

人工智能原理与方法——十五数码问题的 程序实现与评估函数最优化方法讨论

Principles and Methods of Artificial Intelligence: Discussion on Program Implementation and Evaluation Function Optimization Method of Fifteen Digital Problems

北京航空航天大学(BUAA) 自动化科学与电气工程学院(SASEE)

邵金鑫(Jinxin SHAO)

neushaojinxin@163.com

Abstract: This work realizes the solution of 15 digital problems based on the minimization of comprehensive evaluation function, and discusses how to optimize on the basis of the original search evaluation function, so that the calculation amount and the optimal solution relationship are balanced. The author first tried the greedy algorithm to determine whether the problem has a solution. The problem must be a solution problem, but the number of steps obtained is more than 100 steps, which must be non-optimal. Then the traditional A* algorithm and the depth-first non-optimal solution A algorithm are tried to get the result of balancing the calculation amount and the optimal solution as much as possible. A 43-step solution was obtained when solving the problem using the A algorithm, which the author believes may be optimal. Since this work is performed using matlab, the computational efficiency is low. The author discusses the use of dynamic weighting algorithm and uses the evaluation function to discuss. In the initial state, as much as possible, the side restarts the evaluation effect of the function, so that the previous stage is in the number of steps. Convergence as soon as possible under as few conditions as possible, with as little error as possible. The latter stage is as small as possible, the cost is reduced as much as possible, and the whole problem is solved as optimally as possible, that is, an early depth-first and late-wide breadth-first

algorithm. The author has always hoped that a better heuristic function can more scientifically describe the state of the problem, so that the search can reduce the amount of calculation as much as possible, and the problem is more intelligent.

摘要：本次工作实现了基于综合评价函数最小化的 15 数码问题求解，并讨论了如何在原有的搜索评价函数的基础上 $f(p) = g(p) + h(p)$ 进行优化，使计算量和最优解关系得到平衡。其中作者首先尝试了贪心算法 $f(p) = h(p)$ ，判断该问题是否有解，该问题一定是一个有解问题，但是得到的步数为 100 多步，一定不是最优的。而后尝试了传统的 A* 算法 $f(p) = g(p) + h(p)$ 以及侧重深度优先的非最优解 A 算法 $f(p) = g(p) + 2h(p)$ 得到尽可能平衡计算量与最优解的结果。使用 A 算法解决问题时得到了 43 步的解决方案，作者认为可能是最优的。由于本次工作使用 matlab 进行，计算效率较低，作者讨论了使用了动态加权算法，利用评价函数 $f(p) = g(p) + w(p) * h(p)$ $w(p) \geq 1$ 来进行讨论，在初始状态下， $w(p)$ 尽可能大，侧重启发函数的评价效果，使前期在步数尽可能少的条件下尽快收敛，误差尽可能小。后期 $w(p)$ 尽可能小，尽可能减少代价，使整个问题求解尽可能达到最优，即一种前期深度优先后期广度优先的算法。作者一直希望有一种更好的启发函数更能科学的描述问题的状态，使得搜索能尽可能的减少计算量，问题更加智能。

1.引言

十五数码问题是人工智能领域的经典问题，起源于八数码问题的推广，八数码问题的规模较小，总的状态数为 $9! (=362880)$ 个，而十五数码问题的状态数为 $16! (\approx 20.9 \times 10^{12})$ 个。因此，十五数码问题作为逻辑智能领域的经典问题，一般用来检验设计的搜索算法的可执行性和效率。根据 8 数码问题的搜索树设计，将 p 设计为迭代步长（或者问题的状态数），则已经执行的步长设为 $g(p)$ ，根据作者的设定一般认为 $g(p) = p$ 或 $g(p) = p - 1$ ，即已经迭代的步数；一般来说，

八数码问题 A* 算法设计的期望函数 $h(p) = \sum (|X_{jk}^p - X_{jk}^0| + |Y_{jk}^p - Y_{jk}^0|)$ ，即认为在

某一状态下，未来移动步数的估计是每一个元素 X, Y 坐标差绝对值的和。可以证明，这种方法的得到的估计是小于等于实际要移动的最低步数的，所以在八数码问题中，这种估计方法被认为是一种可以找到最优解的完备方法，并取得了很好的效果。然而在解决十五数码的问题中，由于问题的复杂度有了 10 的 7 次方量级的提升，这种方法略有侧重广度优先的搜索算法可能起不到很好的效果，需要采用一些有界深度优先算法或者动态加权算法来在问题初期降低复杂度，在可能性更大的一些树结构上在采取这种略侧重广度的 A* 算法来找到最优解问题，本文则重点讨论了这一问题，并探讨了最优解问题与运算效率，运算可行性之间的相互关联。

2. 程序设计总体思路

程序设计整体环境在 matlab 环境下进行，分为主函数部分 (open 表初始化，open 表不断循环，误差判断是否为 0，在得分最低的节点进行继续扩展)，误差计算部分 (计算每次扩展后的步长，期望，总体得分)，扩展判断执行部分 (分析这个部分是否可以扩展，扩展是否和上一动作相逆，扩展后的结果是否与之前出现的结果相同，即出现循环，出现循环后，对整个节点进行剪枝)，具体的上下左右移动部分 (执行上下左右移位操作)。通过上述部分实现的整体程序。

由于本次设计使用 matlab 操作，为了更好地可视化结果，对于每次扩展后的节点下一步的操作作者直接在数据的下一位进行加写数据。以 A 点 (题中的 '1' 点) 为例，A 如果为 (1,2; 1,2; 1,3)，则代表在 A0 状态下在 (1,2) 点，A1 状态下在 (1,2) 点，A2 状态下在 (1,3) 点。这样假设这个节点最终经过 n 步扩展除了最后的结果，则每一个节点 (题中的 '1' 到 '15'，在程序中命名为了 A 到 O，空点命名为了 X) 都可以画出 n 个状态的直观演化图 (后文中会提到)，这样就可以轻松的实现可视化。所以没有必要表示出 open 表中剩余的子节点与 close 表里的父子指针关系，也没有必要保留扩展后的 close 表，因此为降低计算量，在父节点扩展后，作者直接将 open 表中的节点删去，并没有存在 close 表中。

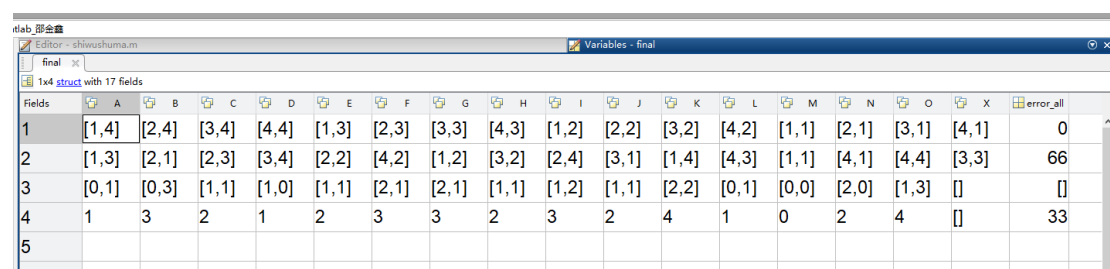
3.分程序块讲解

1.数据结构设计思想（主程序初始化部分）

本次程序设计首先定义了一个 1×4 的 17 个元素的结构体来存放每个节点的数据，即相当于这个 4×17 的数据块相当于每一个节点，其中 A 到 O 相当于题中给的 1 到 15 的十五数码，用字母表示为块名则更为方便。X 代表数码中的空数码，在后文的可视化结果中用 NaN 表示。最后增加了 error_all 一项，用来表示评价函数中的 $f(p), g(p), h(p)$ 三个参数，其中 final(1).error_all 表示扩展步数，即 $f(p)$ ，比如在图 1 中的 0 表示未经过扩展过程，final(4).error_all 表示误差计算值，即标准的 $h(p)$ ，计算方法为标准的 A*算法，即 final(1)代表目标状态的坐标，该行的值是不变的，final(2)是一个不断扩展的矩阵，每一行代表着现在坐标位置，最后一行就代表着当前转态。同理，final(3)代表着 x, y 坐标的误差的绝对值，final(4)代表误差绝对值的和，这两行都是随着第二列的扩展不断扩展的，final(2).error_all 代表着计算后的评价函数值，即：

$$f(p) = g(p) + w(p) * h(p) \quad w(p) \geq 1$$
，作者尝试了完全深度优先算法，A 算法，A*算法，权值变化算法等，在 $w(p)=2$ 时得到了 43 步的解决方案，如图二所示。

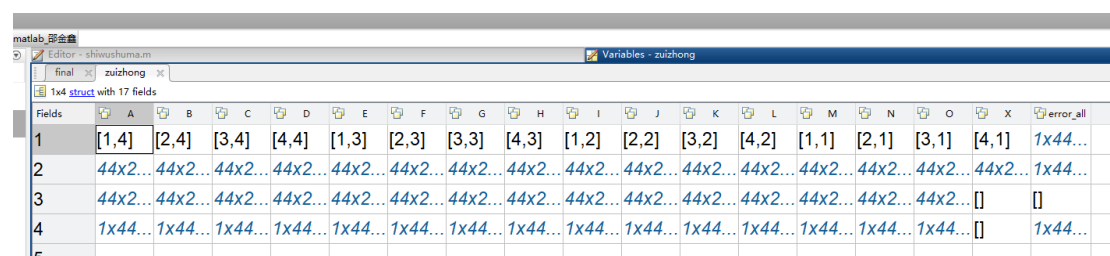
该部分主要对应 matlab 文件的 shiwushuma.m（主函数）和 jisuan.m。



The image shows the MATLAB Variables window for the 'final' variable, which is a 1x4 struct with 17 fields. The fields are A through O, X, and error_all. The data is as follows:

Fields	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	X	error_all
1	[1,4]	[2,4]	[3,4]	[4,4]	[1,3]	[2,3]	[3,3]	[4,3]	[1,2]	[2,2]	[3,2]	[4,2]	[1,1]	[2,1]	[3,1]	[4,1]	0
2	[1,3]	[2,1]	[2,3]	[3,4]	[2,2]	[4,2]	[1,2]	[3,2]	[2,4]	[3,1]	[1,4]	[4,3]	[1,1]	[4,1]	[4,4]	[3,3]	66
3	[0,1]	[0,3]	[1,1]	[1,0]	[1,1]	[2,1]	[2,1]	[1,1]	[1,2]	[1,1]	[2,2]	[0,1]	[0,0]	[2,0]	[1,3]	[]	[]
4	1	3	2	1	2	3	3	2	3	2	4	1	0	2	4	[]	33
5																	

图 1.单个节点的数据结构表示（初始节点）



The image shows the MATLAB Variables window for the 'zuizhong' variable, which is a 1x4 struct with 17 fields. The fields are A through O, X, and error_all. The data is as follows:

Fields	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	X	error_all
1	[1,4]	[2,4]	[3,4]	[4,4]	[1,3]	[2,3]	[3,3]	[4,3]	[1,2]	[2,2]	[3,2]	[4,2]	[1,1]	[2,1]	[3,1]	[4,1]	1x44...
2	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	1x44...
3	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	44x2...	[]	[]
4	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	1x44...	[]	1x44...
5																	

图 2.最终有解的数据结构表示（最终节点）

2.上下左右移动的设计方法

上移下移左移右移四个函数（函数名为汉语拼音）实际上是同一函数的简单改写，这里只选取其中一种方法如 `youyi.m` 进行说明，其中这个函数会调用查重函数 `chongfu.m`。

作者设定的上下左右移动实际上是空点的上下左右移动，首先定义不可行（`bukexing`）和终止（`zhongzhi`）的标志位都为 0，函数的输入为 `final`，输出为 `bukexing`, `zhongzhi`, `final`。

首先判断将 `X`（NaN）坐标右移 1 位，如（2,3）变为（2,4），判断是否在棋盘的范围内，如果在棋盘范围内，说明可以右移，这时要找到右边的坐标现在的那个点，假设 `A` 点的坐标（状态为 `n`）为（2,4），动作的意义实际上就是其他棋子复制而对应的两个棋子换位的过程，这时先不要动作，进行下一步验证。

紧接着判断这个动作是不是上一个动作的逆动作，比如上一个动作是左移，这个动作右移，这样在 `open` 表里就会有这种无意义的循环，会使程序进入无解死循环，判断方法是判断对应节点的上一个状态是不是将要移动的那个状态，就是 `X` 对应坐标向右加一的位置（2,4）是不是 `A` 在 `n-1` 的状态。如果是的话，不要动作，返回原有的节点状态（原始的 `final`，`final` 为函数名），并返回不可行为 1，交给主函数进行下一步处理。

如果没有问题就开始换位操作，换位之后相当于 `final(2)` 的值全部更新，重新调用 `jisuan.m` 函数，更新整个 `final` 节点，这样在 `final(2).error_all` 里就得到了当前的评分，在主函数里会调用这个值进行 `open` 表的扩展删除并优先决定扩展哪个点。这个时候还要进行最后一部分评估，判断是否删除这个节点。这种剪枝方法作者还没进行验证，`shiwushuma.m` 是带剪枝算法的主函数，

`duibi_shiwushuma.m` 是不带剪枝算法的主函数，作者计划在相同参数下对比跑这两个主函数，通过输出效果来判断这种剪枝算法的效果。这种剪枝算法实际上是对比，新扩展的节点是否与之前的节点相同，如果与之前的节点相同，则代表陷入循环移位过程，毫无意义，判断方法见子函数 `chongfu.m`。区别在于

`shiwushuma.m` 仅仅是不使用带循环移位的节点，而且删除了该节点的父节点，而 `duibi_shiwushuma.m` 仅仅是不使用该节点，该节点的父节点还是正常扩展的，即仅仅是通过这种查重删去了重复节点。作者刚开始认为这种剪枝是有道理的，使用的 `shiwushuma.m` 进行训练，后来觉得这种剪枝没有道理，仅做查重删除，

使用 `duibi_shiwushuma.m`。这里的剪枝理论依据有问题，把这种剪枝删去后，计算效率得到了显著地提升，计算时间至少缩减为原来的 10 分之 1，属于我的编程失误，所以请使用 `duibi_shiwushuma.m` 作为主函数而不是 `shiwushuma.m`。

3.主函数的设计思想

最终进入主函数，初始化后 `open` 表里只有一个点，同样可以计算最小估值函数节点的值（估值最小的，如果有多个节点估值相同，则使用 `open` 表里排位靠前的节点进行扩展），扩展对于一个节点顺序进行上下左右扩展，4 种情况当返回的 `flag` 值不出现不可扩展和重复情况时才添加到 `open` 表里，同时在将这 4 种情况讨论后，将父节点从 `open` 表里删除，扩展得到新的 `open` 表之后首先判断是否得到误差为 0 的解情况，如果没有得到循环执行找最小估价函数的操作，找到新的待扩展节点后循环执行上述操作，直到得到最终误差为 0 的解，这个时候跳出循环，将输出节点封装为 `zuizhong` 输出即可。

4.显示函数的设计思想

最终节点输出形式是一个结构体形式，从初态到终态的所有值都有显示，并且包含了每次计算的误差，代数，估值等信息，但是这种形式是不好可视化的（如图 2 所示），因此通过 `kanjiegua.m` 函数进行模拟实际棋盘的可视化输出。

具体实现方法十分简单，原有的 `zuizhong` 输出包含了每个图的每个点的位置信息，首先判断出整个图的个数，比如这个结果得到了 44 个状态图，43 步，构造一个 $4 \times 4 \times 44$ 的矩阵，使用循环进行赋值，对应 A 到 O 的坐标赋给 1 到 15，X 的坐标赋 NaN，然后将矩阵输出即可，效果如附录 1 所示。

4.关于评价函数设置对结果影响的讨论

作者首先要使用无限深度算法判断该问题是否有解，深度优先算法对于找最优解问题是不完备的，但是可以证明的是对于找出是否有解问题是完备的，则作者最初使用完全深度优先方法，即贪心算法 $f(p) = h(p)$ ，进行搜索，发现最终得到了步长 133 步的解，并且实际上这种方法计算量时间较长，计算量较大。很明显这种方法不是最优的，计算结果可以查看保存数据 `chunshenduyouxian_jie.mat`，具体结果在 `keshihua` 项里，由此作者采用了传统的 A* 算法，即 $f(p) = g(p) + h(p)$ ，发现计算超过了一个小时都没有得到收敛解，

说明这种方法虽然是理论上完备的,但是由于计算量过于庞大可能很难计算出收敛的解。这时就需要在原有的基础上适当进入深度加深,由 A^* 算法变为 A 算法,最开始作者尝试了 $f(p) = g(p) + w(p) * h(p)$ $w(p) = 2$ 的情况,计算时间较长才能收敛,大概 10 分钟左右,得到了 43 步的收敛解,数据为 `axing_1bi2jiusanjieguo.mat`,计算结果也在 `keshihua` 项里。则作者进行了逐层深度减小的优化,作者使用了简单的代码:

```
w=2-0.015*size_chang;
```

```
final(1,2).error_all(end+1)=final(1,1).error_all(end)+(w*final(1,4).error_all(end));%
```

侧重深度优先算法

首先判断层的深度,根据层数的多少对加权函数进行线性衰减,比如使用 $w(p) = 2 - 0.15p$ 公式, 计算在 1 分钟内就会得到收敛解,并且也得到了 43 步的解,和上一计算结果相同,但时间有了非常大的节约,这次运算可以在 1 分钟内实现,并且,作者尝试了其他参数,如 0.15, 0.20 等,均取得了很好的效果,但在尝试更多的广度优先权重时,即系数为 0.25 时,出现了较大的计算量,计算大概在 20 分钟内得到了收敛值,但结果并没有变好,得到了 53 步的收敛解,结果存储在 `canshu025_jie.mat` 中,说明并不是广度越重结果就是越好的,存在这种非线性现象,这时加上了 $w(p) < 1$ 时限制 $w(p) = 1$ 的限制条件,并不断缩小 $w(p)$ 的初值,发现并没有得到更好的结果,最优解仍出现在 43 步结果上,尽管可以采用一些蒙特卡洛方法,使 $w(p)$ 在一定程度内随机变化,增加训练更多的可能性,增加训练次数可以得到不同的实验结果,作者使用了:

```
“w=rand(1)/2+1;%蒙特卡洛方法”
```

这种简单语句实现了随机选取权值,作者尝试了几次,结果都没有改变(见 `mengtekaluojie.mat`),因此,有理由认为 43 步结果就是本问题的最优结果。

5.结论

本次工作用 `matlab` 程序实现了 15 数码问题的求解工作,实现了经典逻辑智能中的 `open` 表扩展搜索方法,并在传统的 A^* 方法上实现了基于深度线性加权的评价函数搜索方法和蒙特卡洛方法,得到了 43 步的最优解,对解决诸如此类的

大计算量智能搜索有重要意义，此外，作者认识到此类盲目搜索问题的大计算量性，希望未来使用有针对性的搜索方法解决此类问题。

邵金鑫

2018/11/22

附录:

注:

- 1. 代码执行方法请读取 `readme.txt`。
- 2. 本次设计 `copyright` 归邵金鑫所有，仅供北航人工智能原理与方法王岩老师使用，请勿外传，[有问题联系 neushaojinxin@163.com](mailto:neushaojinxin@163.com)。
- 3. 具体代码查看文件夹十五数码_matlab。

1.43 步解决该方法:

```
keshihua(:,:,1) =
    13     7     1    11
     2     5     3     9
    10     8  NaN     4
    14     6    12    15

keshihua(:,:,2) =
    13     7     1    11
     2     5     3     9
    10  NaN     8     4
    14     6    12    15

keshihua(:,:,3) =
    13     7     1    11
     2     5     3     9
    10     6     8     4
    14  NaN    12    15

keshihua(:,:,4) =
    13     7     1    11
     2     5     3     9
    10     6     8     4
    14    12  NaN    15

keshihua(:,:,5) =
    13     7     1    11
     2     5     3     9
    10     6     3     4
    14    12     8    15

    2     5     3     9
    10     6  NaN     4
    14    12     8    15

keshihua(:,:,6) =
    13     7     1    11
     2     5  NaN     9
    10     6     3     4
    14    12     8    15

keshihua(:,:,7) =
    13     7     1    11
     2     5     9  NaN
    10     6     3     4
    14    12     8    15

keshihua(:,:,8) =
    13     7     1  NaN
     2     5     9    11
    10     6     3     4
    14    12     8    15

keshihua(:,:,9) =
    13     7  NaN     1
     2     5     9    11
    10     6     3     4
    14    12     8    15
```

keshihua(:, :, 10) =

13	NaN	7	1
2	5	9	11
10	6	3	4
14	12	8	15

keshihua(:, :, 16) =

13	9	5	1
NaN	2	7	11
10	6	3	4
14	12	8	15

keshihua(:, :, 11) =

13	5	7	1
2	NaN	9	11
10	6	3	4
14	12	8	15

keshihua(:, :, 17) =

13	9	5	1
10	2	7	11
NaN	6	3	4
14	12	8	15

keshihua(:, :, 12) =

13	5	7	1
2	9	NaN	11
10	6	3	4
14	12	8	15

keshihua(:, :, 18) =

13	9	5	1
10	2	7	11
14	6	3	4
NaN	12	8	15

keshihua(:, :, 13) =

13	5	NaN	1
2	9	7	11
10	6	3	4
14	12	8	15

keshihua(:, :, 19) =

13	9	5	1
10	2	7	11
14	6	3	4
12	NaN	8	15

keshihua(:, :, 14) =

13	NaN	5	1
2	9	7	11
10	6	3	4
14	12	8	15

keshihua(:, :, 20) =

13	9	5	1
10	2	7	11
14	6	3	4
12	8	NaN	15

keshihua(:, :, 15) =

13	9	5	1
2	NaN	7	11
10	6	3	4
14	12	8	15

keshihua(:, :, 21) =

13	9	5	1
10	2	7	11
14	6	3	4
12	8	15	NaN

keshihua(:, :, 22) =

13	9	5	1
10	2	7	11
14	6	3	NaN
12	8	15	4

keshihua(:, :, 28) =

13	9	5	1
10	2	NaN	3
14	6	11	7
12	8	15	4

keshihua(:, :, 23) =

13	9	5	1
10	2	7	11
14	6	NaN	3
12	8	15	4

keshihua(:, :, 29) =

13	9	5	1
10	NaN	2	3
14	6	11	7
12	8	15	4

keshihua(:, :, 24) =

13	9	5	1
10	2	NaN	11
14	6	7	3
12	8	15	4

keshihua(:, :, 30) =

13	9	5	1
10	6	2	3
14	NaN	11	7
12	8	15	4

keshihua(:, :, 25) =

13	9	5	1
10	2	11	NaN
14	6	7	3
12	8	15	4

keshihua(:, :, 31) =

13	9	5	1
10	6	2	3
14	11	NaN	7
12	8	15	4

keshihua(:, :, 26) =

13	9	5	1
10	2	11	3
14	6	7	NaN
12	8	15	4

keshihua(:, :, 32) =

13	9	5	1
10	6	2	3
14	11	15	7
12	8	NaN	4

keshihua(:, :, 27) =

13	9	5	1
10	2	11	3
14	6	NaN	7
12	8	15	4

keshihua(:, :, 33) =

13	9	5	1
10	6	2	3
14	11	15	7
12	NaN	8	4

keshihua(:, :, 34) =

13	9	5	1
10	6	2	3
14	NaN	15	7
12	11	8	4

keshihua(:, :, 35) =

13	9	5	1
10	6	2	3
14	15	NaN	7
12	11	8	4

keshihua(:, :, 36) =

13	9	5	1
10	6	2	3
14	15	7	NaN
12	11	8	4

keshihua(:, :, 37) =

13	9	5	1
10	6	2	NaN
14	15	7	3
12	11	8	4

keshihua(:, :, 38) =

13	9	5	1
10	6	NaN	2
14	15	7	3
12	11	8	4

keshihua(:, :, 39) =

13	9	5	1
10	NaN	6	2

14	15	7	3
12	11	8	4

keshihua(:, :, 40) =

13	9	5	1
NaN	10	6	2
14	15	7	3
12	11	8	4

keshihua(:, :, 41) =

13	9	5	1
14	10	6	2
NaN	15	7	3
12	11	8	4

keshihua(:, :, 42) =

13	9	5	1
14	10	6	2
15	NaN	7	3
12	11	8	4

keshihua(:, :, 43) =

13	9	5	1
14	10	6	2
15	11	7	3
12	NaN	8	4

keshihua(:, :, 44) =

13	9	5	1
14	10	6	2
15	11	7	3
NaN	12	8	4