18: **End**

### 4.3 实验及讨论

为了验证本节给出的利用蚁群算法基于解结构调整的覆盖表生成算法的有效性,本节给出了相关实验数据.

本文设计的整表演化算法是一种后优化算法.算法引入了随机后优化算法中“灵活位置”的概念,下面我们给出基于蚁群算法的整表演化算法和先用 AETG-MMAS 算法然后再利用随机后优化算法的性能比较,见表 20.其中,AETG-MMASp 表示对 AETG-MMAS 算法生成覆盖表使用随机后优化算法的方法.CA-AS 算法表示基于 AS 算法的整表生成算法.实验选取了 10 个覆盖表作为实验对象.从表中我们可以看出,除了第 10 个表 CA-AS 表现略差于 AETG-MMASp 外,CA-AS 算法所得结果的尺寸以及稳定性都要好于 AETG-MMASp.算法采用的是第 4 节给出的 AS 算法推荐参数配置,实验中覆盖表 CA9 与 CA10 的最大迭代次数为 10 000,而其余表的最大迭代次数为 1 000.

**Table 20** Comparison of AETG-MMAS,AETG-MMASp and CA-AS

**表 20** AETG-MMAS,AETG-MMASp 与 CA-AS 的比较

| CA1($N$;2;$3^4$) | | | | MCA2 ($N$;2;$5^1 3^8 2^2$) | | | |
|---|---|---|---|---|---|---|---|
| | AETG-MMAS | AETG-MMASp | CA-AS | | AETG-MMAS | AETG-MMASp | CA-AS |
| Min | 9 | 9 | 9 | Min | 21 | 19 | 17 |
| Max | 13 | 12 | 9 | Max | 25 | 22 | 19 |
| Avg | 10.366 67 | 9.466 667 | 9 | Avg | 23.4 | 20.366 67 | 18 |
| Dev | 0.808 717 | 0.973 204 | 0 | Dev | 0.894 427 | 0.808 717 | 0.454 859 |
| CA3($N$;2;$3^{13}$) | | | | MCA4($N$;2;$4^1 3^{39} 2^{35}$) | | | |
| | AETG-MMAS | AETG-MMASp | CA-AS | | AETG-MMAS | AETG-MMASp | CA-AS |
| Min | 21 | 16 | 16 | Min | 30 | 24 | 24 |
| Max | 23 | 19 | 17 | Max | 33 | 28 | 25 |
| Avg | 21.566 67 | 17.4 | 16.733 33 | Avg | 31.466 67 | 25.833 33 | 24.733 33 |
| Dev | 0.626 062 | 0.813 676 | 0.449 776 | Dev | 0.776 079 | 1.1768 85 | 0.449 776 |
| MCA5($N$;2;$4^{15} 3^{17} 2^{29}$) | | | | MCA6($N$;2;$6^{15} 4^6 3^8 2^3$) | | | |
| | AETG-MMAS | AETG-MMASp | CA-AS | | AETG-MMAS | AETG-MMASp | CA-AS |
| Min | 42 | 35 | 34 | Min | 38 | 34 | 33 |
| Max | 47 | 41 | 37 | Max | 44 | 41 | 36 |
| Avg | 43.866 67 | 38.466 67 | 35.333 33 | Avg | 40.4 | 37.333 33 | 34.266 67 |
| Dev | 1.279 368 | 1.479 36 | 0.606 478 | Dev | 1.631 585 | 1.397 864 | 0.739 68 |
| MCA7($N$;2;$7^1 6^{15} 5^1 4^5 3^8 2^3$) | | | | CA8($N$;2;$4^{10}$) | | | |
| | AETG-MMAS | AETG-MMASp | CA-AS | | AETG-MMAS | AETG-MMASp | CA-AS |
| Min | 48 | 45 | 42 | Min | 31 | 27 | 26 |
| Max | 54 | 53 | 47 | Max | 34 | 30 | 27 |
| Avg | 50.533 33 | 49.133 33 | 44.5 | Avg | 32.366 67 | 28.166 67 | 26.966 67 |
| Dev | 1.795 268 | 1.775 957 | 1.332 615 | Dev | 0.718 395 | 0.592 093 | 0.182 574 |
| CA9($N$;2;$4^{20}$) | | | | CA10(N;2;$6^{10}$) | | | |
| | AETG-MMAS | AETG-MMASp | CA-AS | | AETG-MMAS | AETG-MMASp | CA-AS |
| Min | 41 | 33 | 33 | Min | 60 | 58 | 60 |
| Max | 43 | 37 | 34 | Max | 65 | 62 | 62 |
| Avg | 42.166 67 | 34.4 | 33.033 33 | Avg | 62.3 | 59.633 33 | 60.933 33 |
| Dev | 0.746 64 | 0.968 468 | 0.182 574 | Dev | 1.055 364 | 1.066 2 | 0.520 83 |

表 21 给出了 CA-AS 和其他算法生成覆盖表最小尺寸的对比.其中,MMAS 表示 AETG-MMAS 算法,IPOs 是 IPO 算法的一种改进,SA 表示模拟退火算法[19].实验对象为表 20 中的 10 个表.根据结果可以发现,CA-AS 比 AETG-MMAS 性能有很大的提升,只在 MCA4 和 MCA5 两种输入时生成的尺寸比 IPOs 要大[20],但是当前最优结果都是由 SA 生成的,因此 CA-AS 算法仍然具有一定的改进空间.

**Table 21** Comparison of CA-AS and other algorithms

**表 21** CA-AS 和其他算法的比较

| CA | CA1 | MCA2 | CA3 | MCA4 | MCA5 | MCA6 | MCA7 | CA8 | CA9 | CA10 | $4^{30}$ | $4^{40}$ | $4^{50}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CA-AS | 9 | 17 | 16 | 24 | 34 | 33 | 42 | 26 | 33 | 60 | 37 | 40 | 42 |
| MMAS | 9 | 21 | 21 | 30 | 42 | 38 | 48 | 31 | 41 | 60 | – | – | – |
| IPOs | 9 | 17 | – | 23 | 32 | – | – | 28 | 34 | – | 38 | 41 | 43 |
| SA | 9 | 15 | 15 | 21 | 30 | 30 | 42 | – | – | – | – | – | – |
| Best | 9 | 15 | 15 | 21 | 30 | 30 | 42 | – | – | – | – | – | – |

蚁群算法本身非常耗时.在用蚁群算法进行整表演化实验时,这个问题尤为严重.我们对 CA($N$;2;$4^k$)($k$=10,20,30,40,50)这 5 个输入进行了实验.实验结果如图 8 所示.从图 8 中我们可以看到,随着 $k$ 的增长,CA-AS 算法的时间(s)会显著增加.当 $k$=50 时,算法一次运行的时间接近 2 小时.针对时间开销问题,本文余下部分对算法进行了并行化探索.

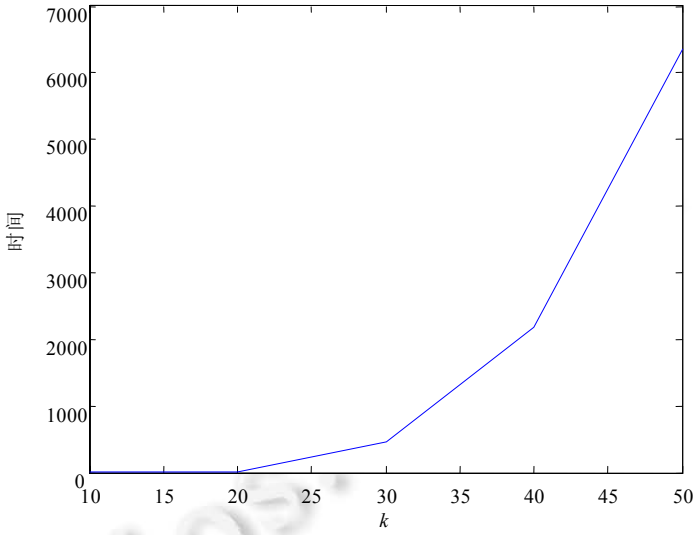



Fig.8 Time cost of CA-AS

图 8 CA-AS 算法的时间开销

## 5 利用蚁群算法生成覆盖表的并行化探索

### 5.1 并行化探索

由上述讨论我们可以看到,用蚁群算法生成覆盖表是一个非常耗时的工作.即便仅使用计算速度较快的 AETG-ACO,当覆盖表规模达到一定程度时,其计算时间可能仍然是难以承受的.所以,我们尝试从两方面来考虑将利用蚁群算法生成覆盖表的方法并行化.其一是将覆盖表本身的一些步骤并行化,其二是将蚁群算法中的某些步骤并行化.

在整表生成的过程中,需要计算灵活位置.当表的行数较多时,计算灵活位置会花费较大的计算代价.因此,我们提出并行化地寻找灵活位置的方法:将算法 2 的第 4 行~第 7 行以及计算所有 $t$ 维组合被覆盖次数的过程并行化处理.

蚁群算法本身是一种群智算法,演化学习过程主要是依靠每一代蚂蚁对环境的改变.在 AS 算法和 MMAS 算法中,每一代蚂蚁中的个体的行为都具有独立性,因此可以考虑将这两个变种的蚁群算法的蚂蚁迭代过程改为并行过程,从而节约计算时间.但是需要注意的是,每一代蚂蚁的迭代过程都依赖之前所有代蚂蚁的行为,约束性太强,所以不同代的蚂蚁不能并行化.另外,由于在 ACS 算法中,每次一只蚂蚁经过一条路径都会更新信息素,所以不方便将蚂蚁的演化过程并行化.对于 AS 和 MMAS,算法 1 中第 9 行~第 13 行也可以并行处理.

### 5.2 实验及讨论

上面我们分析了利用蚁群算法生成覆盖表的过程中的一些可并行的成分,下面我们以并行化的蚂蚁为例设计实验.选取如表 22 所示的 6 个二维覆盖表来进行实验.对于每一个表,分别设置并行的线程数为 1~8,每一种线程设置都重复进行 10 次实验.

**Table 22** Experiment objects of parallel experiments

**表 22** 并行化实验对象

| | | |
|---|---|---|
| CA($N$;2;$3^{13}$) | CA($N$;2;$4^{10}$) | CA($N$;2;$6^{10}$) |
| MCA($N$;2;$5^1$,$3^8$,$2^2$) | MCA($N$;2;$6^1$;$5^1$;$4^6$;$3^8$;$2^3$) | MCA($N$;2;$7^1$;$6^1$;$5^1$;$4^5$;$3^8$;$2^3$) |

我们首先在 hadoop 集群上进行算法并行化的实验,但是 hadoop 本身不适合完成计算密集型的任务[21].在 hadoop 集群上得到的实验结果还不如单机的实验结果,于是我们放弃了实验 hadoop 进行并行化实验的想法.由于现在的计算机往往都是多核的,CPU 使用率很低,于是我们采取另外一种并行化策略,即利用 Java 的多线程机制[22].我们使用了 Java 版本的 OpenMP,将蚂蚁寻找解的过程并行化.实验环境为 Java1.7,Windows8.1 64 位系统,Intel(R) Core(TM)i7-3840QM CPU(4 核 8 处理器 2.8GHz)内存 16GB.算法的参数我们固定为如表 23 所示.

**Table 23** Configuration of parallel AETG-AS

**表 23** 并行化 AETG-AS 算法的配置参数

| $Nc$ | $m$ | $\rho$ | $\tau_{init}$ | $Q$ | $\alpha$ | $\beta$ |
|---|---|---|---|---|---|---|
| 200 | 100 | 0.6 | 0.4 | 0.01 | 0.7 | 0.1 |

最后我们得到的实验结果见表 24,将 3 个水平覆盖表和 3 个混合覆盖表从左到右分别记为 CA1~CA3, MCA1~MCA3.表中记录的数据是不同的表在不同线程数下分别运行 10 次所需要的平均时间.

**Table 24** Run time of parallel AETG-AS when thread number varies

**表 24** 并行化 AETG-AS 算法不同线程数的运行时间

| 线程数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| CA1 | 8.86 | 4.87 | 4.17 | 3.47 | 3.16 | 3.15 | 3.10 | 3.53 |
| CA2 | 6.29 | 3.71 | 3.12 | 2.50 | 2.24 | 2.15 | 2.17 | 2.37 |
| CA3 | 14.08 | 8.21 | 6.77 | 5.55 | 5.02 | 4.80 | 4.74 | 5.36 |
| MCA1 | 6.06 | 3.54 | 2.96 | 2.44 | 2.21 | 2.11 | 2.05 | 2.35 |
| MCA2 | 58.98 | 32.68 | 27.94 | 24.81 | 23.93 | 24.86 | 23.94 | 27.45 |
| MCA3 | 79.20 | 43.77 | 35.98 | 30.69 | 30.78 | 31.15 | 31.00 | 33.84 |

为了表现得更加直观,我们将实验结果以折线图的形式表现出来,如图 9 所示.

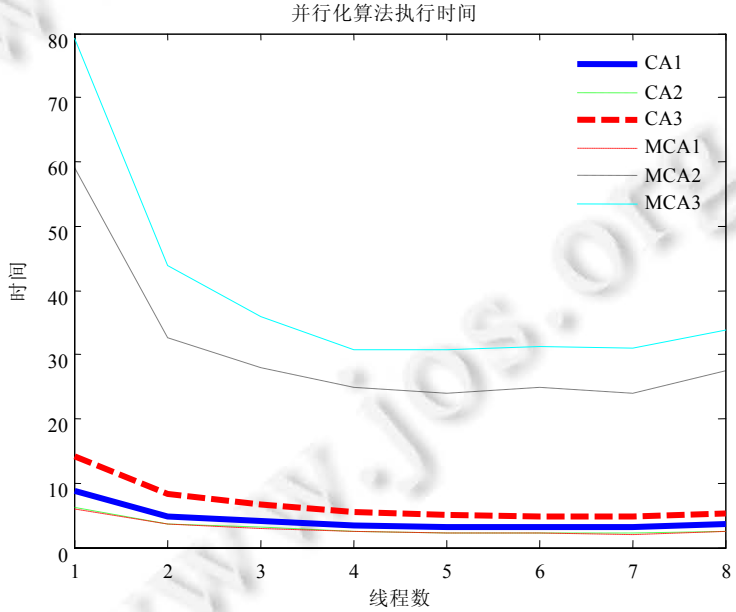



Fig.9 Result of parallel experiments

图 9 并行实验的结果

根据结果我们可以看出,并行化后算法的运行速度更快,但是运行时间不是随着线程数的增加而一直变短.

当线程个数接近处理器的个数时,算法运行的时间反而变长.在实际的运行过程中,我们要根据自己所拥有的计算资源合理设置并行的尺度.

## 6 总 结

本文主要对利用蚁群算法生成覆盖表的性能进行深入的探索与挖掘.蚁群算法作为一种常见的演化搜索算法,成功地解决了一系列组合优化问题.而在求解覆盖表生成问题的过程中,虽然已有的研究工作说明了蚁群算法的可用性,但却缺乏对其实际性能的深入探讨.本文在已有工作的基础上,结合对蚁群算法、覆盖表生成算法的相关研究成果,研究了利用蚁群算法生成覆盖表的算法变种、算法参数、解结构调整以及并行化,充分挖掘了蚁群算法生成覆盖表的潜力.主要包括以下几个方面的工作:

1) 对已有的蚁群算法一次一条测试用例生成覆盖表的算法进行了深入挖掘,研究了算法的变种以及参数的配置.研究算法变种时,我们提出利用蚁群算法生成覆盖表的框架,选择了 3 种算法变种 AS,ACS,MMAS 来进行覆盖表的生成,并得到相应的 AETG-AS,AETG-ACS,AETG-MMAS 算法;
2) 进行了单参数配置的研究,然后研究了二维参数覆盖表,最后为每种算法变种提供了一种推荐配置.在推荐配置下,将 3 种覆盖表生成算法的结果进行比较,发现效果最好的是 AETG-MMAS 算法;
3) 对蚁群算法生成覆盖表的演化目标进行了调整,从已有的演化一条测试用例调整到演化一个整表,提出了在灵活位置上进行演化的 CA-AS 算法.该算法能够克服一次一条测试用例生成方法的局限性,得到更好的结果.实验结果表明,CA-AS 的实验效果要比 AETG-MMAS 的实验效果好得多,非常接近已知最优解;
4) 将蚁群算法生成覆盖表的算法中的一些过程(包括灵活位置求解,每一代蚂蚁的演化)进行并行化处理,解决了在表的规模较大时非常耗时的问题.在 hadoop 平台上的实验失败后,使用 Java 版本的 OpenMP,将蚂蚁寻找解的过程并行化,最后有效地缩短了算法的运行时间.

虽然经过改进的结果未能超越最优解,但是这项工作仍然是有意义的.从蚁群算法的角度来看,该算法作为一种元启发式算法,能够经过小的改动而应用在不同的优化问题中.但是,不同的问题具有不同的特点,所以对特定的问题,对蚁群算法的设计在其性能方面会产生很大的影响.而本文研究了蚁群算法在覆盖表生成问题上的应用,针对这个问题对蚁群算法进行设计,使得在这个问题上的潜力充分发挥出来.从覆盖表生成的角度来看,生成覆盖表有很多种方法,本文对蚁群算法进行了评估,展示了利用蚁群算法来生成覆盖表的优劣.

尽管以上研究探索很好地挖掘了蚁群算法生成覆盖表的潜力,但仍然存在很多不足.例如对算法变种选择而言,蚁群算法的变种有很多,我们却只选择了其中的 3 种算法进行研究.在参数调优的过程中,一方面,参数的离散程度不够,覆盖强度也不够;另一方面,我们选择的实验对象比较少,实验次数也不充足.整表演化时,我们采取逐渐减小覆盖表尺寸的方法,没有对逐渐增加测试用例条数的方法做过多的讨论.此外,表尺寸减小的方式、演化位置的选择也存在多种策略,我们没有进行过多的讨论.在缩短算法时间方面,我们只是让能够并行执行的步骤并行化执行,没有更加深入地设计更有效的分布式算法,也没有成功地利用分布式平台.针对上述不足,将来会展开,作进一步的研究,让利用蚁群算法生成覆盖表的方法能够发挥出更大的潜力.

**References**:

[1] Nie CH. Concepts and Methods of Software Testing. Beijing: Tsinghua University Press, 2013. 1−22 (in Chinese).
[2] Nie CH. Combinatorial Testing. Beijing: Beijing Science Press, 2015. 1−119 (in Chinese).
[3] McCaffrey JD. Generation of pairwise test sets using a genetic algorithm. In: Proc. of the Int'l Conf. on Computer Software and Applications (COMPSAC). 2009. 626−631. [doi: 10.1109/COMPSAC.2009.91]
[4] Nie CH, Leung H. A survey of combinatorial testing. ACM Computing Surveys (CSUR), 2011,43(2):11. [doi: 10.1145/1883612.1883618]
[5] Yu L, Tai KC. In-Parameter-Order: A test generation strategy for pairwise testing. In: Proc. of the High-Assurance Systems Engineering Symp. 1998. 254−261. [doi: 10.1109/HASE.1998.731623]

[6] Bryce RC, Colbourn CJ. The density algorithm for pairwise interaction testing. SoftwareTesting, Verification and Reliability, 2007,17(3):159−182. [doi: 10.1002/stvr.365]

[7] Dorigo M, Maniezzo V, Colorni A. Ant system: Optimization by a colony of cooperating agents. IEEE Trans. on Systems, Man, and Cybernetics (Part B Cybernetics), 1996,26(1):29−41. [doi: 10.1109/3477.484436]

[8] Dorigo M, Stützle T. Ant Colony Optimization. Cambridge: MIT Press, 2004. 1−151.

[9] Cohen DM, Dalal SR, Fredman ML, Patton GC. The AETG system: An approach to testing based on combinatorial design. IEEE Trans. on Software Engineering (TSE), 1997,23(7):437−444. [doi: 10.1109/32.605761]

[10] Shiba T, Tsuchiya T, Kikuno T. Using artificial life techniques to generate test cases for combinatorial testing. In: Proc. of the Computer Software and Applications Conf. 2004. 72−77. [doi: 10.1109/CMPSAC.2004.1342808]

[11] Chen X, Gu Q, Li A, Chen DX. Variable strength interaction testing with an ant colony system approach. In: Proc. of the Asia-Pacific Software Engineering Conf. (ASPEC). 2009. 160−167. [doi: 10.1109/APSEC.2009.18]

[12] Chen X, Gu X, Zhang X, Chen DX. Building prioritized pairwise interaction test suites with ant colony optimization. In: Proc. of the Int'l Conf. on Quality Software (QSIC). 2009. 347−352. [doi: 10.1109/QSIC.2009.52]

[13] Nie CH, Wu HY, Liang YL, Leung H, Kuo FC, Li Z. Search based combinatorial testing. In: Proc. of the Asia-Pacific Software Engineering Conf. (ASPEC). 2012. 778−783. [doi: 10.1109/APSEC.2012.16]

[14] Dorigo M, Gambardella LM. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Trans. on Evolutionary Computation, 1997,1(1):1−24. [doi: 10.1109/TEVC.1997.585887]

[15] Dorigo M, Blum C. Ant colony optimization theory: A survey. Theoretical Computer Science, 2005,344(2-3):243−278. [doi: 10.1016/j.tcs.2005.05.020]

[16] Liang YL, Nie CH. The optimization of configurable genetic algorithm for covering arrays generation. Chinese Journal of Computers, 2012,35(7):1522−1538 (in Chinese with English abstract).

[17] Bryce RC, Colbourn CJ. One-Test-at-a-Time heuristic search for interaction test suites. In: Proc. of the 9th Annual Conf. on Genetic and Evolutionary Computation (GECCO). 2007. 1082−1089. [doi: 10.1145/1276958.1277173]

[18] Nayeri P, Colbourn CJ, Konjevod G. Randomized post-optimization of covering arrays. European Journal of Combinatorics, 2013,34(1):91−103. [doi: 10.1016/j.ejc.2012.07.017]

[19] Torres-Jimenez J, Rodriguez-Tello E. New bounds for binary covering arrays using simulated annealing. Information Sciences, 2012,185(1):137−152. [doi: 10.1016/j.ins.2011.09.020]

[20] Calvagna A, Gargantini A. IPO-s: Incremental generation of combinatorial interaction test data based on symmetries of covering arrays. In: Proc. of the IEEE Int'l Conf. on Software Testing Verification and Validation Workshops. 2009. [doi: 10.1109/ICSTW.2009.7]

[21] White T. Hadoop: The Definitive Guide. 4th ed., Sebastopol: O'Reilly Media, 2015. 3−97.

[22] Eckel B. Thinking in Java. 4th ed., Upper Saddle River: Prentice Hall, 2006. 797−822.

**附中文参考文献**:

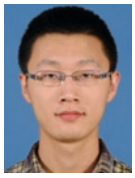
[1] 聂长海.软件测试的概念与方法.北京:清华大学出版社,2013.1−22.

[2] 聂长海.组合测试.北京:科学出版社,2015.1−119.

[16] 梁亚澜,聂长海.覆盖表生成的遗传算法配置参数优化.计算机学报,2012,35(7):1522−1538.

**曾梦凡**(1992－),男,湖北仙桃人,学士,主要研究领域为软件测试.

**张文茜**(1994－),女,学士,主要研究领域为软件测试.



**陈思洋**(1989－),男,硕士,CCF 学生会员,主要研究领域为软件测试.

**聂长海**(1971－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试.