遗传算法应用的分析与研究

福州八中 钱自强

【摘要】

随着科技水平的不断发展，人们在生产生活中遇到的问题也日益复杂，这些问题常常需要在庞大的搜索空间内寻找最优解或近似解，应用传统算法求解已经显得相当困难。而近年来，生物学的进化论被广泛地应用于工程技术、人工智能等领域中，形成的一类有效的随机搜索算法——进化算法，有效的解决了诸多生产生活中的难题而显得越来越流行。

本文的首先将介绍进化算法的原理以及历史使大家对进化算法有一个初步的了解，其次将详细介绍应用遗传算法解题的步骤，并提出有效改进和应用建议。紧接着通过一个 NP 难题的优化实例让大家对遗传算法有更深刻的了解，最后通过数据分析证明其方法的有效性。

【关键词】

人工智能；进化算法；遗传算法（GA）；多目标最小生成树

目录

1. 进化算法理论

1.1 进化算法概述

从远古时代单细胞开始，历经环境变迁的磨难，生命经历从低级到高级，从简单到复杂的演化历程。生命不断地繁衍生息，产生出具有思维和智能的高级生命体。人类得到生命的最佳结构与形式，它不仅可以被动地适应环境，更重要的是它能够通过学习，模仿与创造，不断提高自己适应环境的能力。

进化算法就是借鉴生物自然选择和遗传机制的随机搜索算法。进化算法通过模拟“优胜劣汰，适者生存”的规律激励好的结构，通过模拟孟德尔的遗传变异理论在迭代过程中保持已有的结构，同时寻找更好的结构。作为随机优化与搜索算法，进化算法具有如下特点：进化算法不是盲目式的乱搜索，也不是穷举式的全面搜索，它根据个体生存环境即目标函数来进行有指导的搜索。进化算法只需利用目标的取值信息而不需要其他信息，因而适用于大规模、高度非线性的不连续、多峰函数的优化，具有很强的通用性；算法的操作对象是一组个体，而非单个个体，具有多条搜索轨迹。

1.2 遗传算法

遗传算法（Genetic Algorithm）是进化算法的一个重要分支。它由 John Holland 提出，最初用于研究自然系统的适应过程和设计具有自适应性能的软件。近来，遗传算法作为问题求解和最优化的有效工具，已被非常成功地应用与解决许多最优化问题并越来越流行。

遗传算法的主要特点是群体搜索策略和群体中个体之间的信息互换,它实际上是模拟由个体组成的群体的整体学习过程,其中每个个体表示问题搜索空间中的一个解点.遗传算法从任一初始的群体出发,通过随机选择，交叉和变异等遗传操作,使群体一代代地进化到搜索空间中越来越好的区域,直至抵达最优解点.

遗传算法和其它的搜索方法相比,其优越性主要表现在以下几个方面：首

先,遗传算法在搜索过程中不易陷入局部最优,即使在所定义的适应度函数非连续.不规则也能以极大的概率找到全局最优解,其次,由于遗传算法固有的并行性,使得它非常适合于大规模并行分布处理,此外,遗传算法易于和别的技术(如神经网络.模糊推理.混沌行为和人工生命等)相结合,形成性能更优的问题求解方法.

2. 遗传算法

2.1　遗传算法的基本流程

一个串行运算的遗传算法通常按如下过程进行：

(1) 对待解决问题进行编码；t:=0

(2) 随机初始化群体 X(0):=$(x_1, x_2, \cdots, x_n)$；

(3) 对当前群体 X(t)中每个染色体 x_i 计算其适应度 F (x_i)，适应度表示了该个体对环境的适应能力，并决定他们在遗传操作中被抽取到的概率；

(4) 对 X(t)根据预定概率应用各种遗传算子，产生新一代群体 X(t+1)，这些算子的目的在于扩展有限个体的覆盖面，体现全局搜索的思想；

(5) t:=t+1(新生成的一代群体替换上一代群体)；如果没有达到预定终止条件则继续(3)。

2．2 遗传算法中各重要因素分析

▲ 编码理论

遗传算法需要采用某种编码方式将解空间映射到编码空间（可以是位串、实数、有序串）。类似于生物染色体结构，这样容易用生物遗传理论解释，各种遗传操作也易于实现。编码理论是遗传算法效率的重要决定因素之一。二进制编码是最常用的编码方式，算子处理的模式较多也较易于实现。但是，在具体问题中，根据问题的不同，采用适合解空间的形式的方式进行编码，可以有效地直接在解的表现型上进行遗传操作，从而更易于引入相关启发式信息，往往可以取得比二进制编码更高的效率。

▲ 染色体

每个编码串对应问题的一个具体的解，称为染色体或个体。一个染色体可以通过选用的编码理论与问题的一个具体的解相对应，一组固定数量的染色体则构成一代群体。群体中染色体可重复。

▲ 随机初始化

随机或者有规律（如从一个已知离解较近的单点，或者等间隔分布地生成可行解）生成第一代群体。一次遗传算法中有目的采用多次初始化群体会使算法拥有更强的搜索全局最有解的能力

▲ 适应度

一个染色体的适应度是对一个染色体生存能力的评价，它决定了该染色体在抽取操作中被抽取到的概率。估价函数是评价染色体适应度的标准，常见的估价函数有：直接以解的权值（如 01 背包问题以该方案装进背包物品的价值作为其适应度），累计二进制串中 1/0 的个数（针对以二进制串为编码理论的遗传算法），累加该染色体在各个目标上的得分（针对多目标最优化问题，另外，对于此类问题，本文提出了一种更有效的估价函数）。

▲ 遗传算子

遗传算子作为遗传算法的核心部分，其直接作用于现有的一代群体，以生成下一代群体，因此遗传算子的选择搭配，各个算子所占的比例直接影响遗传算法的效率。一个遗传算法中一般包括多种遗传算子，每种算子都是独立运行，遗传算法本身只指定每种算子在生成下一代过程中作用的比例。算子运行时从当前这代群体中抽取相应数量的染色体，经过加工，得到一个新的染色体进入下一代群体。

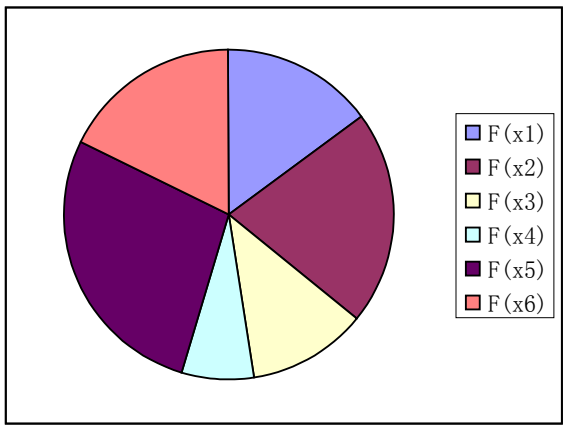
下面列出几种常见的遗传算子：

- 保持算子：抽取 1 个染色体，直接进入下一代。该算子使算法能够收敛。
- 交配算子：抽取 2 个染色体，交换其中的某些片段，选择适应度高的（或 者都选），进入下一代群体。该算子使得遗传算法能够利用现有的解寻求更优的解。
- 变异算子：抽取 1 个染色体，对其进行随机的改变，进入下一代群体。该算子使得算法可以跳出局部最优解，拥有更强搜索全局最有解的能力，防止陷入局部搜索，该算子的概率不可过高，否则将引起解的发散，使得算法无法收敛。

▲ 抽取

抽取操作是遗传算法中一个重要基本操作，作用是按照“优胜劣汰”的原则根据各个染色体的适应度从当前这代群体中挑选用于遗传算子的染色体通常采用的手段是偏置转盘：

设算法中群体数量为 $population$，首先计算当代群体的各染色体适应度之和 $S(t)=\sum_{i=1}^{population}F(x_i)$。将



1～$S(t)$内的整数划分成 $population$ 个区间段，每个染色体所占的区间段的长度既是它的适应度。这样，随机产生一个1～$S(t)$的整数，抽取该点所属区间段相对应的染色体，就可以保证任意一个染色体 x_i 在抽取操作中被抽取到的概率为 $\frac{F(x_i)}{S(t)}$ 。

▲ **终止条件**

遗传算法的终止条件用于防止遗传算法无止境的迭代下去，一般限制条件可以设为达到指定的迭代次数后终止，或当解的收敛速度低于一定值时自动终止。当遗传算法达到终止条件时，遗传算法结束，并按照要求返回中途最优的一个染色体（或所有满足条件的非劣最优解）

2．3 重要参数设置

在应用遗传算法解决问题的时候，往往需要根据实际情况的不同，对不同问题使用不同的遗传参数。在大规模的问题上，一次遗传算法的不同时期也可以设置不同的遗传参数。对遗传算法效率影响较大的参数如下：

群体大小：一代群体中染色体的数量，群体大小越大所能容纳的染色体品种也越多，越有利于搜索全局最有解，但是会下降收敛的速度，所需的时间也更多。

迭代次数：最多更新群体的次数，迭代次数的增加可以使得解收敛更精确但是所需的时间也越多，如果时间允许，采用多次初始化群体的

操作要比设置很大的迭代次数来得更高效些。

保持率：保持算子所占的比例，通常不超过 70%

交配率：交配算子所占的比例，通常不超过 50%

变异率：变异算子所占的比例，通常不超过 1%

3. 遗传算法在多目标最小生成树问题中的应用

生活中的很多问题，例如道路铺设，电网架设，网络构设等，其实都可以归结到最小生成树模型，经典的 Prim 算法和 Kruskal 算法都可以解决该问题，算法的时间复杂度都是线形的，但是现实生活中的问题往往没有那么简单，一条边上可能不只带一个权，例如一条公路的铺设道路长度还要考虑环境和人文因素，电网架设时除了考虑线路费用还要考虑架设难度，一个网络连接除了考虑网络延时还要考虑传输稳定性和安全性等，于是问题就转化为求解多目标的最小生成树问题的非劣最优解，这个问题是 NP 难的，Prim 算法，Kruskal 算法等常规算法就显得无能为力，搜索算法的复杂度却又过高。

3．1 多目标最小生成树问题

3.1.1　　最小生成树

在图论中，一个无回路（圈）的连通子图称为树。设 $T=(N, E_T)$ 是 $|N|\geq 3$ 的一个图，则下列关于树的 6 个定义是等价的：① T 连通且无回路；② T 有 $|N|-1$ 条边且无回路；③ T 连通且有 $|N|-1$ 条边；④ T 连通且每条边都是割边；⑤ T 的任两点间都有唯一的路相连；⑥ T 无回路，但在任一对不相邻的点之间加连一条边则构成唯一的回路。

设有一无向图 $G=(N,E,W)$，其中 $W=\sum_{e\in E} w_e$ 为权函数，若树 $T=(N,E_T,W_T)$ 包含了图 G 的所有顶点，则树 T 为 G 的一个生成树，其中 $W_T=\sum_{e\in E_T} w_e$ 为树 T 的权。图 G 的生成树不唯一，权和最小的生成树称为 G 的最小生成树。

3.1.2　多目标最小生成树

而当图中每个边的权值有多个时，相应的问题称为多目标最小生成树问题。该类问题的目标是找出问题的所有非劣最优生成树，显然该类问题即使不加任何约束条件也属于NP问题。

设有一个连通的无向图 $G=(V, E)$，其中 $V=\{v_1, v_2, \cdots, v_n\}$ 是一个有限集合，代表图 G 的顶点，$E=\{e_{1,2}, e_{1,3}, \ldots, e_{i,j}, \ldots, e_{n-1,n}\}$，

$$e_{i,j}:\begin{cases}1, v_i, v_j\text{之间有边}\\ 0, otherwise\end{cases},\quad (i=1,2,\ldots,n-1; j=i+1,\ldots,n)$$ 为图 G 的边的集合，

若边 $e_{i,j}$ 存在则该边有 m 个值为正数的属性与之对应，用 $w_{i,j}=\{w_{i,j}^1, w_{i,j}^2, \ldots, w_{i,j}^m\}$ 表示，实际问题中 $w_{i,j}^k(k=1,2,\ldots,m)$ 可以是距离、代价等等。

令　　$x=(x_{1,2}x_{1,3}\ldots x_{i,j}\ldots x_{n-1,n})$，　　$x_{i,j}:\begin{cases}1, if\ e_{i,j}:1\text{且被选中}\\ 0, otherwise\end{cases}$，

$i=1,2,\ldots,n-1;\quad j=i+1,\ldots,n$，表示图 G 的一棵生成树。X 为所有 x 的集合，则多目标最小生成树问题描述如下：

$$\begin{aligned}
&\min \quad f_1(\mathbf{x}) = \sum w^1_{i,j} x_{i,j};\\
&\min \quad f_2(\mathbf{x}) = \sum w^2_{i,j} x_{i,j};\\
&\cdots\\
&\min \quad f_m(\mathbf{x}) = \sum w^m_{i,j} x_{i,j}; \quad i = 1,2,\ldots n-1, j = i+1,\ldots,n
\end{aligned}$$

其中 $f_i(x)$ 为问题中需要最小化的第 i 个目标。

与一般的最小生成树问题相比，多目标最小生成树问题只是目标函数不止一个，然而正是因为目标不止一个而且这多个目标之间是经常是相互冲突的，因此无法使用经典的最小生成树算法通过找出具有最小权值的边逐步生成最小树，而如果将多个目标转换成单个目标再应用经典算法，则只能求出一个解，而不是找到问题的一组非劣最优解，而且多目标到单目标之间的转换对决策者来也是一个难题。

3．2　应用遗传算法解决多目标最小生成树问题

3．2．1 编码设计

Cayley 证明了对于一个完全图 G，连接所有 n 个顶点的树有 n^{n-2} 棵。为此 Prüfer 建立了一个这些树与从 n 个数取 n-2 个数的所有组合之间的一一对应关系，即如果对完全图中所有顶点从 1 到 n 开始编号，则任意一个在从 1 到 n 的 n 个数中取 n-2 个数的组合都与唯一的一棵生成树相对应。

本文对生成树的编码采用基于以上的一一对应建立起来的 Prüfer 数编码机制，把每一棵树与一个长度为 n-2 的数字串对应，而对于任意一个长度为 n-2 的数字串也与唯一的一棵生成树相对应，生成树到数字串的编码与数字串到生成树的解码的详细证明可参考相关文献，本文这里只作简要描述。

★编码过程

▲ 编码串初始为空串

▲ 令 j 为树中编号最小的叶节点；

▲ 找到唯一于 j 相邻的点 i，把 i 加入编码串的最右端

▲ 把 j 以及连接 i 和 j 的边从树中删除，这时候树只有 n-1 个顶点

▲ 重复以上 3 个步骤直到树中只剩下一条边这时候得到的编码串即为相应树的 Prüfer 编码

★解码过程

▲设 P 为编码串，$\overline{P}$ 为图的顶点编号不出现在 P 中的顶点的集合；
▲设 i 为 $\overline{P}$ 中编号最小的顶点，j 为 P 中最左端的顶点，则将连接 i 和 j 的边加入到树中，然后分别把 i 和 j 从 P 和 $\overline{P}$ 中删除，如果 P 中不在出现顶点 j 则把 j 加入到 $\overline{P}$ 中
▲重复以上步骤，直到 P 为空；
▲当 P 为空串时，$\overline{P}$ 中刚好剩下两个顶点，将连接这两个顶点的边加入到树中，最后构成的树即为与最初 P 对应的生成树。

例如，图 3-1 则为一棵生成树以及其相对应的 Prüfer 编码图 3-2。

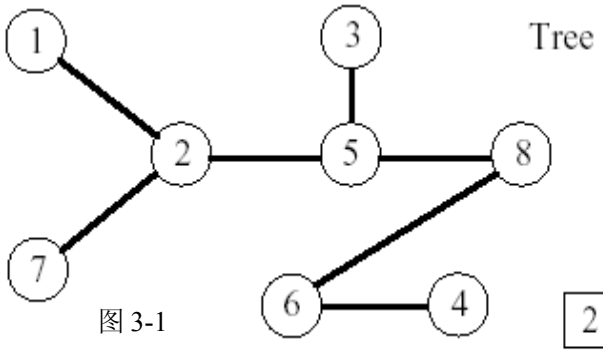


图 3-1

Tree

2	5	6	8	2	5

Prüfer number

图 3-2

显然 Prüfer 编码是生成树的一个有效表示方式，应用该编码方式，可以很容易地随机生成一棵生成树，而且 Prüfer 数编码串的每个位置的信息量又相对均匀因此很适合遗传操作。

3.2.2　估价函数设置

定义估价函数 g(x)为 $\left[\sum_{i=1}^{k}\left(\frac{\min[i]}{f_i(x)}\times 100\right)^2\right]$，$f_i(x)$ 表示当前的染色体在目标 i 的费用情况，min[i]表示截止到上一代为止，产生的所有染色体在目标 i 的费用的最小值。

本文提出的这样定义比起常见的直接累加各目标上的权值的好处在于，其不仅很好的体现了一个染色体在各个目标上的优势，与此同时还避免了由于每个目标的取值范围不同或者取值的整体趋势不同而造成的某些个体在某些目标的优势无法被体现，使得算法能够适应现实生活中各类问题。

3．2．3　遗传操作定义

★交配算子

交叉算子我们使用的是小片段等位交叉算子，随机在两个染色体中抽取等位的，长度不超过 2 的片段进行互换，然后选择适应度较高的进入下一代，具体操作方法如下图 3-3 所示：

Parent 1	2	5	6	8	2	5
Parent 2	8	4	5	2	8	7
Offspring 1	2	5	5	2	2	5
Offspring 2	8	4	6	8	8	7

图 3-3 小片段等位交叉算子示意图

★变异算子

变异算子采用常规的单点变异，即随机生成一个 1 到 n 之间的数替换 Prüfer 数编码串中的某一位，如图 3-4 所示。

Parent	2	5	6	8	4	5
Offspring	2	5	6	2	4	5

图 3-4 单点变异示意图

可以看出单点变异也可以很大程度上保留了原染色体的性质。

3.3　测试

我们将保持率定为 54%，交配率定为 45%，变异率定为 1%，并且根据数据不同对迭代次数和群体大小进行调整，将其于原始搜索算法进行对比（注：为了能够直观看出遗传算法的近似程度，所有数据采用 2 目标）。

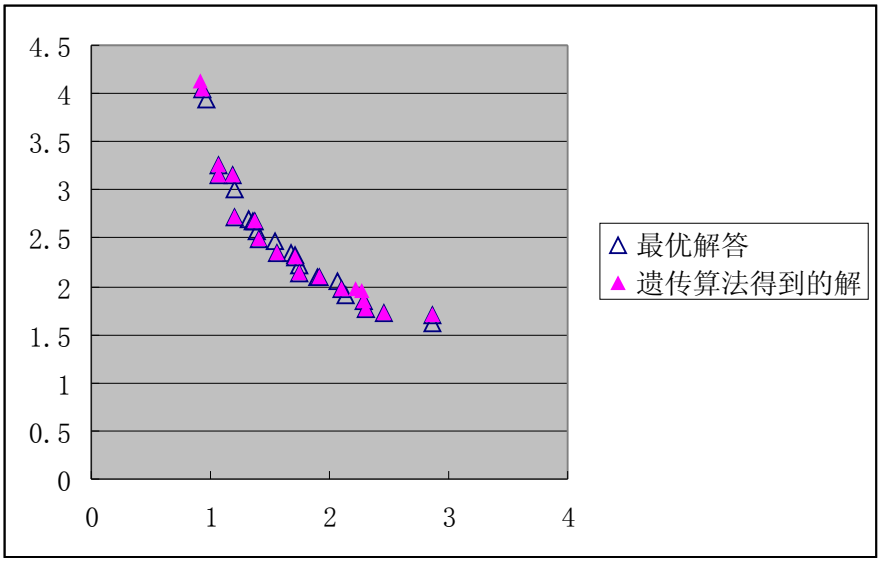
测试环境：P4 2.0GhzA 256MDDR WinXp Delphi 7.0

3．3．1 小规模经典测试数据列表

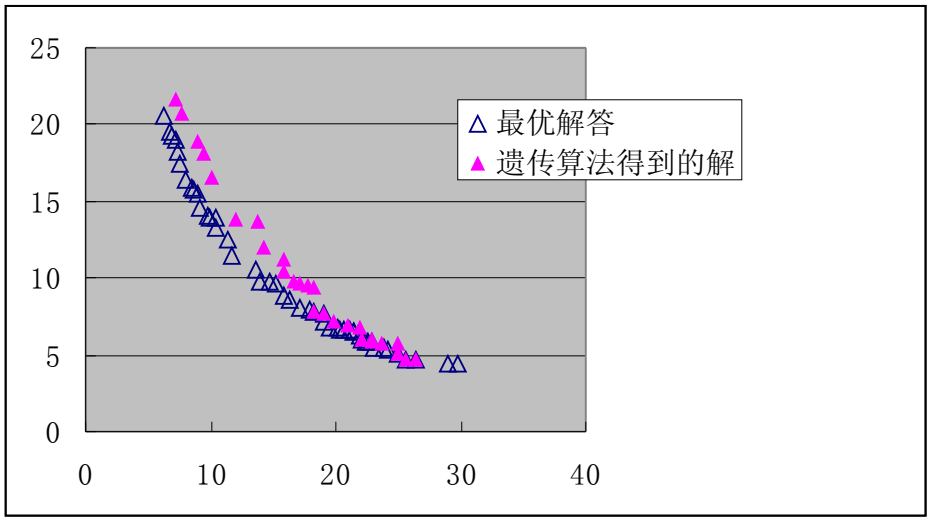
数据文件编号	数据信息	搜索算法表现		遗传算法表现		
		耗时	正确性	参数选择	耗时	正确性
E1	N=3 K=2 Ans=2	0s	完全正确	L=20 P=10	0s	完全正确
E2	N=4 K=2 Ans=6	0s	完全正确	L＝100 P=30	0s	完全正确
E3	N=5 K=2 Ans=12	0s	完全正确	L=1000 P=100	2s	完全正确

3．3．2 大中规模随机测试数据列表

数据文件编号		H1	H2
数据规模		N=10 K=2	N=11 K=2
搜索算法表现		22 分钟	6 小时 20 分钟
遗传算法表现	参数选择	L=20000 P=400	L=30000 P=400
	耗时	116s	296s
	正确性	算法得到了一组十分近似的解，参见图示	算法得到了一组比较近似的解，参见图示



测试数据 h1 的解答分析



测试数据 h2 的解答分析

3．3．3 特殊测试数据分析

对于 2 目标最优化考虑到现实生活中问题的 2 个目标的费用之间可能具有的一些关系，例如当设备相似时某个网络链接的速度相对较高，那么它的稳定性一定较低，因此我们对正相关和反相关两种特殊情况进行了测试，在这里先给出正相关与反相关的简单定义：

【正相关】

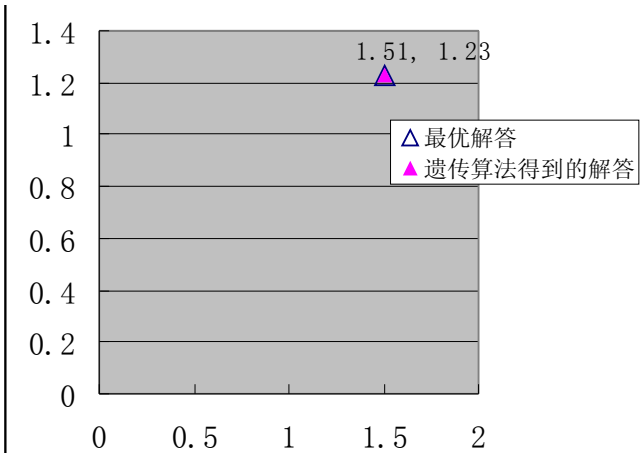
如果对于图中的任意两条边 $e_{i1,j1}, e_{i2,j2}$ 满足如果 $W^1_{i1,j1} \geq W^1_{i2,j2}$ 成立，那么 $W^2_{i1,j1} \geq W^2_{i2,j2}$ 成立，则称这张图的权为正相关的。

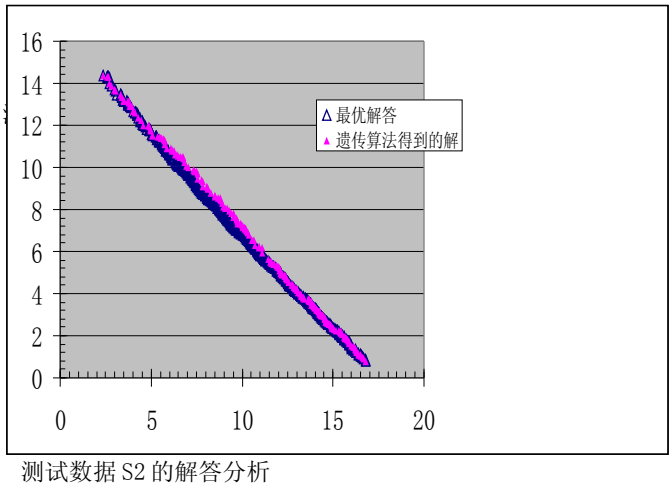
【反相关】

类似的定义，如果对于图中的任意两条边 $e_{i1,j1}, e_{i2,j2}$ 满足如果 $W^1_{i1,j1} \geq W^1_{i2,j2}$ 成立，那么 $W^2_{i1,j1} \leq W^2_{i2,j2}$ 成立，则称这张图的权为反相关的。

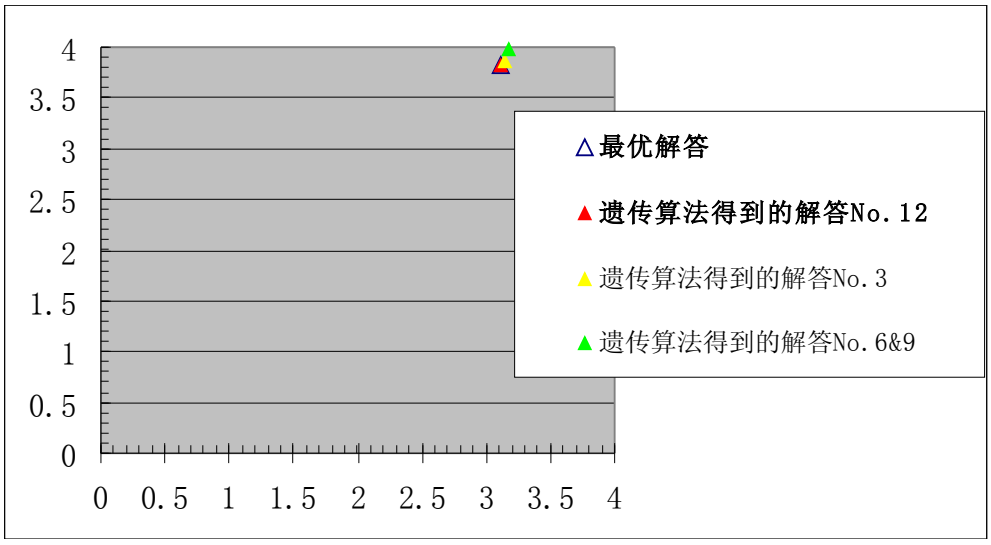
特殊数据测试表

数据文件编号		S1	S2	S3
数据信息	规模	N=10 K=2	N=10 K=2	N=15 K=2
	特殊性	正相关	反相关	正相关
搜索算法表现		15 分钟	28 分钟	600 年（估计）
遗传算法表现	参数选择	L=2000 P=400	L=20000 P=400	L=20000 P=400
	耗时	11 秒	304s	114 秒*15 （15 次运行）
	正确性	该算法在 5 次执行有内 80%（4 次）的概率得到该最优解	算法得到了一组比较相似的解，参见图示	该程序在第 12 次运行的时候得到了我们的最优解(3.11,3.83)，并且值得一提的是在第 3 次运行时就得到一个相当接近的近似解(3.14 3.86)，并且在第 6 次和第 9 次都出现了(3.18,3.99)这个不错的解





测试数据 S2 的解答分析



测试数据 S3 的解答分析

四．结束语

本文主要介绍了遗传算法时的一些基本知识和一些使用心得，以及通过测试结果让大家看到了遗传算法在解决组合优化类问题有着和其他算法无法比拟的强大优势。它的特点就是可以在较短的时间内，得到比较令人满意的解，而且算法相对简明。对于现实生活中的大量常规算法无法解决问题，遗传算法都有着良好的应用前景。

遗传算法不仅一种算法，更是一种思想。在搜索中通过灵活的运用进化的思想来解决问题，往往能够收到事半功倍的效果。目前遗传算法在信息学竞赛中还不是那么普遍。本文的目的就是希望越来越多的信息学爱好者了解遗传算法，了解进化算法的思想。

由于作者能力有限，文中难免有所疏漏，欢迎来信指正。

参考文献

陈志平 徐宗本　　《计算机数学》　　科学出版社

周　明 孙树栋　　《遗传算法原理及应用》 国防工业出版社

邵军力 张　景　魏长华　《人工智能基础》 电子工业出版社

附录：

遗传算法求解多目标最小生成树问题程序源代码：

```
Program Ga;

const

    MaxN          =    50;

    MaxK          =     5;

    PopulationMax =   1000;

    Maxans        =   1000;
```

```
     Population    =   40;
     Live          = 2000;
     Change        =    45;
     Suddenly      =     1;
 type code=array[1..MaxN]of integer;
      vector=array[1..Maxk]of real;
      note=record
            v    :vector;
            data :code;
            power:longint;
           end;
      Group=array[1..PopulationMax]of note;
var
  inf,ouf:text;
  d:array[1..MaxK,0..MaxN,0..MaxN]of real;
  ans,ans2:array[1..Maxans]of vector;
  ansb,ansb2:array[1..maxans]of code;
  sn,n,k:longint;maxv:vector;
  population,live,change,suddenly:longint;
   function Genetic_Uncode(s:code):vector;
    var temp:vector;uu:boolean;
        i,j,l,r:longint;
```

```
    s2:code;Se:set of 1..maxn;
function get:integer;
 var i:longint;
 begin
  for i:=1 to n do
   if i in Se then begin Se:=Se-[i];get:=i;exit;end;
 end;
begin
 Se:=[1..n];
 FOR I:=1 to k do temp[i]:=0;
 for i:=1 to n-2 do Se:=Se-[s[i]];
 for i:=1 to n-2 do
  begin
   l:=get;
   for j:=1 to k do
    begin
      if (l<=0)or(l>n) or (s[i]<=0 )or(s[i]>n) then
        begin L:=L+1; end;
      temp[j]:=temp[j]+d[j,l,s[i]];
    end;
   uu:=true;
   for j:=i+1 to n-2 do if s[j]=s[i] then begin uu:=false;break;end;
```

```
        if uu then Se:=Se+[s[i]];
       end;
      l:=get;r:=get;
      for j:=1 to k do
        temp[j]:=temp[j]+d[j,l,r];
      Genetic_Uncode:=temp;
     end;
procedure Input_Initial;
 var s,i,j:longint;tempcode:code;
  begin
   readln(inf,n,k);
   for s:=1 to k do
    for i:=1 to n do
     for j:=1 to n do
      read(inf,d[s,i,j]);
   for i:=1 to n-2 do tempcode[i]:=random(n)+1;
   maxv:=Genetic_uncode(tempcode);randomize;
   sn:=1;ans[1]:=maxv;ansb[1]:=tempcode;
    close(inf);
  end;
function find(v1,v2:vector):longint;
 var i:longint;
```

```
      check:boolean;
  begin
   check:=true;
   for i:=1 to k do
    if v1[i]<v2[i] then begin check:=false;break;end;
   if check then begin find:=1;exit;end;
   check:=true;
   for i:=1 to k do
    if v1[i]>v2[i] then begin check:=false;break;end;
   if check then begin find:=-1;exit;end;
   find:=0;
  end;
procedure insert(v:vector;vcode:code);
 var i,sn2:longint;
 begin
  sn2:=0;
  if (v[1]=2) then
   begin end;
  for i:=1 to sn do
    case find(v,ans[i]) of
     0:begin inc(sn2);ans2[sn2]:=ans[i];ansb2[sn2]:=ansb[i];end;
     1:exit;
```

```
    end;
  inc(sn2);ans2[sn2]:=v;ansb2[sn2]:=vcode;sn:=sn2;ans:=ans2;ansb:=ansb2;
 end;
procedure Genetic;
  var S,S0:Group;
      i:longint;all:int64;
  procedure Genetic_Initial;
    var i,j:longint;
    begin
      for i:=1 to Population do
       for j:=1 to n-2 do
        s[i].data[j]:=random(n)+1;
    end;
   function Genetic_Compute(v:vector;vcode:code):longint;
   var i,mark:longint;nowv:vector;
   begin
    nowv:=maxv;mark:=0;insert(v,vcode);
    for i:=1 to k do
     begin
      mark:=mark+sqr(round((nowv[i]/v[i])*100));
      if v[i]<maxv[i] then maxv[i]:=v[i];
     end;
```

```
  Genetic_Compute:=mark;
 end;
procedure Genetic_Start;
 var i:longint;
 begin
  all:=0;
  for i:=1 to Population do
   begin
    s[i].v:=Genetic_Uncode(s[i].data);
    s[i].power:=Genetic_Compute(s[i].v,s[i].data);
    all:=all+s[i].power;
   end;
 end;
function Genetic_Random:code;
  var now:longint;seed:int64;
   begin
    Seed:=random(all)+1;now:=1;
    while Seed>s[now].power do begin Seed:=Seed-s[now].power;inc(now);end;
    Genetic_Random:=s[now].data;
   end;
function Genetic_Suddenly:code;
 var temp:code;
```

```
        Seed:longint;
     begin
      temp:=Genetic_Random;Seed:=random(N-2)+1;
      Temp[Seed]:=random(n)+1;
      Genetic_Suddenly:=temp;
     end;
   function Genetic_Change:code;
    var temp1,temp2:code;
        Seed1,Seed2,temp:longint;
     begin
      temp1:=Genetic_Random;Seed1:=random(N-2)+1;
      temp2:=Genetic_Random;Seed2:=random(N-2)+1;
      Temp:=temp1[Seed1];Temp1[Seed1]:=Temp2[Seed2];Temp2[Seed2]:=Temp;
                                                                    if
Genetic_Compute(Genetic_Uncode(temp1),temp1)>Genetic_Compute(Genetic_Uncode(tem
p2),temp2) then Genetic_Change:=temp1
                                                                  else
Genetic_Change:=temp2;
     end;
   procedure Genetic_Genarate;
    var Seed,i:longint;
    begin
     Genetic_Start;
```

```
     for i:=1 to Population do
      begin
       Seed:=random(100)+1;
       if Seed <= Suddenly then s0[i].data:=Genetic_Suddenly
                                          else if seed<= Change+Suddenly then
s0[i].data:=Genetic_Change
                                                                          else
s0[i].data:=Genetic_Random;
      end;
     s:=s0;
    end;
   begin
    Genetic_Initial;
    for i:=1 to Live do
     begin
      Genetic_Genarate;
    end;
   end;
  procedure sortans(l,r:longint);
   var tl,tr:longint;tmp:vector;
   begin
    tl:=l;tr:=r;tmp:=ans[l];
    while tl<tr do
```

```
    begin
     while (ans[tr][1]>tmp[1])and(tr>tl) do dec(tr);
     if tr=tl then break;ans[tl]:=ans[tr];inc(tl);
     while (ans[tl][1]<tmp[1])and(tl<tr) do inc(tl);
     if tr=tl then break;ans[tr]:=ans[tl];dec(tr);
    end;
   ans[tl]:=tmp;
   if tl>l+1 then sortans(l,tl-1);
   if tr+1<r then sortans(tr+1,r);
  end;


procedure Output_Result;
 var i,j:longint;
 begin
  sortans(1,sn);
  for i:=1 to sn do
   begin
    write(ouf,'The ',i,'th Answer:');
    for j:=1 to k do
     write(ouf,ans[i][j]:9:2);
     write(ouf,'  Code=');for j:=1 to n-2 do write(ouf,' ',ansb[i][j]);
    writeln(ouf);
```

```
    end;
  writeln(ouf);
   for j:=1 to k do
    begin
     for i:=1 to sn do
      writeln(ouf,ans[i][j]:9:2);
      writeln(ouf,'------');
     end;
   writeln(ouf,'Total=',sn);
   close(ouf);
  end;


procedure start;
begin
   assign(inf,' input.txt' );reset(inf);
   assign(ouf,' edit8.text' );rewrite(ouf);
   Input_Initial;
   Genetic;
   Output_Result;
 end;
begin
 start;
```

end.