TiD2016
质量竞争力大会
软件研发顶级盛会

SOFTWARE DEVELOPMENT
下一代软件研发
更高 更新 更深

AutoCAD自动化测试数据挖掘：提高自动化测试效率

(Data Mining on AutoCAD Automation; Improve Automation Efficiency)

巨霞

AUTODESK

议程

- 背景介绍
- 解决方案
- 具体算法
- 研究成果
- 未来展望

背景介绍

- AutoCAD 三十多年历史
- 千万行级代码量
- 六万多自动化测试用例
- 每年增加一千多新用例

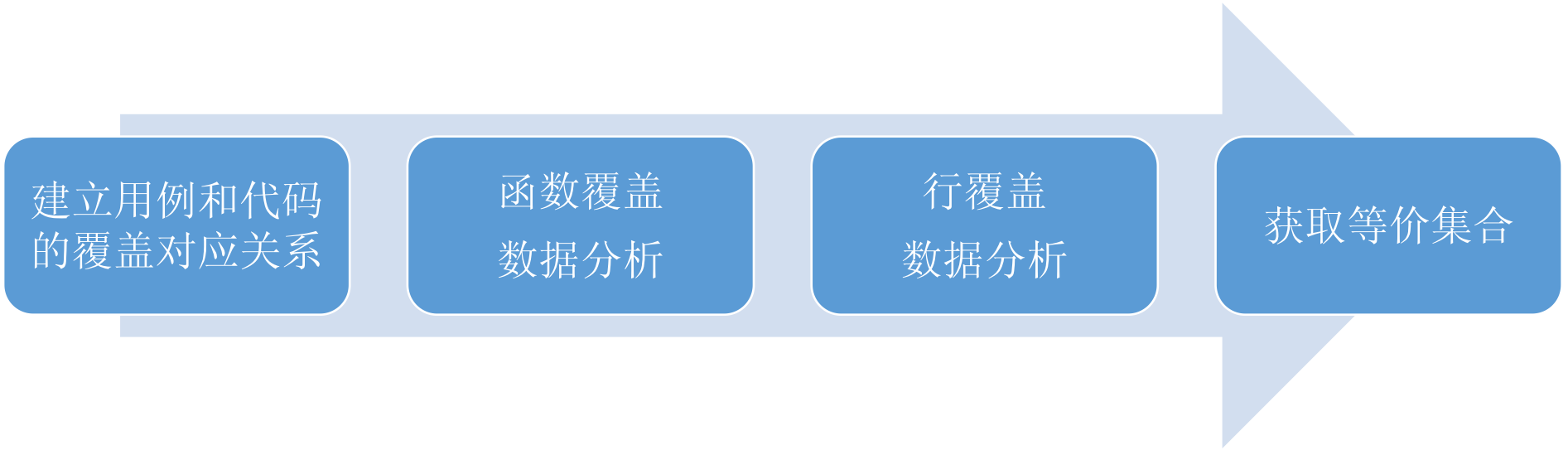
- 是否有重复?
- 能否更高效?

解决方案

通过对代码覆盖率的数据挖掘
寻找等价测试集合
提高测试效率

智能中心点功能演示

解决方案



源程序：

```
11          static void Main(string[] args)
12          {
13              if (Convert.ToBoolean(args[0]) && Convert.ToBoolean(args[1]))
14              {
15                  Console.WriteLine("covered");
16              }
17          }
18      }
19  }
```

测试用例：

测试用例	参数一	参数二
用例一	true	true
用例二	true	false
用例三	false	true
用例四	false	false

块划分：（IL代码）

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       261 (0x105)
  .maxstack  3
  .locals init ([0] bool V_0)
  IL_0000:  call       void Microsoft.VisualStudio.Coverage.Init_14cfcba2451ce11837296a904ac5ca41::Register()
  IL_0005:  ldsfld     uint64[] Microsoft.VisualStudio.Coverage.Init_14cfcba2451ce11837296a904ac5ca41::m_vscov
  IL_000a:  ldc.i4     0x5
  IL_000f:  ldelem.i8
  IL_0010:  ldc.i8     0x0
  IL_0019:  add
  IL_001a:  conv.i
  IL_001b:  ldc.i4.1
  IL_001c:  stind.i1
  IL_001d:  nop
  IL_001e:  ldarg.0
  IL_001f:  ldc.i4.0
  IL_0020:  ldelem.ref
  IL_0021:  call       bool
  IL_0026:  ldsfld     uint6
  IL_002b:  ldc.i4     0x5
  IL_0030:  ldelem.i8
  IL_0031:  ldc.i8     0x1
  IL_003a:  add
  IL_003b:  conv.i
  IL_003c:  ldc.i4.1
  IL_003d:  stind.i1
  IL_003e:  brfalse.s  IL_00
  IL_0040:  ldsfld     uint6
  IL_0045:  ldc.i4     0x5
  IL_004a:  ldelem.i8
  IL_004b:  ldc.i8     0x2
  IL_0054:  add
  IL_0055:  conv.i
  IL_0056:  ldc.i4.1
  IL_0057:  stind.i1
  IL_0058:  ldarg.0
  IL_0059:  ldc.i4.1
  IL_005a:  ldelem.ref
  IL_005b:  call       bool [mscorlib]System.Convert::ToBoolean(string)
  IL_0060:  ldsfld     uint64[] Microsoft.VisualStudio.Coverage.Init_14cfcba2451ce11837296a904ac5ca41::m_vscov
  IL_0065:  ldc.i4     0x5
  IL_006a:  ldelem.i8
  IL_006b:  ldc.i8     0x3
  IL_0074:  add
  IL_0075:  conv.i
  IL_0076:  ldc.i4.1
```

```
  IL_00ea:  nop
  IL_00eb:  nop
  IL_00ec:  ldsfld     uint64[] Microsoft.VisualStudio.Coverage.Init_14cfcba2451ce11837296a904ac5ca41::m_vscov
  IL_00f1:  ldc.i4     0x5
  IL_00f6:  ldelem.i8
  IL_00f7:  ldc.i8     0x8
  IL_0100:  add
  IL_0101:  conv.i
  IL_0102:  ldc.i4.1
  IL_0103:  stind.i1
  IL_0104:  ret
} // end of method Program::Main
```

```
  IL_0078:  br.s       IL_0093
  IL_007a:  ldsfld     uint64[] Microsoft.VisualStudio.Coverage.Init_14cfcba2451ce11837296a904ac5ca41::m_vscov
  IL_007f:  ldc.i4     0x5
  IL_0084:  ldelem.i8
  IL_0085:  ldc.i8     0x4
  IL_008e:  add
  IL_008f:  conv.i
  IL_0090:  ldc.i4.1
  IL_0091:  stind.i1
```

```
e.Init_14cfcba2451ce11837296a904ac5ca41::m_vscov
```

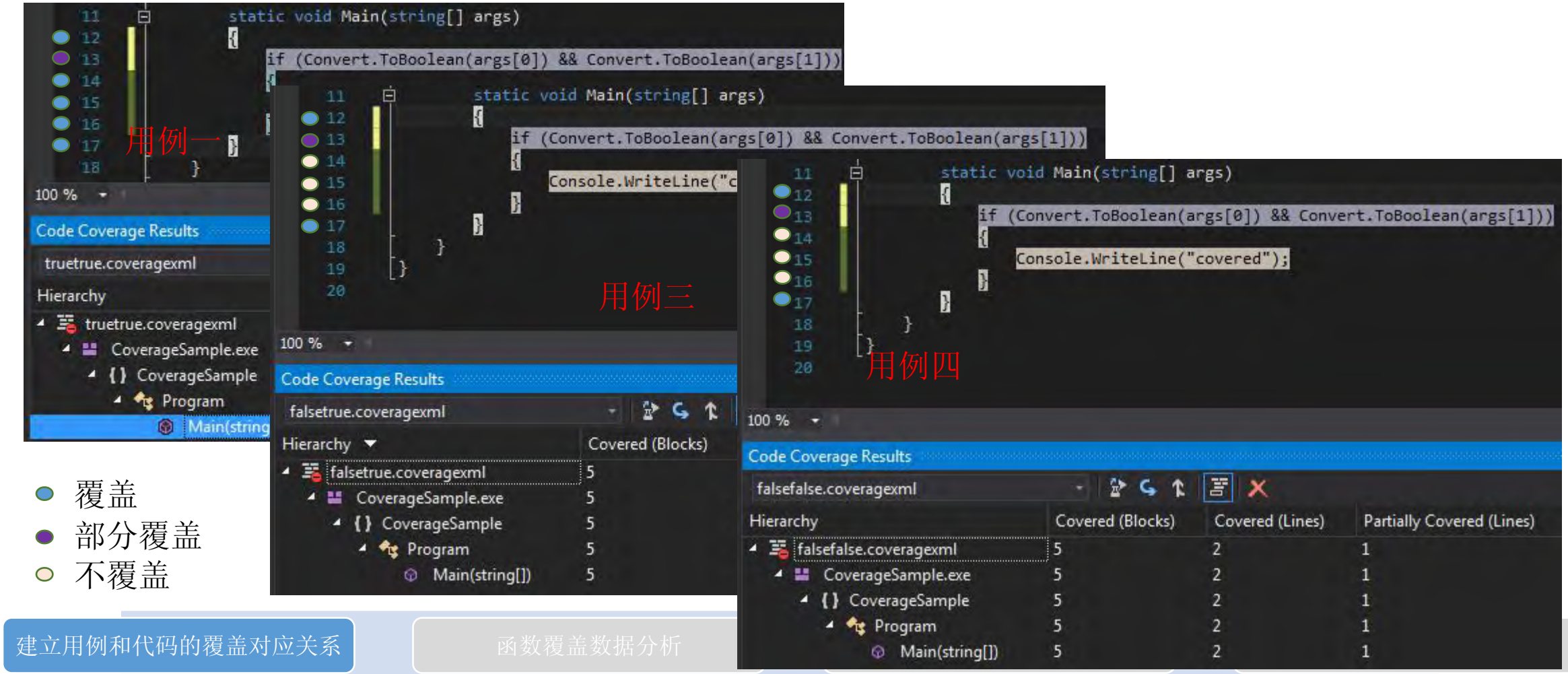
```
e.Init_14cfcba2451ce11837296a904ac5ca41::m_vscov
```

```
  IL_00c4:  conv.i
  IL_00c5:  ldc.i4.1
  IL_00c6:  stind.i1
  IL_00c7:  nop
  IL_00c8:  ldstr      "covered"
  IL_00cd:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_00d2:  ldsfld     uint64[] Microsoft.VisualStudio.Coverage.Init_14cfcba2451ce11837296a904ac5ca41::m_vscov
  IL_00d7:  ldc.i4     0x5
  IL_00dc:  ldelem.i8
  IL_00dd:  ldc.i8     0x7
  IL_00e6:  add
  IL_00e7:  conv.i
```

划分出了9个块

建立用例和代码的覆盖对应关系 | 函数覆盖数据分析 | 行覆盖数据分析 | 获取等价集合

覆盖率：



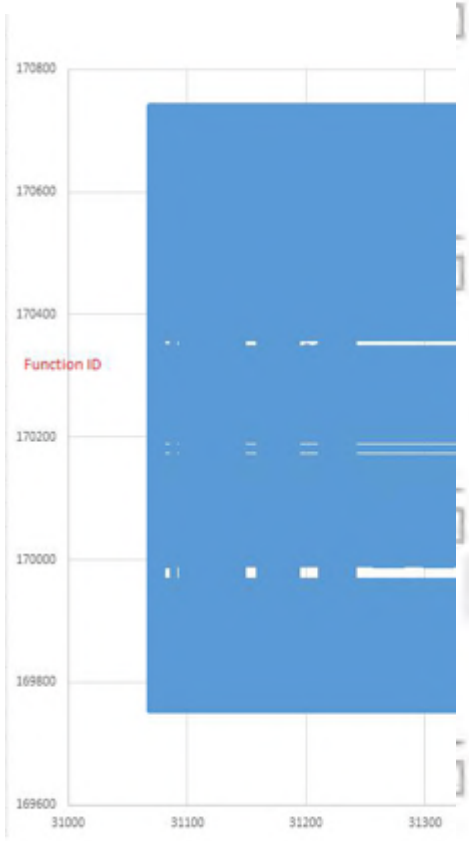
- 覆盖
- 部分覆盖
- 不覆盖

建立用例和代码的覆盖对应关系

函数覆盖数据分析

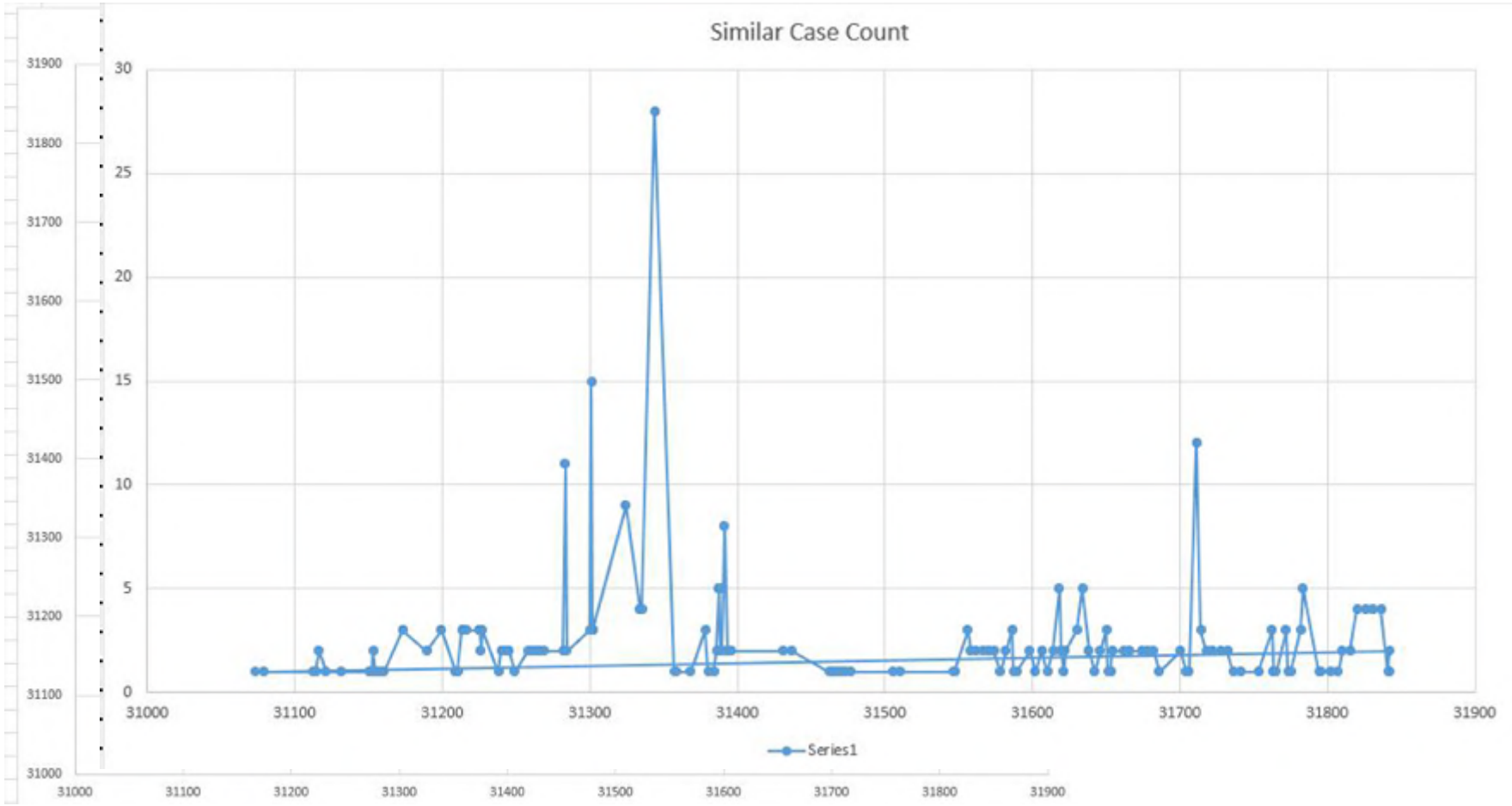
测试用例	参数一	参数二	覆盖块	行覆盖
用例一	true	true	0，1，2，3，5，6，7，8	010000
用例二	true	false	0，1，2，3，4，5，8	012220
用例三	false	true	0，1，4，5，8	012220
用例四	false	false	0，1，4，5，8	012220

0	全覆盖
1	部分覆盖
2	不覆盖




```
{
  "31069,31069": {
    "similarityFunc": 1,
    "similarityFuncLine": 1
  },
  "31069,31070": {
    "similarityFunc": 0.94,
    "similarityFuncLine": 0.9394827891649678 6
  },
  "31070,31069": {
    "similarityFunc": 0.94,
    "similarityFuncLine": 0.9394827891649678 6
  },
  "31069,31071": {
    "similarityFunc": 0.94,
    "similarityFuncLine": 0.9472937565497687 7
  }
```


建立用例和代码的覆盖对应关系 | 函数覆盖数据分析 | 行覆盖数据分析 | 获取等价集合

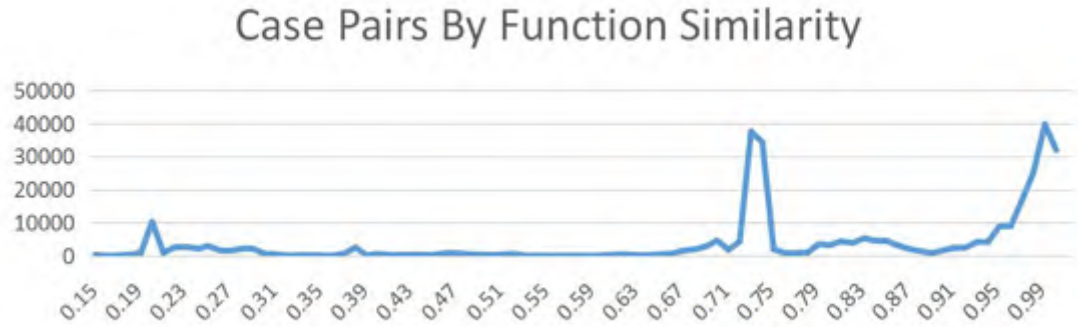


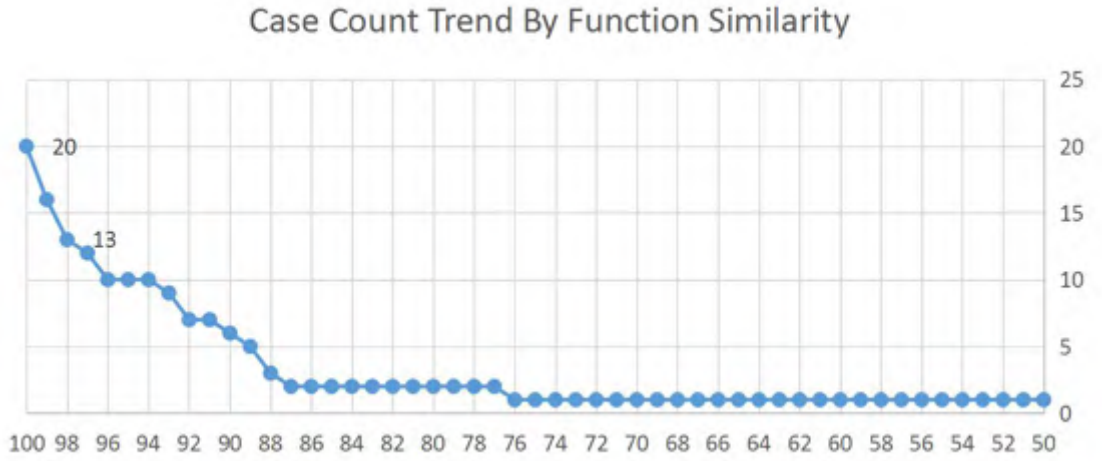
建立用例和代码的覆盖对应关系

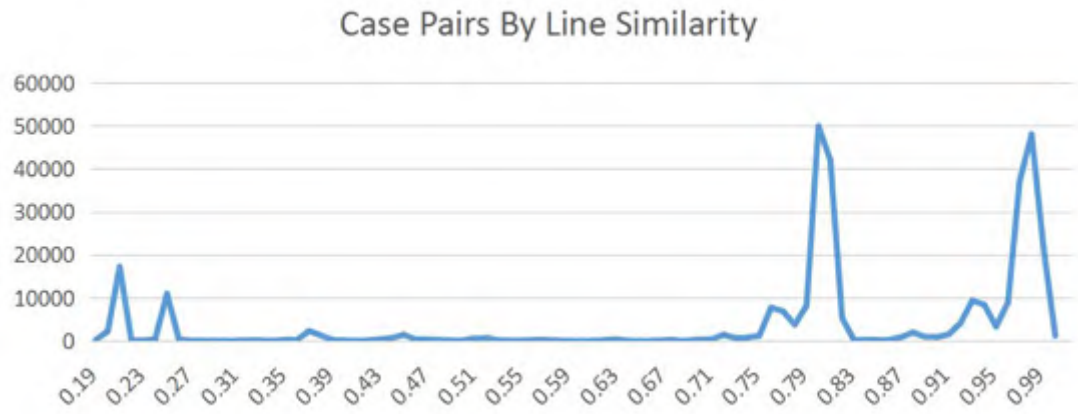
函数覆盖数据分析

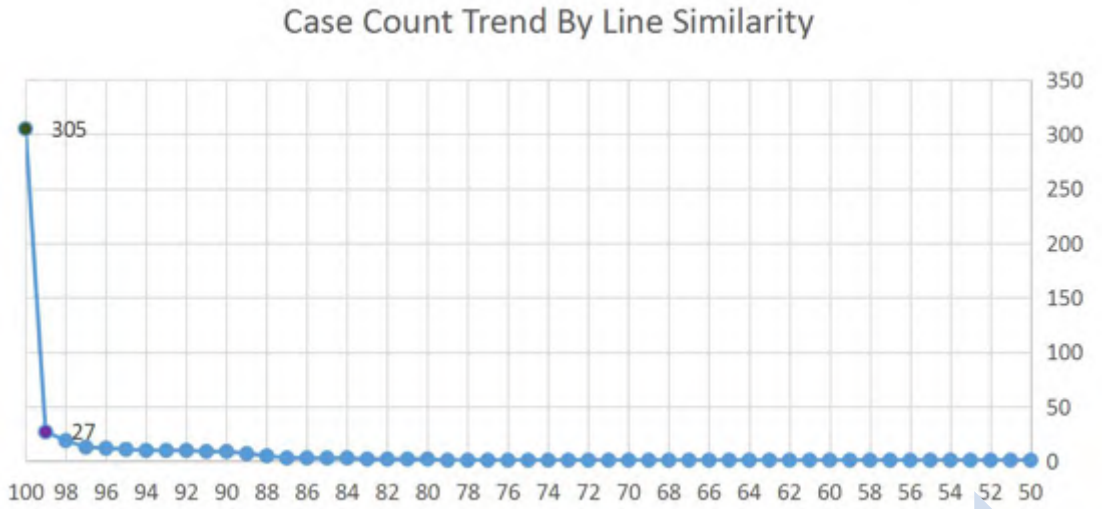
行覆盖数据分析

获取等价集合





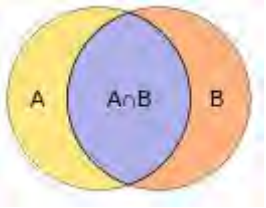


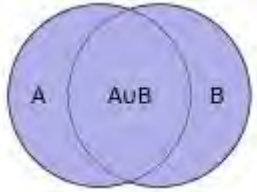


建立用例和代码的覆盖对应关系 | 函数覆盖数据分析 | 行覆盖数据分析 | 获取等价集合

具体算法

A = {f1, f3, f6, f8, f9}
B = {f3, f4, f7, f8, f10}





A ∩ B={f3, f8}
A ∪ B={f1, f3, f4, f6, f7, f8, f9, f10}
相似度 = |A∩B| / |A∪B|=2/8=0.25

具体算法

Jaccard

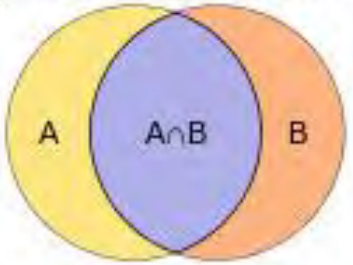
假如有集合 A、B，那么

J(A,B)=（A intersection B）/（A union B)

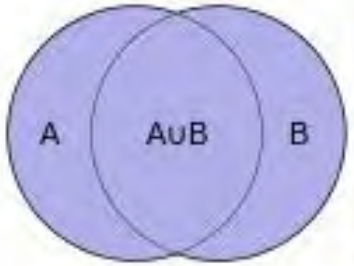
也就是说，集合 A,B 的 Jaccard 系数等于 A,B 中共同拥有的元素数与 A,B 总共拥有的元素数的比例。很显然，Jaccard 系数值区间为[0,1]。

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

(If A and B are both empty, we define J(A,B) = 1.)



Intersection of two sets A and B



Union of two sets A and B

具体算法

MinHash

我们随机从两个集合中各挑选一个元素f(A)、f(B)，刚好这两个无素相同的概率是多少呢？

概率就是**在A∪B这个大的随机域里，选中的元素落在A ∩*B* 这个区域的概率，这个概率就等于Jaccard的相似度！**

具体算法

函数相似度：

- 对于函数的相似度，我们直接MinHash来计算两个测试用例的函数相似度
- 对于两个测试用例的行相似度，我们做了一些调整来减少大函数对于行相似度结果的影响

如何消除大函数对于行相似度的影响？

初始思路
- 展开,MinHash

初始结果
- 相似度受大函数影响

改进
- 降低大函数中行的权重

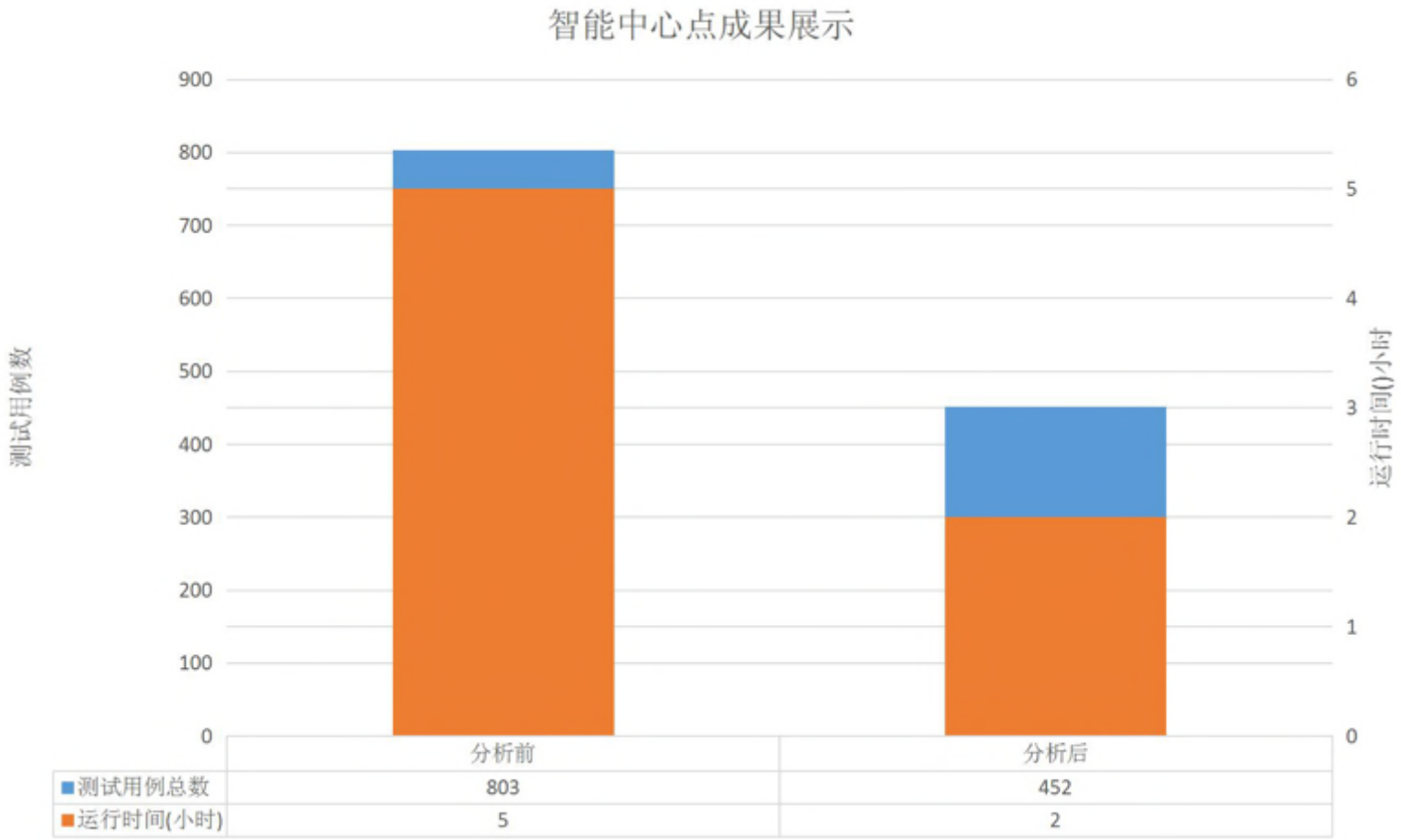
改进结果
- 随机抽取样本，代码覆盖率

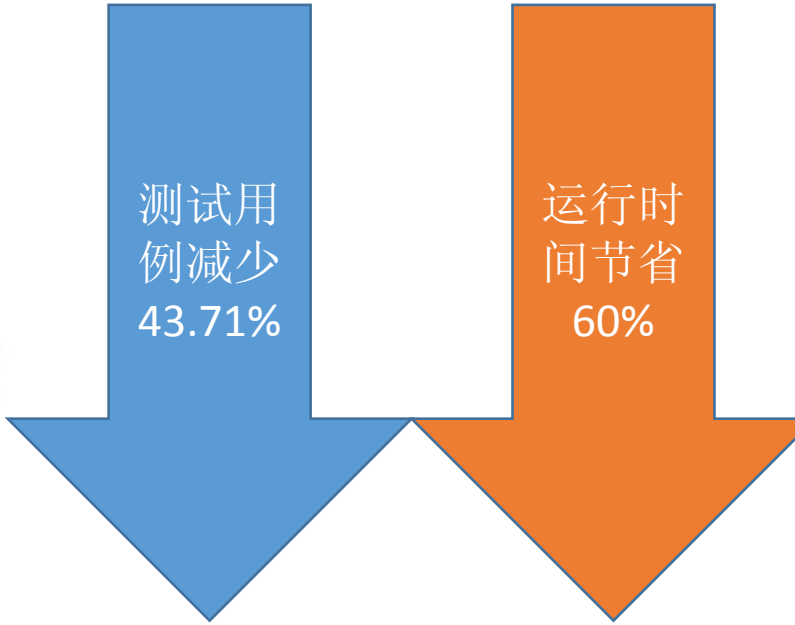
$$\text{Linesimilarity} = \frac{\text{Sum(Jac(line1)+.....+Jac(lineN))}}{\text{FunctionunionCount}}$$

行相似度中，对于部分覆盖的处理：

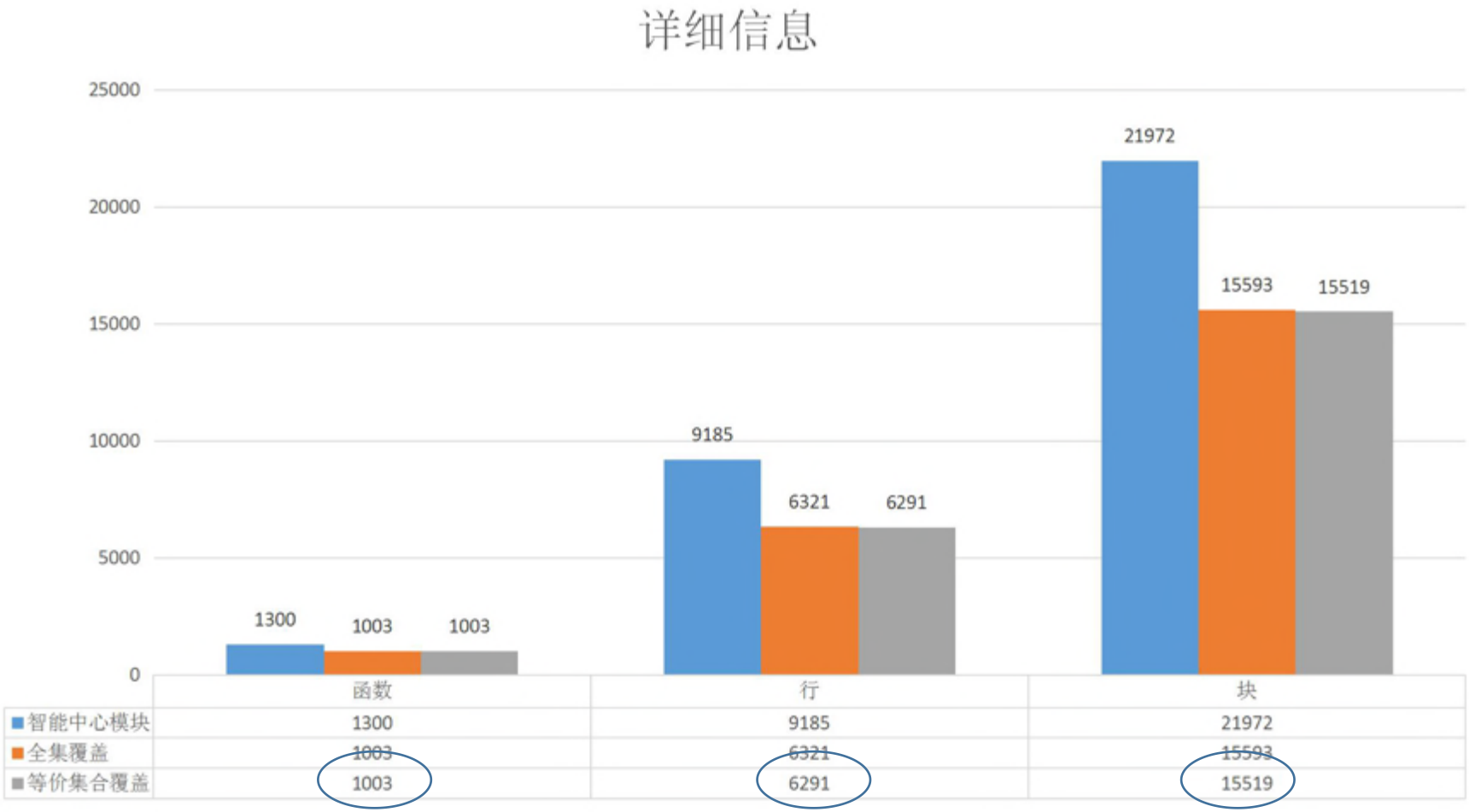
Line1 bits	Line2 bits	Intersect Result	Union Result
00	00	0	0
01	00	0	1
10	00	0	1
00	01	0	1
01	***01***	***1***	***1***
10	01	0	1
00	10	0	1
01	10	0	1
10	10	1	1

研究成果 - AutoCAD智能中心功能





研究成果 - AutoCAD智能中心功能



未来展望

- 测试用例，函数，行对源代码进行数据挖掘
- 将函数调用的堆栈结合到数据挖掘中

Q&A

AUTODESK®