非最优化算法初探

北京四中　杨培

【关键字】贪心、随机化、最优化、局部搜索

【摘要】本文介绍了非最优化算法的基本理论，总结了贪心算法的适用条件和部分使用技巧。并在此基础上介绍了禁忌搜索算法及其应用实例。指出了随机化方法的若干适用范围。总结了非最优化算法的优越性。

概论

现代信息学问题分为两类：一类是存在有效算法的所谓 P 类问题，另一类是目前尚未找到有效算法的所谓 NPC 问题。为了解决后一类问题，我们引进了非最优化算法的概念。本节将对 NPC 的概念和提出非最优化算法的必要性等问题进行阐述。

一．引言

信息学是 20 世纪后半叶诞生的一门崭新的学科。数学、物理学、化学等学科的诞生是建立在一些科学家纯粹兴趣的研究基础上，其初期，科学家们的工作是纯理论性的。而信息学与他们的最大不同之处在于：自诞生之日起，它就与实际应用紧密地结合在一起了。每一个算法的提出，都有它广泛的应用背景。国际信息学奥林匹克竞赛（IOI）自 1989 年创办以来，已经举行了 11 届，其间题目的风格经过了一个曲折的发展过程。但我们应该看到，总的趋势是：向实际靠拢，每个题目都有一个实际背景，考察选手从实际问题中抽象出数学模型的能力。

但是，在众多的实际问题中，真正存在有效算法的 P 问题是少数，而大多数也是困扰人们的是 NPC 问题。在没有有效算法的情况下，要解决 NPC 问题，只能用一些非最优化算法在可接受的时间复杂度内求得一些近似解。因此为了考察选手在解决这方面题目的能力，在近年来的竞赛中，分段计分等非正常计分的题目逐渐增多。另外，在竞赛过程中，对于一些暂时想不出有效算法或实现有效算法比较困难的题目，使用非最优化算法可以得到不错的效果。（见表一）由上面两个原因可以看出，今后非最优化算法还是大有用武之地的，对非最优化算法的研究、总结是必要的。

年份	比赛	题目	较好的算法	题目类型
1997	IOI	火星探测车	贪心	最大流问题
		地图标签	贪心/随机化	NPC 问题
		集装箱	概率+贪心	随机规划
		千足虫	构造近似算法	
		HEX 游戏	构造近似算法	博弈
1998	NOI	并行计算	贪心+随机化	大规模搜索
1999	冬令营	迷宫改造	贪心+随机化	动态规划
	NOI	01 串	随机化	最长路径
	国家队作业	保卫地球——邵铮	随机化	NPC 问题
	IOI	地下城市	贪心	
		均分纸牌	贪心/随机化	

表格 1 应用非最优算法效果较好的题目

二．基本概念

可行性问题和最优性问题的关系

应用非最优化算法的题目可简单的分为两类：可行性问题和最优性问题。虽然在选择具体算法时，要对这两种问题加以区分、分别对待，但这两种问题在本质上是统一的，都可以划归为如下的判定问题。

定义 1.2.1：如果一个问题的每一个实例只有“是”或“否”两种答案，则称这个问题为判定问题。

可行性问题可直接转化为“是否存在解”的判定问题。而对于最优性问题，可转化为若干个“是否存在比当前解更优的解”的判定问题。从这种意义上讲，可行性问题$\subset$最优性问题，我们只需重点研究后者。

临域概念

定义 1.2.2：对于一个最优性问题，它的所有可行解的集合 D 上的一个映射：

$$N：S\in D \rightarrow N(S)\in 2\text{^}D$$

称为一个临域映射，其中 2^D 表示 D 的所有子集组成的集合，N(S)称为 S 的临

域，S’∈N(S)称为 S 的一个邻居。

事实上，传统的简单算法如爬山法、贪心法都是建立在对临域的搜索基础上的。显然，上面的方法只能得到问题局部最优解，不能保证得到全局最优解。而非最优算法的目的之一就是用较小的代价跳出局部最优点，从而尽可能接近全局最优点。

三．非最优化算法分类

非最优化算法可简单的分为两类：一步算法和改进算法。

一步算法

该算法的特点是：不在两个可行解之间选择，在未终止的迭代中，又可能不是一个可行解，算法结束时得到一个可行解。这种算法的时间复杂度是容易接受的。

该算法的典型例子是火星探测器问题的贪心算法，每一辆车选一条可装矿石最多的路线，直到分配完所有车的路线。该算法没有在两个可行解之间比较选择，算法结束时得到一可行解。应当注意的是：在解决可行性问题的时候，在算法运行中可能发现无法得到最终的一可行解，这就需要进行简单的回溯或者干脆推翻了重来（如果使用了随机化方法）。例子是 NOI’99 的“01 串”问题。

改进算法

改进算法的迭代过程是从一个可行解到另一个可行解，通常通过两个解的比较而选择好的解，进而作为新的起点进行新的迭代，直到满足一定的要求为止。该算法一般应用于最优性问题。例如局部搜索算法或爬山法，都是改进算法的一种。另外，在使用随机化方法时，例如迷宫改造，需要反复随机求得可行解，最后选出其中最优的。这也可以说是一种改进算法。

四．非最优化算法的性能分析

虽然非最优化算法由许多优点，但最大的缺陷是不能保证得到全局最优解，所以对算法的评价就显得十分重要。评价一个非最优化算法需要两条标准：一是看它的解与最优解的接近性，这是根本条件。如果算法得出的解与最优解的差距较大，按目前已有的计分方法，得分会相当低，更不要说应用在求最优解的题目上了。其二是考察算法的稳定性，这也是十分必要的。下面介绍了几种测试算法的简单方法。

最坏情况分析

对于任何算法，都要考察它在最坏情况的时间复杂度和空间复杂度，但对

于非最优化算法，还需要考察它的最坏情况解的效果。通过最坏情况分析来评价算法的效果，其指标是计算解的值同最优解的值之间的差距，差距越小说明算法越好。

但是，最坏情况分析不能全面地评价算法的好坏，某些算法在小规模时效果很差，但不能说这个算法一定很差。（例如，集装箱问题中钱文杰的算法）另外，找到一个算法的最坏情况并不容易，它需要很多数学技巧，这也限制了这种分析方法的应用。

概率分析

概率统计分析的方法是从理论上考虑的，它假设实例的数据服从一定的概率分布。在这个数据概率分布的假设下，研究其算法或解的平均效果。这种分析方法对于估计随机化算法的可行性是十分重要的。例如，对于随机化的素数判定算法，概率分析为算法的稳定性提供了理论基础。但是概率分析是一种理论分析方法，它需要对问题本身有较深入的理解，并且掌握概率模型的建立和概率理论，要求有较强的数学基础。对于一些较复杂的问题，在考场上无法完成分析，因此，它有较大的局限性。

大规模计算分析

大规模计算分析就是常说的用测试数据测试的方法。它可对多个算法进行评价，比较分析不同算法的效果。可根据各个算法的计算结果，采用简单或统计的方法比较不同算法的性能，而不需要预先得到每个实例的最优解。用这种方法分析算法时，需要产生一些具有代表性的测试数据。关于如何使测试数据全面、准确，请参阅杨帆的《准确性、全面性、美观性——测试数据设计中的三要素》一文。

非最优化算法初探

一．贪心算法

1．概念

贪心算法是从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快地求得更好的解。当达到某算法中的某一步不能再继续前进时，算法停止。这时就得到了问题的一个解，但不能保证求得的最后解是最优的。在改进算法中，贪心算法演化为爬山法。

2．特点及使用范围

贪心算法的优点在于时间复杂度极底。贪心算法与其他最优化算法的区别在于：它具有不可后撤性，可以有后效性，一般情况下不满足最优化原理。贪心算法的特点就决定了它的适用范围，他一般不适用于解决可行性问题，仅适用于较容易得到可行解的最优性问题。这里较容易得到可行解的概念是：当前的策略选择後，不会或极少使后面出现无解的情况。另外，对于近年来出现的交互性题

目，贪心算法是一个较好的选择。这是因为，在题目中，一个策略的结果是随题目的进行而逐渐给出的，我们无法预先知道所选策略的结果，这与贪心算法不考虑策略的结果和其具有后效性的特点是不谋而合的。当然，贪心算法还可以为搜索算法提供较优的初始界值。

3．使用贪心算法的技巧

贪心与最优化算法的结合

尽管贪心算法有一定的优越性，但它毕竟在一般情况下得不到最优解。因此为了尽量减小贪心算法带来的副作用，使得最后得到的解更接近最优解，应该在算法尽可能多的地方使用有效的最优化算法（如动态规划）。

例如，在火星探测器这道题中，当我们找不到有效的算法使探测器之间互相帮助，综合分配任务时，就只能使用贪心算法。让每辆探测器独自运行找一条可使自己得到矿石最多的路线。在这一过程中，我们可以使用动态规划算法来保证每辆探测器走的是当前最优的路线。虽然每个局部的最优解加在一起不等于全局最优解，但也很大程度上弥补了全局上使用贪心算法的弊端。（详见文献[1]Vol.1No.1）

而在邵铮的《保卫地球》一题中，对于用有限个导弹将舰队的战斗力降至最低的子问题，使用了贪心的算法。但在分配打击每个舰队的导弹数时，使用了动态规划算法。这样在局部使用贪心，在总体上用“最优”的算法，得到的效果也适当好的。（详见文献[7]邵铮的解题报告）

贪心算法中权值的选择

贪心算法的核心是在所选择的策略中，选一个权值最优的策略作为当前策略。因此贪心算法的好坏主要决定于权值的确定。

权值的确定算法应遵循两个原则：

1．尽可能的包含一些选择这一策略後对整个问题的影响的信息。显然，如果包含的信息越多，权值越接近于这个策略的真正价值，这使得到的最后结果越接近最优解。如果权值包含了选择这一策略後对整个问题的全部影响，则算法退化为搜索算法。

2．计算权值的计算复杂度不能太高，否则将丧失贪心算法的唯一优点。

可以看出，以上两个标准是矛盾的，我们的目标就是在这对矛盾中找到平衡点。

4．贪心算法的改进

贪心算法的缺点在于解的效果比较差，而最大优势在于极低的时间复杂度

而且往往时间复杂度远远低于题目的限制。那么，我们为什么不再花一部分时间来提高目标解的效果呢？这就是对贪心算法改进必要性，在这里我们讨论一种贪心算法与搜索策略相结合的算法。

局部搜索算法

首先回顾一下爬山算法，即纯粹的局部搜索策略。

算法 2.4.1 局部搜索算法：

STEP1：选定一个初始可行解 x^0；记录当前最优解 $x^{best} := x^0$，令 $P = N(x^{best})$；

STEP2：当 $P \neq \varnothing$ 时，或满足其他停止运算准则时，输出计算结果，停止运算；否则，从 $N(x^{best})$ 中选一集合 S，得到 S 中的最优解 x^{now}；若 $f(x^{now}) < f(x^{best})$，则 $x^{best} := x^{now}$，$P := N(x^{best})$；否则，$P := P - S$；重复 STEP2。

在局部搜索算法中，STEP1 的初始可行解可采用贪心算法或其他算法求得。这种算法的效果取决于 S 的大小。

禁忌搜索的一个例子

下面我们引入一种禁忌搜索的算法来对局部搜索算法进行改进，它的一个重要思想是标记已得到的局部最优解，并在进一步的迭代中避开这些局部最优解。首先看一个例子。

例 2.4：NOI99 “01 串”问题。先定义一个目标函数 $f(x)$：

$g_0(x,i)$ 为解 x 中从第 i 个字符开始连续 L0 个字符中 “0”的个数。

$g_1(x,i)$ 为解 x 中从第 i 个字符开始连续 L1 个字符中 “1”的个数。

$$f(x) = \sum_{i=1}^{N-L_0+1} \begin{cases} g_0(x,i) - B_0 \,|\, g_0(x,i) > B_0 \\ 0 \,|\, A_0 \le g_0(x,i) \le B_0 \\ A_0 - g_0(x,i) \,|\, g_0(x,i) < A_0 \end{cases} + \sum_{i=1}^{N-L_1+1} \begin{cases} g_1(x,i) - B_1 \,|\, g_1(x,i) > B_1 \\ 0 \,|\, A_1 \le g_1(x,i) \le B_1 \\ A_1 - g_1(x,i) \,|\, g_1(x,i) < A_1 \end{cases}$$

显然，我们的任务是求一个解 x’，使得 $f(x')=0$ 。在求解的过程中，目标是寻找*f(x)*尽可能小的解“x”。

用一个实例求解的过程来举例：N=10,A0=1,B0=2,L0=3,A1=1,B1=1,L1=3。假设：初始解 $x^0=(1111111111)$，临域映射为对某一个位置进行取反操作，目标值 $f(x^0)=24$ 。

第 1 步：候选集

操作	1	2	3	4	5	6	7	8	9	10
评价值	22	20	18★	18	18	18	18	18	20	22
禁忌长度	0	0	0	0	0	0	0	0	0	0

此处评价值为目标值。从候选集中选一个最好的操作——对位置 3 取反，用★标记入选的操作。目标值由 24 下降为 18，有所改善。

第 2 步：

$x^1=(1101111111)$， $f(x^1)=18$

操作	1	2	3	4	5	6	7	8	9	10
评价值	17	16	24T	14	13	12★	12	12	14	16
禁忌长度	0	0	3	0	0	0	0	0	0	0

由于第 1 步中选择了对位置 3 取反，于是，我们希望这样的交换在下面的若干次迭代中不再出现，以避免计算中的循环，操作 3 称为禁忌对象并限定在 3 次迭代中不允许再次对位置 3 取反。对应位置记录 3。在 $N(x^1)$ 中又出现了被禁忌的操作 3，故用 T 标记而不选此交换。

第 3 步：

$x^2=(1101101111)$， $f(x^2)=12$

操作	1	2	3	4	5	6	7	8	9	10
评价值	11	10	18T	9	9	18T	8	7★	8	10
禁忌长度	0	0	2	0	0	3	0	0	0	0

新选的操作 6 被禁后，操作 3 在被禁一次后还有两次禁忌。

第 4 步：

$x^3=(1101101011)$， $f(x^3)=7$

操作	1	2	3	4	5	6	7	8	9	10
评价值	6	5	13T	4★	4	12T	7	12T	5	6
禁忌长度	0	0	1	0	0	2	0	3	0	0

第 5 步：

$x^4=(1100101011)$， $f(x^4)=4$

操作	1	2	3	4	5	6	7	8	9	10
评价值	3	5	8	7T	7	8T	4	9T	2★	3
禁忌长度	0	0	0	3	0	1	0	2	0	0

操作 3 解禁。

第 6 步：

$x^5=(1100101001)$，$f(x^5)=2$

操作	1	2	3	4	5	6	7	8	9	10
评价值	1★	3	6	5T	5	6	5	5T	4T	4
禁忌长度	0	0	0	2	0	0	0	1	3	0

第 7 步：

$x^6=(0100101001)$，$f(x^6)=1$

操作	1	2	3	4	5	6	7	8	9	10
评价值	2T	5	4	4T	4	5	4	4	3T	3★
禁忌长度	3	0	0	1	0	0	0	0	2	0

此处所有操作的评价值都劣于原值，在原有的局部搜索算法中，此时已达到局部最优解而停止。但现在，我们允许从候选集中选一个最好的操作——对位置 10 取反。这样就能跳出当前的局部最优解。

第 8 步：

$x^7=(0100101000)$，$f(x^7)=3$

操作	1	2	3	4	5	6	7	8	9	10
评价值	4T	7	6	6	6	7	6	3★	2T	1T
禁忌长度	2	0	0	0	0	0	0	0	1	3

由于评价值更好的操作 9 和操作 10 被禁，只能选操作 8。这样才能避免跳回原来的局部最优区域。

第 9 步：

$x^8=(0100101100)$，$f(x^8)=3$

操作	1	2	3	4	5	6	7	8	9	10
评价值	4T	7	6	6	6	8	0★	3T	6	4T
禁忌长度	1	0	0	0	0	0	0	3	0	2

这样就得到了本题的一个解（0100100100）。

由此，可以总结出禁忌搜索算法的步骤：

算法 2.4.2 禁忌搜索算法：

STEP1：选定一个初始解 x^{now} 及给以禁忌表 $H=\varnothing$；

STEP2：若满足停止规则，停止计算；否则，在 x^{now} 的邻域 $N(H,x^{now})$ 中选出满足禁忌要求的候选集 $Can\_N(x^{now})$；在 $Can\_N(x^{now})$ 中选出一个评价值最佳的解 x^{next}，$x^{now}=x^{next}$；更新历史记录 H，重复 STEP2。

一些技术问题

禁忌搜索中的技术包括：禁忌对象、候选集合的构成、禁忌长度的确定、评价函数的构造、特赦规则、终止规则等。他们对算法的效率有较大的影响。这些都需要在具体问题中具体分析，来设计出效率较高的算法。

禁忌对象：是指禁忌表中被禁的变化元素，它可以记录解的简单变化（例如从(1101)→(1011)），也可以记录解的某个分量的变化（例 2.4 中 01 串问题），还可以记录目标值 f(x)的变化。

禁忌长度：禁忌长度是被禁对象不允许选取的迭代次数。若长度越长，越容易跳出局部最优区域，但有可能引起目标值收敛速度过慢，影响算法效率。在“01 串”这道题上，我使用了非定长的禁忌长度，当前最优目标值越小，禁忌长度相应增加，以利于跳出局部最优解。而在算法运行之初，禁忌长度较小，以使目标值尽快收敛，使算法在此阶段更接近于贪心算法。

评价函数：是指候选集合元素选取的一个评价公式，候选集合的元素通过评价函数值来选取。在“01 串”中，使用目标值 f(x)作为评价函数，这是比较容易理解的。但某些问题中，目标值的计算复杂度太大，若使用目标值作为评价函数，则算法速度较慢，这时需要设计一个替代的评价函数。替代的评价函数不但要求计算复杂度底，而且它应能反映一些目标函数的特性。

特赦规则：在迭代过程中，会出现候选集合中的全部对象都被禁忌，或有一对象被禁忌，但若解禁则其目标值将有非常大的下降情况。在这样的情况下，为了达到全局最优，我们会让一些禁忌对象重新可选。这种方法称为特赦。在“01 串”中，如果某被禁操作的评价值低于当前出现过的最优的目标值——$f_{\min}$，则可以对其解禁。这种特赦规则在 f(x)较小的情况下可以迅速使当前解向最终解靠拢。

禁忌搜索算法适用的范围

禁忌搜索算法的程序较简单，在竞赛中可以用问题的时间限制作为终止规则，从而最大限度的利用所给的时间，弥补贪心算法的不足。它适于解一些标准的组合最优化问题（如地图标签问题、泄洪区规划问题——文献[7]周天凌），对于类似“01 串”问题的可行性问题，则要求题目能较容易的转变为最优化问题，而且要求题目的可行解较多。

二．随机化方法

随机化不是一种完整的算法，在解决非最优化问题过程中，它需要与贪心、局部搜索等算法结合使用，它的作用体现在两个方面：对于可行性问题，提高算法的效率稳定性；对于最优化问题，提高算法的计算结果与最优解的接近度。

随机化在贪心中的应用

由于贪心算法的时间复杂度是极低的，随机化的应用主要指算法中在策略选择中引入随机因子，应通过大量的重复执行算法来获得一个最优的解。这在《并行计算》和《迷宫改造》的随机化算法中有很好的体现并产生了很好的效果。这里就不过多论述了。

随机化在局部搜索中的应用

在局部搜索算法中的随机化应用有两种：在初始可行解的选择上和搜索过程中对相同权值策略的选择上。

如果用随机的办法产生初始可行解能有效地避开人为设置好的局部最优陷阱。它可以提高局部搜索算法的效率（平均）。

如果在局部搜索过程中，出现某些评价值一样的情况，可以用随机的方法来选择其中的一个，同样可提高算法的效率。

上述两种技巧在“01 串”问题的效果如下表：

搜索的迭代次数	Sequence.003	Sequence.004	Sequence.005
普通禁忌搜索算法	1092	1357	758
随机选择评价值相同的操作	854(3.85%)*	726(0.00%)	954(0.00%)
效率提高程度	21.8%	46.5%	-25.9%
初始解由随机确定	27	13	19
效率提高程度	97.5%	99.0%	97.5%

*括号内表示 3.85%的实例中算法掉入局部最优陷阱而出不来。

总结

非最优化算法不但可应用于求解非最优问题，也可以应用于其他最优化问题，它具有如下优点：

1. 适用范围较广。非最优化算法是一种普适性的算法，对目标函数等题目性质没有特殊要求。
2. 用非最优化算法在求解最优化问题中，不需要有很强的技巧和对问题有非常深入的了解。尤其是一些搜索优化的题目，需要针对性很强的技巧。而非最优化算法中，计算过程和程序复杂度都是比较简单的，且可以较快得到一个满意解。

求非最优解的题目和非最优化算法在竞赛中的大规模的应用还是近几年的

事情，许多理论还不成熟，缺乏必要的实践经验，因此本文的总结是十分肤浅的。本文介绍的几种算法只是众多非最优化算法中的几种，其他的还有模拟退火法、遗传算法、人工神经网络等新兴的现代优化计算方法。这些算法在竞赛中的应用还需要大家来探索、实践、总结。

【参考文献】

1．《信息学奥林匹克》（98~99）。中国计算机学会。

2．《IOI’98 中国集训队优秀论文集》。

3．《IOI’99 中国集训队论文集》。

4．刘福生、王建德《青少年国际信息学奥林匹克竞赛指导——人工智能搜索与程序设计》。电子工业出版社。1993 年。

5．邢文训、谢金星《现代优化计算方法》。清华大学出版社。1999 年。

6．王树和《数学模型基础》。中国科技大学出版社。

7．《IOI’99 中国集训队第三次作业》。

【附录】

NOI’99——01 串

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 16384,0,655360}
program noi99_1;
const
  ifilename:string='sequence.005';
type
  polytype=array[1..1000] of 0..1;
  pathtype=array[0..1,1..1000] of integer;
var
  timer:longint;
  len:integer;                                  {禁忌表长度}
  n,n0,n1:integer;
  a0,b0,l0,a1,b1,l1:integer;
  poly:polytype;                                {记录当前的字符序列}
  path:pathtype;                                {记录 poly 中连续字符的 0 和 1 的个数}
  now:longint;                                  {当前字符序列的目标值}
  total:integer;                                {记录迭代的次数}
procedure init;
var
```

```
  f:text;
  i,j:integer;
begin
  assign(f,ifilename);reset(f);
  readln(f,n,a0,b0,l0,a1,b1,l1);
  for i:=1 to n do                              {随机产生初始解}
    poly[i]:=random(2);
  fillchar(path,sizeof(path),0);
  n1:=n-l1+1;n0:=n-l0+1;
{计算初始解的目标值}
  now:=0;
  for i:=1 to n1 do
    begin
      for j:=i to i+l1-1 do
        if poly[j]=1 then inc(path[1,i]);
      if path[1,i]<a1 then inc(now,a1-path[1,i]);
      if path[1,i]>b1 then inc(now,path[1,i]-b1);
    end;
  for i:=1 to n0 do
    begin
      for j:=i to i+l0-1 do
        if poly[j]=0 then inc(path[0,i]);
      if path[0,i]<a0 then inc(now,a0-path[0,i]);
      if path[0,i]>b0 then inc(now,path[0,i]-b0);
    end;
  close(f);
end;
procedure changevalue(p:integer);                {对位置p进行操作后修改path}
var
  i,t1,t2:integer;
begin
  if p>l1 then t1:=p-l1+1
          else t1:=1;
  if p<n1 then t2:=p
          else t2:=n1;
  if poly[p]=0 then begin
    for i:=t1 to t2 do
      dec(path[1,i]);
  end
  else begin
    for i:=t1 to t2 do
      inc(path[1,i]);
  end;
  if p>l0 then t1:=p-l0+1
```

```
          else t1:=1;
  if p<n0 then t2:=p
          else t2:=n0;
  if poly[p]=1 then begin
    for i:=t1 to t2 do
      dec(path[0,i]);
  end
  else begin
    for i:=t1 to t2 do
      inc(path[0,i]);
  end;
end;
function calcvalue(p:integer):longint;          {计算对位置p取反后的评价值}
var
  t:longint;
  i,t1,t2:integer;
begin
  t:=now;
  if p>l1 then t1:=p-l1+1
          else t1:=1;
  if p<n1 then t2:=p
          else t2:=n1;
  if poly[p]=1 then
    for i:=t1 to t2 do
      begin
        if path[1,i]>b1 then dec(t);
        if path[1,i]<=a1 then inc(t);
      end
  else
    for i:=t1 to t2 do
      begin
        if path[1,i]>=b1 then inc(t);
        if path[1,i]<a1 then dec(t);
      end;
  if p>l0 then t1:=p-l0+1
          else t1:=1;
  if p<n0 then t2:=p
          else t2:=n0;
  if poly[p]=0 then
    for i:=t1 to t2 do
      begin
        if path[0,i]>b0 then dec(t);
        if path[0,i]<=a0 then inc(t);
      end
```

```
  else
    for i:=t1 to t2 do
      begin
        if path[0,i]>=b0 then inc(t);
        if path[0,i]<a0 then dec(t);
      end;
  calcvalue:=t;
end;
procedure compute;          {禁忌搜索过程}
var
  i,mi,link:integer;
  sign:array[0..1000] of boolean;        {记录各个位置是否在禁忌表中}
  list:array[1..200] of integer;         {禁忌表}
  min,t,mmin:longint;    {min:候选集中的最优评价值；mmin:当前曾达到的最优目标值}
  ll:array[0..5] of integer;             {mmin在不同阶段的禁忌长度}
begin
  fillchar(sign,sizeof(sign),false);
  fillchar(list,sizeof(list),0);
  link:=0;mmin:=maxlongint;
  len:=trunc(n/10);
  ll[0]:=trunc(n/5);ll[1]:=trunc(n/6);ll[2]:=trunc(n/7);ll[3]:=trunc(n/8);
  ll[4]:=trunc(n/9);
  total:=0;
  while now>0 do
    begin
      inc(total);
      if total>2000 then exit;
      if now<mmin then mmin:=now;
      if mmin<1000 then len:=ll[mmin div 200];
      min:=maxlongint;
{从候选集中选出一个最优操作}
      for i:=1 to n do
        if not sign[i] then
          begin
            t:=calcvalue(i);
            if (t<min) {or ((t=min) and (random(2)=0))} then
            begin
              min:=t;mi:=i;
            end;
          end
          else begin
            t:=calcvalue(i);
            if (t<mmin) and (t<min) then begin          {特赦规则}
              min:=t;mi:=i;
```

```
            end;
          end;
      now:=min;
      poly[mi]:=1-poly[mi];
      changevalue(mi);
      inc(link);
      if sign[mi] then continue;              {特赦情况}
{修改禁忌表}
      sign[list[link]]:=false;
      list[link]:=mi;sign[mi]:=true;
      if link=len then link:=0;
    end;
  for i:=1 to n do
    write(poly[i]);
  writeln;
end;
begin
  randomize;
  timer:=meml[$40:$6c];
  init;
  compute;
  writeln('Time=',(meml[$40:$6c]-timer)/18.2:0:2);
end.
```