：首先从全局 ROUTE 指针 Link 处得到汽车下一次将要到达的结点 ID，再找出此结点到“待配送的客户结点”中最近的那个客户结点，调用 A 星算法即 FindRoute()函数算出他们俩的最短路线作为汽车的局部新路线。此路线走完之后，汽车应该就到达了先前提到的客户结点。此时只要提取出所有没有配送到的客户结点 ID，将他们组成一个新数组，然后再次调用改进模拟退火算法即可得剩余的路线。然后将上述两条路线连接即可得到接下来的新路线。至此，一条新的最短路线就修改完成。

③流程逻辑

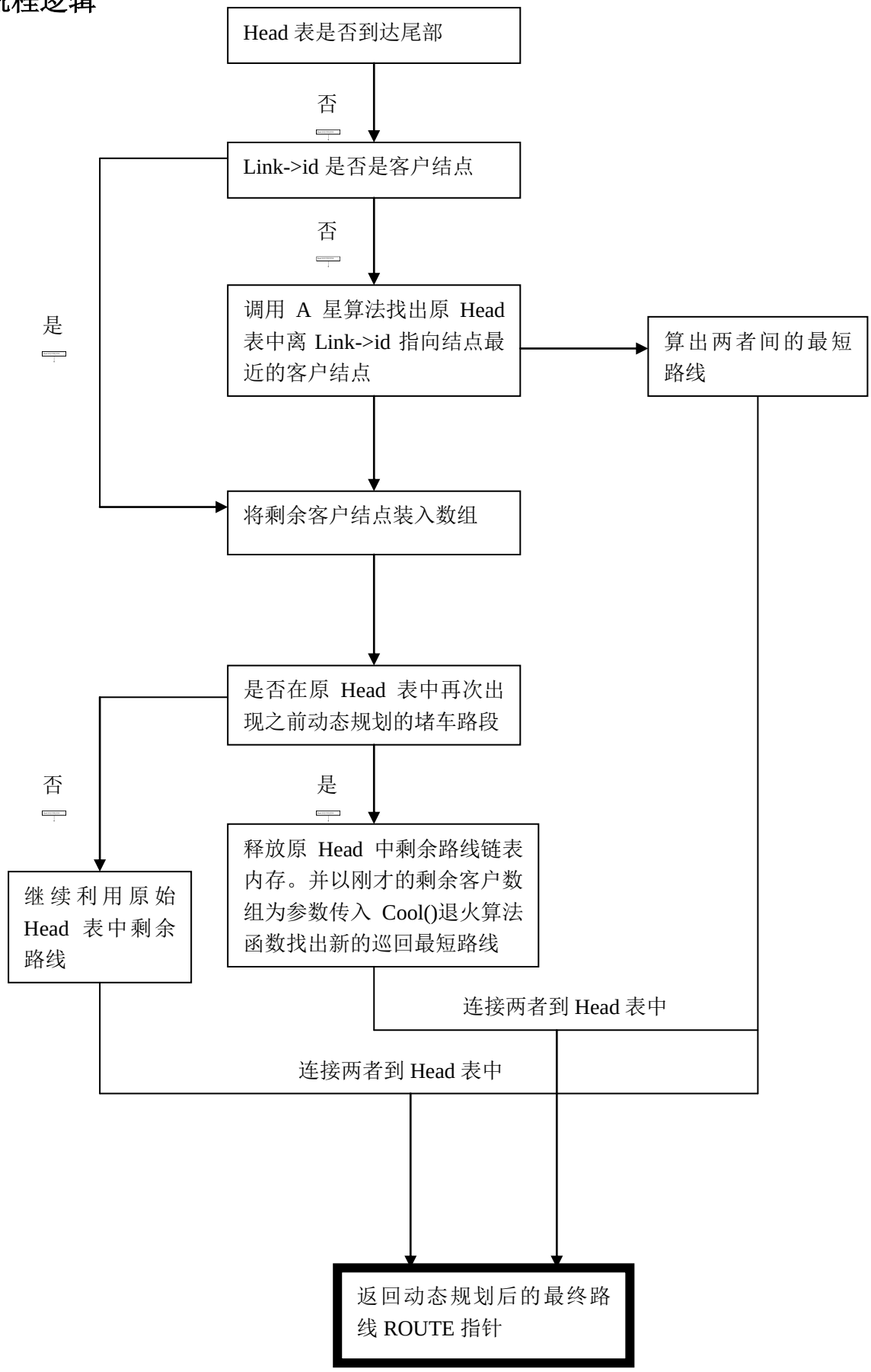

Head 表是否到达尾部
否
Link->id 是否是客户结点
否
是
调用 A 星算法找出原 Head 表中离 Link->id 指向结点最近的客户结点
算出两者间的最短路线
将剩余客户结点装入数组
是否在原 Head 表中再次出现之前动态规划的堵车路段
否
是
继续利用原始 Head 表中剩余路线
释放原 Head 中剩余路线链表内存。并以刚才的剩余客户数组为参数传入 Cool()退火算法函数找出新的巡回最短路线
连接两者到 Head 表中
连接两者到 Head 表中
返回动态规划后的最终路线 ROUTE 指针

图 3-12　流程逻辑图

④具体实现（伪代码，在 ChangeCircle()函数中编写）

```
while(Head 链表未到表尾)
{
    找出原 Head 表中离 Link 所指向的下一个将要到的结点紧接着的客户结点
}
获取所有剩余的客户结点装入数组
if(Link->id 下一个结点是普通的道路结点)
{
    A 星算法找上述两结点间最短路线
}
if(动态标记的路段在 Head 剩余链表中继续出现)
{
    以上述所到达的客户结点为起始点，调用改进模拟退火算法找出新巡回最短路线
    连接 A 星算法找到的路线与新巡回路线组合成新的路线
}
```

（4）手动测试

①情景描述

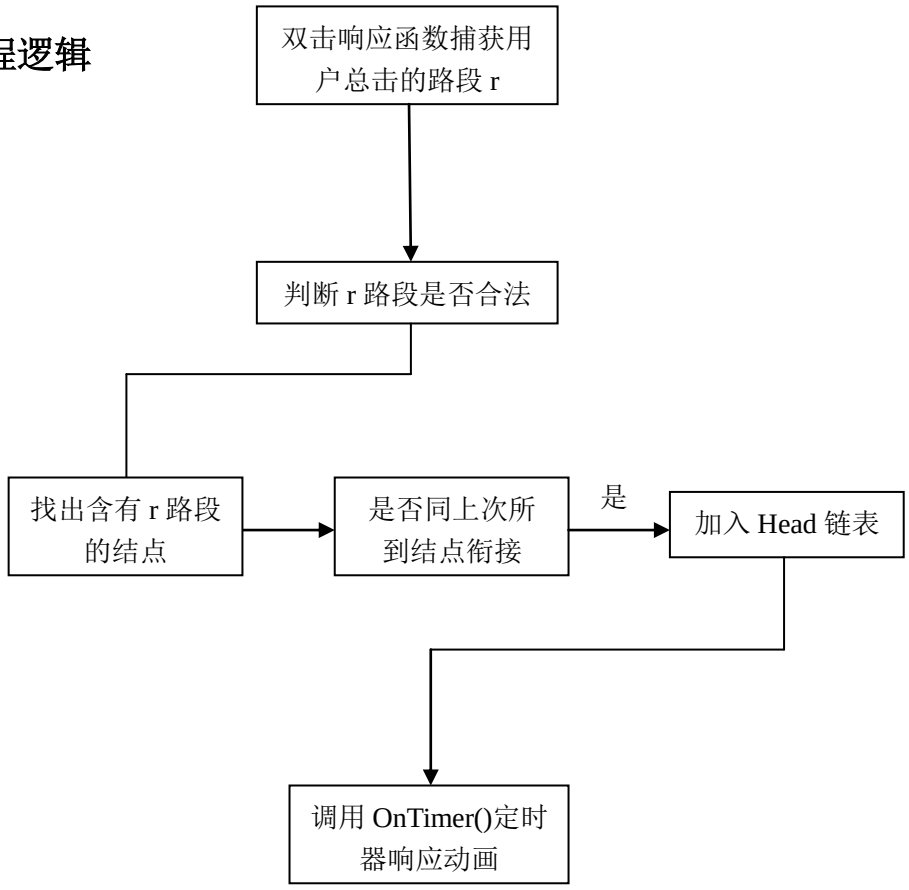
用户可以在找到了一条合适的路线以后，或者是直接标注完客户、仓库地址、输入信息等后，直接按“测试”按钮进行手动测试。以侧

面应证系统搜索结果的准确性。使用方式是：用户从仓库开始，朝着下一次，或者下下次将要到达的道路“双击”，如果小车所在路段同用户所点的路段能够衔接，则小车将到达用户点击的位置（红线模拟）。与此同时，状态栏将显示所到之处的各种信息。

②算法核心

首先利用父类对话框的类向导设置双击响应事件以构成函数。其次，在该函数中识别出用户所点的路段标号。将该标号传递给JoinTable()函数判断所双击的道路是否合法，若合法则通过所点道路寻找到它对应的下一次将要到达的结点，并将该结点添加至 Head 全局链表中。同时，开启定时器，响应红线移动操作。

③流程逻辑



④具体实现（伪代码，在 JoinTable()函数中编写）

```
for(遍历所有结点)
{
   for(遍历当前结点的所有相邻道路)
   {
      if(   当前道路 = = 用户双击道路
        &&当前结点同上次所到结点衔接吻合
        &&(当前结点是客户 || 当前结点是仓库)
        将该结点添加至 Head 链表;
   }
   SetTimer()响应 OnTimer()定时器中的红线移动操作;
}
```

3.7 接口

（1）硬件接口：PC 机、1G 内存、320G 硬盘、鼠标接口、打印机

（2）软件接口：多媒体相关应用程序接口"winmm.lib"、安装 mfc42d.dll　MFCDLL 共享库文件、安装 VC 可视化开发工具、在 Windows2000 及以上的操作系统中运行.

3.8 注释设计

（1）设计 1：用来分隔大模块，且同时表示该大模块功能的注释。

```
/*****************************************************/
/*函数声明*/
```

/***/

（2）设计 2：按语句段作用分隔，且同时表示语句段功能的注释。

/************************/

/*图像处理*/

/************************/

（3）设计 3：普通注释。

//标志定时器关闭

（4）设计 4：函数间的分隔。

/***/

3.9 限制条件

（1）系统必须存在的人工处理过程

比如一些信息需要人工手动输入、点击。其次用户需要熟悉所有按钮的逻辑顺序。比如：任何时刻可以点击重置按钮、确定按钮只能在未标记的时候可以点击等。

（2）地图变化问题

①地图需要程序员录入数据信息。

②背景地图等需要做相应调整。

③鼠标捕获的坐标需要做相应调整。

3.10 测试计划

（1）同手动测试路线对比

用户按照自己主观所想的路线双击道路，在状态区得到的数据同智能搜索得到的数据进行对比。

（2）同测试文档中数值演算的结果对比(详见测试文档)

（3）同题目要求的各项功能进行对比

包括随机标注仓库客户地址、采用 Windows 设备坐标系、距离采用设备坐标像素间距模拟、坐标之间行驶速度采用随机算法生成、用户选择路径规划策略、动态规划绕开堵车路段等。

3.11 尚未解决的问题

（1）随机算法产生的问题

改进模拟退火算法与遗传算法本身是随机算法，不一定每一次都能找到最优解即最短路径。而系统目前采用“重找”按钮模拟生物进化的特点进行筛选更短的路线。（具体请见测试文档）

（2）与真实数据的差距

模拟软件中的数据和实际生活中的真实值存在一定的差距。

（3）多辆车配送

未能实现根据需求吨数分配车辆数目进行配送。