

# 基于编码的状态空间表示方法

闫海珍, 李绪成

(大连东软信息学院 计算机科学与技术系, 辽宁 大连 116023)

**摘要:**传统状态空间表示方法中,一个状态元素需要一个单独的值来表示,每个状态需要多个值来表示,在状态转移过程中需要拷贝这些状态值。为了减少状态转移中拷贝状态所占用的时间以及状态本身所占用的空间,对状态元素进行编码。使用一个值表示多个状态元素,通过位运算实现局部状态的修改。当每个状态中包含的元素个数比较多而元素种类不多,并且每次状态转移所修改的元素数量比较少的时候,采用该方法,状态转移的速度会有显著提高,占用的空间会显著降低。

**关键词:**状态;编码;状态转移;位运算;状态拷贝

**中图分类号:**TP18

**文献标识码:**A

**文章编号:**1673-629X(2010)07-0136-04

## State Space Representation Method Based on Encode

YAN Hai-zhen, LI Xu-cheng

(Department of Computer Science & Technology, Dalian Neusoft Institute of Information, Dalian 116023, China)

**Abstract:** In traditional methods, it needs a single value to express one state element. A state must be expressed by several values and these values need to be copied during the progress of state transfer. To reduce the state copy time in state transfer and memory to store state, every state element can be encoded. Several state elements can be represented as one value and the state elements can be edited through bit operation. When the quantity of state elements is much and the kinds of state elements is less and the quantity of state elements needed to edit in every state transfer is less, the speed of state transfer is improved distinctly and the space used by state is reduced distinctly.

**Key words:** state; encode; state transfer; bit operation; state copy

## 0 引言

2009年在日本举行的高等专门学校编程大赛竞技组题目是:有 $n \times n$ 的方格,分别有 $k$ 种颜色,其中 $n \leq 20, k \leq 8$ ,图1是 $6 \times 6$ 的方格,有3种颜色。要求通过转动方格把相同的颜色转动到一起(只要互相连接即可,例如图1左上角矩形中第1行第2、3列,第2行第3、4、5列,第3行第4列相互连接),转动的次数最少获胜。每次转动的方格是一个矩形,可以是4个(半径是1,例如图1中左上角的4个)、16个(半径为2,例如图1中右下角的16个)、36个(半径为3)等, $n$ 为偶数则最大半径为 $n/2$ ,否则为 $(n-1)/2$ 。每个矩形可以有3种转法:顺时针转动90度,如图1右上角所示;可以转动180度,如图1左下角所示;可以转动270度,如图1右

下角所示。

在进行解空间搜索的时候,需要根据当前状态生成各种可能的状态,从当前状态获取下一层目标状态的时候,需要把当前状态中的所有方格的颜色都复制到下一层的所有状态中。在 $n$ 比较大的时候,需要赋值 $n \times n$ 次,赋值操作占用了大部分的时间。

要表示图1的状态信息,传统的方法是采用 $n \times n$ 的二维数组,每种颜色使用一个数字来表示。图1中的颜色可以表示为:

```
{
  {1,2,2,3,3,3},
  {1,1,2,2,2,3},
  {3,1,1,2,3,3},
  {1,2,2,3,3,1},
  {1,2,2,2,3,3},
  {1,2,3,3,2,2}
}
```

计为color1。

采用这种表示方法,从图1左上角的状态生成图

收稿日期:2009-11-10;修回日期:2010-02-21

基金项目:大连东软信息学院青年科研基金项目(NEUSOFTIIT 20080010)

作者简介:闫海珍(1979-),女,山西原平人,讲师,硕士,研究方向为数据挖掘、智能计算。

1 中其他 3 种状态的时候,需要把 color1 中的每个值赋值给 color2、color3 和 color4 (color2、color3 和 color4 分别是表示图 1 右上、左下、右小部分的颜色数组),然后再修改旋转所影响的左上角的 4 个元素的值,这个过程存在大量的赋值操作。因为在求目标解的过程中主要的操作就是生成新的状态并判断是不是目标状态,所以减少赋值操作可以显著提供性能<sup>[1,2]</sup>。文中通过对颜色进行编码来减少赋值操作的次数来提高效率<sup>[3]</sup>,同时也减少了状态所占用的空间。

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | 3 | 3 | 3 | 1 | 1 | 2 | 3 | 3 | 3 |
| 1 | 1 | 2 | 2 | 2 | 3 | 1 | 2 | 2 | 2 | 2 | 3 |
| 3 | 1 | 1 | 2 | 3 | 3 | 3 | 1 | 1 | 2 | 3 | 3 |
| 1 | 2 | 2 | 3 | 3 | 1 | 1 | 2 | 2 | 3 | 3 | 1 |
| 1 | 2 | 2 | 2 | 3 | 3 | 1 | 2 | 2 | 2 | 3 | 3 |
| 1 | 2 | 3 | 3 | 2 | 2 | 1 | 2 | 3 | 3 | 2 | 2 |

|   |   |   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 3 | 3 | 2 | 1 | 2 | 3 | 3 | 3 |
| 2 | 1 | 2 | 2 | 2 | 3 | 1 | 1 | 2 | 2 | 2 | 3 |
| 3 | 1 | 1 | 2 | 3 | 3 | 3 | 1 | 1 | 2 | 3 | 3 |
| 1 | 2 | 2 | 3 | 3 | 1 | 1 | 2 | 2 | 3 | 3 | 1 |
| 1 | 2 | 2 | 2 | 3 | 3 | 1 | 2 | 2 | 2 | 3 | 3 |
| 1 | 2 | 3 | 3 | 2 | 2 | 1 | 2 | 3 | 3 | 2 | 2 |

图 1 方格示例及旋转后情况

## 1 对颜色进行编码

传统方法采用  $n \times n$  的数组来表示方格中的所有颜色。文中通过编码把每行的颜色表示成 1 个整数。对于  $n \times n$  的矩形,采用长度为  $n$  的数组,每个值是该行颜色的编码,假设为  $value[n]$ ,则

$$value[i] = \sum_{j=0}^{n-1} color[i][j] * 2^{m*j}$$

其中,  $n$  表示矩形的规模,  $color[i][j]$  表示第  $i$  行第  $j$  个方格的颜色编码,  $m$  表示每种颜色需要的编码位数,两种颜色使用 1 位编码即可表示,3 种或者 4 种颜色使用 2 位编码表示,5 至 8 种颜色使用 3 位编码即可<sup>[4]</sup>。

下面以 4 种颜色为例介绍如何对颜色进行编码,4 种颜色需要使用两位编码表示。

红:00

黄:01

蓝:10

白:11

一行的颜色使用一个整数表示,例如“红 黄 蓝 白 黄 红 白 蓝”。转换过程如表 1 所示。

表 1 颜色转换过程

| 颜色 | 红  | 黄  | 蓝  | 白  | 黄  | 红  | 白  | 蓝  |
|----|----|----|----|----|----|----|----|----|
| 编码 | 00 | 01 | 10 | 11 | 01 | 00 | 11 | 10 |
| 整数 | 0  | 1  | 2  | 3  | 1  | 0  | 3  | 2  |

编码:  $0 * 1 + 1 * 4 + 2 * 16 + 3 * 64 + 1 * 256 + 0 * 1024 + 3 * 4096 + 2 * 16384 = 45540$ 。

如果  $k < 5, n > 16, n \leq 20$ , 每种颜色需要两个 bit 位来表示,  $n$  在 16 和 20 之间, 所以表示一行颜色需要使用的位数在  $2 * 16$  到  $2 * 20$  之间, 即 32 到 40 位, 所以每行的颜色可以使用 long 类型(64 位) 表示;

如果  $k < 5, n \leq 16$ , 每种颜色需要两个 bit 位来表示,  $n$  小于等于 16, 所以表示一行颜色需要使用的位数小于等于  $2 * 16$ , 即 32 位, 所以每行颜色可以使用 int 类型(32 位) 表示;

如果  $k \geq 5, k \leq 8, n > 10, n < 20$ , 每种颜色需要 3 个 bit 位来表示,  $n$  在 10 和 20 之间, 所以表示一行颜色需要的位数为  $3 * 10$  到  $3 * 20$  之间, 即 30 到 60 之间, 每行的颜色使用 long 类型表示;

如果  $k \geq 5, k \leq 8, n \leq 10$ , 每种颜色需要 3 位, 表示一行颜色需要的位数小于  $3 * 10$ , 每行的颜色使用 int 类型表示。

为了尽可能高地提高效率, 在程序中可以动态调整元素的类型<sup>[5,6]</sup>。

## 2 采用编码方式的状态转移

状态转移的操作主要包括: 把原始状态中的颜色全部复制到新的状态中, 然后修改因转动而影响到的颜色, 例如转动 4 个方格, 会影响相应的 4 个位置的颜色。

假设表示原状态的颜色数组为 originalColors, 表示当前状态的颜色数组为 newColor。

### 2.1 状态复制

状态复制需要把原来表示颜色的  $n$  个整数复制到新的状态中。过程如下:

```
for(int i=0; i<n; i++) {
    newColor[i] = originalColors[i];
}
```

### 2.2 局部状态修改

修改某个位置的颜色需要多步操作, 例如要把  $(x2, y2)$  位置的颜色赋给  $(x1, y1)$  位置, 首选需要从表示  $(x2, y2)$  所在行的颜色的整数中解析出具体的颜色, 然后把这个颜色再编码到表示  $(x1, y1)$  所在行的颜色的整数中。

修改颜色的具体操作过程如下:

1) 取出  $(x2, y2)$  位置的颜色在当前行中的编码。

\* 根据  $(x2, y2)$  所在的列构造一个值, 该列对应位置为全 1 编码, 而其它列为全 0。

```
int temp2 = 3 << (2 * y2);
```

如果使用 3 位表示颜色, 则需要把 3 改为 7, 分别

为 2 进制的 11 和 111。

假设  $y_2$  为 3, 得到的整数是:

0000 0000 0000 0000 0000 0000 1100 0000

这里的代码假设每行颜色使用 int 类型的变量表示, 如果使用 long 类型表示, temp2 需要定义为 long 类型。

\* 进行与运算, 得到原来的颜色。

int temp3 = originalColors[x2] & temp2;

originalColors[x2] 表示第  $x_2$  行的颜色编码, 与 temp2 进行与运算, 得到该颜色的编码, 过程如下:

0000 0000 0000 0000 0001 1011 0100 1110

& 0000 0000 0000 0000 0000 0000 1100 0000

结果 0000 0000 0000 0000 0000 0000 0100 0000

这里得到的结果是当前颜色在该行中的编码值, 例如当前例子中的 64, 而不是颜色的编码值, 但是再经过移位可以得到具体的颜色编码 01, 即黄色。这里没有进行移位操作, 是因为使用该颜色设置新的位置的颜色的时候还要进行移位操作, 假设要设置第 4 列的颜色, 需要把 01 左移 8 位, 所以为了减少操作在后面的移位中直接移位差值即可<sup>[7]</sup>。

2) 删除  $(x_1, y_1)$  位置的颜色。

value[x1] = value[x1]

& (Integer.MAX\_VALUE - (3 << (2 \* y1)));

过程如下(假设  $y_1$  为 0):

原颜色: 0000 0000 0000 0000 0001 1011 0100

1110

& 1111 1111 1111 1111 1111 1111 1111 1100

结果 0000 0000 0000 0000 0001 1011 0100 1100

根据要修改的颜色的位置把原来的颜色删除掉, 用 0 进行与操作, 为了不影响其他位置, 使用 1 进行与操作。这样就把颜色直接删除调用了。

3) 把从  $(x_2, y_2)$  位置获取的颜色赋值给  $(x_1, y_1)$  位置。

\* 计算原来的颜色在新的位置的编码。

if( $y_2 > y_1$ )

temp4 = temp3 >> (2 \*  $y_2$  - 2 \*  $y_1$ );

else

temp4 = temp3 << (2 \*  $y_1$  - 2 \*  $y_2$ );

原位置: 0000 0000 0000 0000 0000 0000 0100

0000

新位置: 0000 0000 0000 0000 0000 0000 0000

0001

经过这个计算就可得到旋转后的颜色编码。

\* 修改  $(x_1, y_1)$  所在行的颜色编码。

value[x1] = value[x1] + temp4;

或者

value[x1] = value[x1] | temp4;

设置前: 0000 0000 0000 0000 0001 1011 0100

1100

新颜色: 0000 0000 0000 0000 0000 0000 0000

0001

结果: 0000 0000 0000 0000 0001 1011 0100 1101

这样把  $(x_2, y_2)$  位置的颜色赋给了  $(x_1, y_1)$  位置。

### 3 占用空间分析与比较

下面对采用编码方法和采用原始方法这两种方式下的表示每种状态所占的空间进行比较。假设可能的状态数  $k$ , 规模为  $n$ 。采用传统方法所使用的空间为  $S_1 = n * n$  个字节(假设  $k < 256$ , 因为  $k$  的最大值为 8 所以满足)。

采用改进方法所使用的空间为:

$$S_2 = n * (n * \log_2 k) / 8$$

单位: 字节, 其中  $\log_2 k$  表示每种颜色所占用的 bit 位。

$$\begin{aligned} S_2/S_1 &= (n * (n * \log_2 k) / 8) / (n * n) \\ &= \log_2 k / 8 \end{aligned}$$

因为  $k$  小于 256,  $\log_2 k$  小于 8,  $\log_2 k / 8$  小于 1, 所以  $S_2 < S_1$ 。假设传统方法占用的空间为  $S_1$ , 改进方法占用的空间为  $S_2$ , 则表 2 列出了  $k$  为 1~4 和 5~8 的时候两种方法占用空间的对比情况。

表 2  $S_1$  和  $S_2$  占用空间比较

|       | $1 \leq k \leq 4$ |     |     |     | $5 \leq k \leq 8$ |     |     |     |
|-------|-------------------|-----|-----|-----|-------------------|-----|-----|-----|
|       | 8                 | 12  | 16  | 20  | 8                 | 12  | 16  | 20  |
| $S_1$ | 64                | 144 | 256 | 400 | 64                | 144 | 256 | 400 |
| $S_2$ | 16                | 36  | 64  | 100 | 24                | 54  | 96  | 150 |

占用空间的对比如图 2 所示。

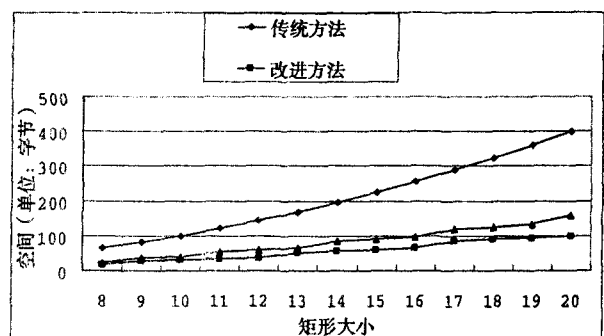


图 2 占用空间对比

从表中可以看出当  $k$  比较小的时候, 采用编码方式可以节省更多的空间。而在文中所要解决的问题中,  $k$  的最大值为 8。所以采用改进方法能够显著降低存储空间。

## 4 时间性能分析与比较

从原始状态生成新的状态的主要操作包括两部分操作:

- \* 把原始状态全部状态值复制到新的状态中;
- \* 在新的状态中修改操作所影响的状态值,这个通常都是局部。

采用传统方法,从原始状态生成新的状态所使用的计算需要的操作次数为:

$$n * n + x$$

其中  $n * n$  用于拷贝  $n * n$  个方格的颜色,  $x$  表示需要修改的状态值的个数,例如转动半径为 2,要转动的单元格为 16 个,则需要修改的颜色  $x$  等于 16。

采用改进方法,从原始状态生成新的状态所使用的计算需要  $n + x * k$ ,其中  $n$  用于拷贝状态,应为每行颜色使用一个数字表示,每种状态使用  $n$  个数字表示,  $x$  表示需要修改的状态值的个数,  $k$  表示修改 1 个状态值需要的代价,因为某个位置的颜色被编码在一个数字中,所以修改的时候需要从原来所在行解析出颜色,然后再编码到新的行中,  $k$  就表示该过程所占用的时间。

$x$  由旋转的半径决定,假设旋转半径为  $r$ ,则:

$$x = 4 * r * r$$

其中  $r$  的最大值由  $n$  决定,如果  $n$  为偶数,  $r$  的最大值为  $n/2$ ,否则  $r$  的最大值为  $(n-1)/2$ 。对于一个具体的状态转移  $x$  与  $r * r$  成正比。但是从当前状态获取下一层状态的时候,对于不同的  $r$  状态数差别很多。例如对于  $10 * 10$  的方格,  $r$  等于 1 的时候可能的目标状态有 243 种,当  $n$  等于 5 的时候,可能的目标状态只有 3 种,状态数 count 与  $r$  的关系如下:

$$\text{count} = 3 * (n - 2 * r + 1) * (n - 2 * r + 1)$$

$x$  的平均值为:

$$\frac{\sum_{r=1}^{(n+1)/2} 3 * (n - 2 * r + 1) * (n - 2 * r + 1) * 4 * r * r}{\sum_{r=1}^{(n+1)/2} 3 * (n - 2 * r + 1) * (n - 2 * r + 1)}$$

图 3 是  $n$  从 10 到 20 情况下,从原始状态生成新的状态需要的平均时间对比。

实际上在搜索状态空间的时候不会获取所有下一层状态,而是根据估值选择某些状态。通过实验发现,  $r$  比较小的时候被选择的概率更大一些,所以在实际操作中,  $x$  的值更小,所以采用改进算法的性能更高<sup>[8,9]</sup>。

通常  $x$  比较小,  $k$  固定并且比较小,这样生成新状态的主要时间在状态拷贝上,而改进算法的时间与规模  $n$  成线性关系,所以效率有很大改善。

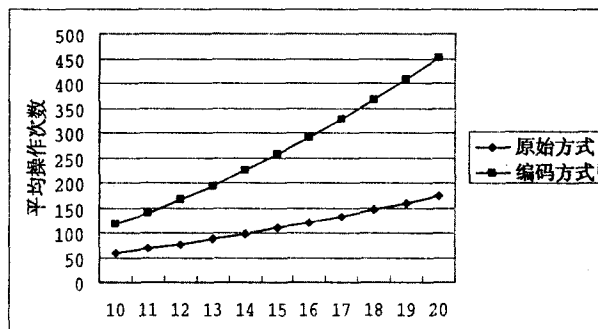


图 3 操作次数对比

## 5 性能测试

使用 Java 语言编写程序对文中的方法进行了测试<sup>[10]</sup>。测试过程没有考虑估值,所以每次在获取下一层的状态的时候获取的是所有的状态,而在实际运行中会根据估值选择目标状态,所以与测试数据本身会有差别。

测试过程如下:

- 1) 采用原始方法,使用 4 种颜色分别对  $n$  等于 8 到  $n = 20$  的情况进行测试,并记录结果。
- 2) 采用编码方法,使用 4 种颜色分别对  $n$  等于 8 到  $n$  等于 20 的情况进行了测试。
- 3) 采用原始方法,使用 8 种颜色分别对  $n$  等于 8 到  $n = 20$  的情况进行测试,并记录结果。
- 4) 采用编码方法,使用 8 种颜色分别对  $n$  等于 8 到  $n$  等于 20 的情况进行了测试。

为了使测试数据更有说服力,对测试过程进行了改进:

- \* 对每种情况进行了 3 次运行,然后取平均值。
- \* 测试每种情况的时候,让代码重复执行 100 次,然后取平均值。

图 4 显示了采用 4 种颜色时候改进方法与传统方法的性能比较,横坐标是矩形的大小从  $11 * 11$  到  $20 * 20$ ,纵坐标是从当前状态得到可能的下一层状态所需要的时间,下面的线表示改进方法,上面的线表示传统方法。可以看出改进方法的性能优势非常明显。

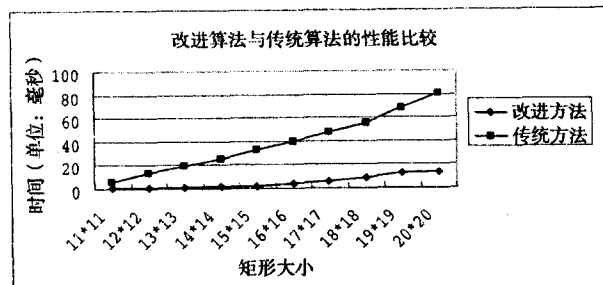


图 4 改进算法与传统算法的性能比较

(下转第 144 页)

## 4 结束语

文中提出了一种基于 DWT 与 SVD 相结合的多功能水印新方案,该方案利用 DWT 特性和 SVD 分解的稳定性,并保持了载体图像的不可视性和鲁棒性的均衡性。通过实验结果和攻击测试得出,该算法能够对剪切攻击、JPEG 压缩攻击、噪声攻击、拼凑攻击等图像处理具有很好的抵御能力,尤其是对剪切、JPEG 压缩等攻击。同时,能有效检测图像恶意攻击和进行篡改定位。但此方法在任意旋转攻击提取出的水印不够理想,希望在将来的工作中进一步的改进。总的来说,本算法在检测过程中不需要原始水印和原始图像,可以克服载体媒体占用空间过大的缺点,此外嵌入和提取算法简单,运行速度比较快速,可以说一种比较理想的数字水印方案。

### 参考文献:

- [1] Cao J G, Fowler J E, Younan N H. An Image - Adaptive Watermark Based on a Redundant Wavelet Transform [C]//in Proceedings of the International Conference on Image Processing. Thessaloniki, Greece: IEEE, 2001: 277 - 280.
- [2] Kourkchi H, Ghaemmaghami S. Image Adaptive Semi - fragile Watermarking Scheme Based on RDWT - SVD [C]//in proceeding of the 5th international conference in innovations in in-

formation technology. United Arab Emirates: [s. n.], 2008.

- [3] 于帅珍,沈建国.基于小波域的自适应彩色图像双重水印算法[J].微计算机信息,2006,22(1):190 - 192.
- [4] Lee T Y, Lin S D. Dual Watermark for Image Tamper Detection and Recovery [J]. Pattern Recognition, 2008, 41: 3497 - 3506.
- [5] Ho Seok Moon, Myung Ho Sohn, Dong Sik Jang. DWT-based image watermarking for copyright protection [J]. Artificial Intelligence and Simulation, 2004(3397): 490 - 497.
- [6] 李养胜,李俊.基于小波分解的图像数字水印算法[J].计算机技术与发展,2008,18(12):150 - 152.
- [7] 马建湖,何甲兴.基于小波变换的零水印算法[J].中国图象图形学报,2007,12(4):581 - 585.
- [8] Chang Chin - Chen, Tsai Piyu, Lin Chia - Chen. SVD - based digital image watermarking scheme [J]. Pattern Recognition Letters, 2005, 10(26): 1577 - 1586.
- [9] 刘瑞祯,谭铁牛.基于奇异值分解的数字图像水印方法[J].电子学报,2001,29(2):168 - 171.
- [10] 吴亚丽.置乱的新型 W 变换矩阵[J].电子科技,2008,21(3):69 - 72.
- [11] 许红山.置乱技术在信息隐藏中的应用[J].广州大学学报,2004,3(2):134 - 136.
- [12] 董梅,高康.矩阵奇异值分解和 Arnold 置乱技术在图像隐藏中的应用[J].山东大学学报:理学版,2005,40(3):71 - 75.

(上接第 139 页)

## 6 结束语

通过对颜色值进行编码,使用一个值表示多种颜色,从而减少了存储状态所需要的空间,同时因为表示状态所需要的变量减少,在进行状态转移的时候赋值次数减少,赋值所需要的时间也显著减少。在空间和性能方面都有提高。但是文中采用的方法也存在一个缺点:因为采用一个值表示多个元素,修改某个具体元素需要多步操作。所以当状态中包含的元素比较多,而元素的种类比较少,并且状态转移所需要修改的元素比较少的时候,文中提出的方法具有显著的优势<sup>[11,12]</sup>。

### 参考文献:

- [1] 王永庆.人工智能原理与方法[M].西安:西安交通大学出版社,1998.
- [2] 刘志镜,王小愚,李绪成,等.一种新的项集表示方法[J].计算机工程与设计,2002,23(6):42 - 44.
- [3] 许精明.状态空间的启发式搜索方法研究[J].微机发展(现更名:计算机技术与发展),2002,12(4):87 - 79.
- [4] 满君丰,刘强,杨鼎.空间数据的表示方法研究[J].计

算机应用,2004(11):97 - 99.

- [5] 徐宝祥,叶培华.知识表示的方法研究[J].情报科学,2007(5):690 - 694.
- [6] 年志刚,梁式,麻芳兰,等.知识表示方法研究与应用[J].计算机应用研究,2007(5):234 - 236.
- [7] 陈华竣,郑智,倪德明.真前缀标记树——一种面向用户的子树选取策略表示方法[J].计算机技术与发展,2006,16(12):9 - 12.
- [8] Pelegri A, Shan Baoxiang. Dynamic analysis of soft tissues using a state space model [R]. US: American Society of Mechanical Engineers (ASME), 2008.
- [9] Berners - Lee T, Hendler J, Lassila O. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities [J]. Scientific American, 2001, 13(5): 100 - 105.
- [10] 汤赛丽,郑逢斌.基于问题的面向对象知识表示方法的研究[J].现代计算机,2005(8):13 - 15.
- [11] Tao JunYuan, Li DeSheng. Cooperative strategy learning in multi - agent environment with continuous state space [C]// 2006 International Conference on Machine Learning and Cybernetics. [s. l.]: [s. n.], 2006.
- [12] Zhang Chengjin, Bitmead B R. MIMO Equalization With State - Space Channel Models [J]. IEEE Transactions on Signal Processing, 2008, 56(10): 5222 - 5231.