相似索引:适用于重复数据删除的二级索引*

张志珂，蒋泽军，蔡小斌，彭成章
(西北工业大学 计算机学院，西安 710072)

摘　要：由于 EB(extreme binning)使用文件的最小块签名作为文件的特征,它不适合处理主要包括小文件的数据负载,会导致较差的重复数据删除率。为了改进 EB,提出了相似索引。它把相似哈希作为文件的特征,是一种适用于以小文件为主的数据负载的重复数据删除的二级索引。实验结果表明,相似索引的重复数据删除率比 EB 高 24.8%;相似索引的内存使用量仅仅是 EB 的 0.265%。与 EB 相比,相似索引需要更少的存储使用量和内存使用量。

关键词：重复数据删除；相似哈希；相似索引；块查找磁盘瓶颈问题；二级索引
中图分类号：TP301.6　　**文献标志码**：A　　**文章编号**：1001-3695(2013)12-3614-04
doi:10.3969/j.issn.1001-3695.2013.12.025

Similar index: two-level index used for deduplication

ZHANG Zhi-ke, JIANG Ze-jun, CAI Xiao-bin, PENG Cheng-zhang
(*School of Computer, Northwestern Polytechnical University, Xi'an* 710072, *China*)

Abstract: However, since EB (extreme binning) utilized the minimum chunk ID of a file as the representative chunk signature, EB was not suitable for backup data stream mainly containing small files. To improve EB, this paper proposed simi index using simi hash as the feature of a file. It was a novel two-level index suitable for workload mainly consisting of small files. Experiment results show that, the deduplication efficiency of simi index is 24.8% better than EB, and the RAM usage of simiIndex only 0.265% of that of EB. Compared with EB, simi index needs less storage and less RAM.

Key words: deduplication; simi hash; similar index; chunk-lookup disk bottleneck problem; two-level index

0　引言

块查找磁盘瓶颈问题是基于块的重复数据删除最重要的问题之一。为了快速地找到重复的数据块,必须把块索引完全放到内存中。当系统的数据量比较小时,可以把整个块索引全部放到内存中。当平均数据块尺寸是 4 KB,对于 100 TB 数据量(实际存储的数据量),Jumbo store[1] 的整个块索引大约需要 1 500 GB 内存。当数据量的规模达到 PB 级时,基于块的重复数据删除的块索引会在 TB 级。把一个如此大的索引全部放在内存中是很困难的,必须把一部分索引放在磁盘上,在需要的时候,再从磁盘读出来。那么,每次在块索引中查找一个数据块,就可能需要从磁盘读入一部分索引到内存中,这会导致极其低的吞吐量。目前有几种研究成果可以缓解块查找磁盘瓶颈问题,包括 Bloom filter[2]、Sparse indexing[3] 和 EB[4]、基于相似的重复数据删除[5~8] 以及一些重复数据删除集群[9~11]。其中最著名的是 EB。EB 选择文件的所有块 ID 中最小的块 ID 作为块 ID 代表,并把它放在内存中的索引里,把具有相等最小块 ID 的文件的所有块 ID 放到磁盘上的同一个箱子里。

上述方法可以大量地减少内存使用量。但是有些备份数据主要包含小文件,如 NAS 备份数据、连续数据保护的备份数据以及电子邮件备份。新电子邮件需要在存储的同时就备份。当文件很小时,文件的最小块 ID 很容易发生改变。那么,EB 使用的块 ID 代表也以高概率发生改变。在这种情况下,EB 根据最小块 ID 可能无法找到相似文件和重复的数据块。例如,在图 1 中,文件 M 中具有最小的块 ID 的数据块有一个字节 A 变成了 X,产生了文件 N。先对文件 M 进行重复数据删除。然后当重复数据删除文件 N 时,块 ID 代表变成了 2,所以 EB 无法找到文件 M。然而,除了数据块 1,文件 N 与文件 M 的数据块都是相同的。

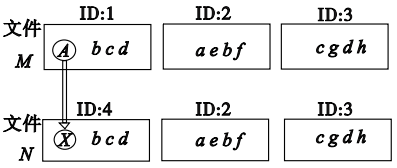

图1　问题描述图

为了解决这个问题,本文提出相似索引。它是基于相似哈希[12] 的一种新颖的二级索引。相似哈希最重要的特性是,两个文件越相似,它们的相似哈希值也越相似。相似哈希值是根据文件的所有内容来计算的,而不是基于最小块 ID,因此更适合处理小文件。

用一个真实世界的数据集评估了相似索引。实验结果表明,相似索引的重复数据删除率比 EB 高 24.8%;它的内存使用量仅仅是 EB 的 0.265%。

收稿日期：2013-04-16；**修回日期**：2013-05-23　　**基金项目**：陕西省自然科学基金资助项目(2010JM8023);航空科学基金资助项目(2010ZD53042)

作者简介:张志珂(1983-),男,河南平顶山人,博士,主要研究方向为储存系统、重复数据删除(zhikezhang1983@gmail.com);蒋泽军(1964-),男,陕西西安人,教授,硕士,主要研究方向为储存系统、网络安全、重复数据删除;蔡小斌(1957-),男,陕西西安人,教授,硕士,主要研究方向为储存系统、测控系统、嵌入式系统;彭成章(1978-),男,河南焦作人,博士研究生,主要研究方向为云储存.

1　设计

图 2 展示了使用相似索引的重复数据删除。当文件流进入重复数据删除系统时,系统逐个处理文件流中的文件。分块器负责对文件分块。大部分分块算法使用 Rabin 哈希算法[13]。本文使用 TTTD 分块算法[14]对文件分块。设置平均块长度为 4 KB。分块器对文件分块后,计算每个数据块的 SHA-1 哈希值,把它作为数据块的 ID。分块器也计算整个文件的 SHA-1 哈希,作为文件的签名。

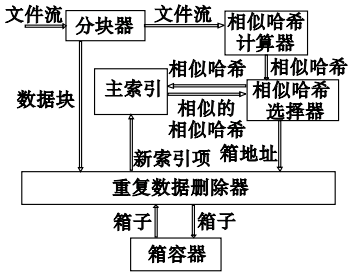

图2　基于相似哈希的重复数据删除

相似索引的第一级索引是主索引,放置在内存中。它是一个〈关键字,值〉映射表。主索引的关键字是相似哈希。与关键字对应的值包括标签数量、箱子地址、文件签名(整个文件的 SHA-1 哈希值)。相似索引的第二级索引是箱子,放置在磁盘上。箱容器在磁盘上存储箱子,箱子包含许多块 ID,这些块 ID 是类似文件的块签名。相似哈希计算器负责计算每个文件的相似哈希。它先对每个标签计数,再计算所有标签量的总和得到相似哈希值。相似哈希选择器先在主索引中搜索相似范围内的一些相似哈希值,然后通过计算相似哈希距离来确定唯一的最类似的相似哈希(具体在 1.1 节)。然后,相似哈希选择器把最类似的相似哈希所在的箱子地址发送给重复数据删除器。

根据相似哈希计算器发送的最类似的相似哈希的箱子地址,重复数据删除器读入这个箱子。它在这个箱子中查找目标文件的所有块 ID,找到重复的块 ID,并把新的块 ID 存入这个箱子。如果没有新的块 ID,就没必要更新箱子。如果箱子地址为空,则意味着与目标相似哈希类似的相似哈希不存在。在这种情况下,重复数据删除器创建一个新的箱子。同时,它在主索引中为目标文件插入一个新的索引项。

1.1　用相似哈希选择最相似的文件

相似哈希函数[12]与加密哈希函数(SHA-1 或 MD5)是完全不同的两类哈希函数。加密哈希函数的目标是使哈希冲突最小化。两个非常相似的数据的加密哈希值是显著不同的,而相似哈希函数的目标与加密哈希函数的目标是完全相反的,即非常相似的数据的相似哈希值是非常接近的,甚至是相同的。这也是相似哈希的重要特征。不同数据的相似哈希值的距离是不同数据的内容差异的一种度量方式。相似哈希的主要功能是相似检测。

相似哈希[12]的核心思想是计算文件中每个标签的数量,然后通过计算所有标签的数量的总和得到相似哈希值。标签是一个字符。随机选择 16 个标签。用 8 Byte 来存储相似哈希值,用 2 Byte 来存储一个标签数量。

仅仅使用相似哈希不能够完全确定两个文件是否相似。为了快速地计算相似哈希的距离和测量文件的相似程度,需要把标签数量也放在主索引中。对于相似区间内的相似哈希,具有最近的相似哈希距离的相似哈希所对应的文件是最相似的文件。

通过式(1)来计算相似区间。T 是相似区间,h 是相似哈希,s 是文件的尺寸,n 是预选择的标签数量,t 是相似容忍度。相似容忍度是一个相似标准($0 \leqslant t \leqslant 1$),它用于决定两个文件是否相似。例如,$t=0.1$ 的含义是:如果两个文件有少于 10% 的区别,这两个文件被认为是相似的,它会影响相似文件搜索的结果。相似容忍度越大,就可以搜索到越多的相似文件。因此,t 可以影响找到的重复数据块的数量。

$$T \in [h - \frac{s}{n} \times t, h + \frac{s}{n} \times t] \tag{1}$$

通过式(2)计算相似哈希距离。n 是预选择的标签数量,$t_{i,k}$是文件 i 中第 k 个标签的数量。

$$d_{i,j} = \sum_{k=0}^{n} |t_{i,k} - t_{j,k}| \tag{2}$$

搜索目标文件的相似文件需要两步操作:a)在主索引搜索相似区间内的相似哈希;b)计算目标文件的相似哈希与相似区间内的每个相似哈希的距离,选择具有最近的相似哈希距离的文件作为最相似的文件。

以图 1 为例计算相似哈希距离。令 $n=4$,选择四个标签 (A,b,c,d),$t=0.4$。那么,可以得到 $s=12$,文件 M 的每个标签的数量分别为(1,2,2,2),文件 M 的相似哈希值是 7。文件 N 的每个标签的数量分别为(0,2,2,2),相似哈希值是 6。那么,文件 N 的相似区间 $T_N=[4.8,7.2]$。当重复数据删除文件 N 时,在主索引中搜索相似区间 T_N 内的相似哈希,就可以找到文件 M。由于文件 N 与文件 M 的最小块 ID 不同,当重复数据删除文件 N 时,EB 不能找到文件 M。然后,计算文件 M 与文件 N 的相似距离为 1。

1.2　相似索引

相似索引是基于相似哈希的二级索引。与 EB 相比,它可以改进重复数据删除率,并降低内存使用量。

EB 为每个箱子的第一个文件在主索引中创建一个索引项,选择这个文件的最小块 ID 作为这个文件和箱子的块 ID 代表。当文件的最小块 ID 不改变时,EB 就可以找到这个文件的后续新版本中重复的数据块。EB 不是根据文件的所有内容来计算块 ID 代表。所以,当具有最小块 ID 的数据块发生改变时(EB 假设文件的最小块 ID 改变的概率很低),EB 不是很有效。

用相似哈希创建主索引。与 EB 不同的是,相似哈希是根据文件的所有内容计算的,而不仅仅是最小块 ID。当文件的最小块 ID 发生变化时,它不会显著地改变文件的相似哈希。所以,相似索引能够找到具有不同的最小块 ID 的相似文件,从而找到重复的数据块,而 EB 则不能。相似索引的第一级索引(主索引)放在内存中,相似索引的第二级索引由许多箱子组成,放置在磁盘上。每个箱子在主索引中只有一个索引项。

表 1 给出了一个主索引的索引项。它是来自实验的真实数据。在表 1 中,相似哈希是索引项的关键字,其他的项是关键字对应的值。相似哈希是一个 64 位的整数,用于找到一些相似的文件。标签数量表示每个标签在文件中出现的次数。选择 16 个标签。标签数量用于确定唯一的最相似文件。文件签名链是箱子中保存的文件签名。在找到最相似的文件后,搜索文件签名链来判断目标文件是否与箱子中的某个文件是完全重复的。如果目标文件是重复的,就没有必要再从磁盘读入箱子,否则就必须从磁盘读入箱子。箱子的地址是目标文件的最相似文件的主索引索引项中

的箱子地址。在实验中,只在每个索引项中保存一个文件签名,就是第一个被存入箱子的文件签名。

表1 主索引的索引项示例

相似哈希	25356
标签数量	207 315 167 51 42 2224 9182 15 436 954 664 310 5106 644 1803 3236
文件签名链	aae76b04831db7c2281f0e0bafc6336cc7b58e01 46eed627d8dda7326e3d634b2c95d52a7ff77d2b
地址	1

在计算文件的相似哈希之后,相似哈希选择器从主索引中查找所有在相似区间内的相似哈希;然后相似哈希选择器计算目标文件的相似哈希与相似区间内的每个相似哈希的距离,选择具有最近距离的相似哈希所对应的文件为最相似文件;最后重复数据删除器导入这个最相似的文件所在的箱子,并根据它对目标文件进行重复数据删除。

1.3 相似索引与EB的计算复杂度比较

相似索引与EB的核心功能是找到文件中的重复数据块。EB先查找并更新内存中的主索引,找到包含相似文件的箱子,再从箱子里查找重复的数据块并更新箱子。假设EB存储了n个文件,每个文件平均包括n_1个数据块(n_1是常数且$n_1 << n$),每个箱子平均包括n_2个数据块($n_2 << n$),用B树保存主索引和箱子的索引。EB主要步骤的时间复杂度分别是:检索和更新内存中的主索引($O(\log_2 n)$),检索和更新磁盘上的箱子($n_1 \times O(\log_2 n_2)$)。那么EB的时间复杂度是$O(\log_2 n) + n_1 \times O(\log_2 n_2)$。因为$n_1$是常数并且$n_2 << n$,所以EB的时间复杂度是$O(\log_2 n)$。

相似索引与EB的主要区别是需要额外找到相似区间内的相似哈希并确定最相似的文件。假设每个文件的相似区间平均包括n_3个相似哈希(n_3是常数且$n_3 << n$)。相似索引的主要步骤的时间复杂度分别是:检索和更新内存中的主索引($O(\log_2 n)$),查找相似区间内的相似哈希(n_3),计算最相似的文件(n_3),检索和更新磁盘上的箱子($n_1 \times O(\log_2 n_2)$)。那么,相似索引的时间复杂度是:$O(\log_2 n) + 2 \times n_3 + n_1 \times O(\log_2 n_2)$,即$O(\log_2 n)$。相似索引与EB具有相同的时间复杂度$O(\log_2 n)$。

相似索引和EB的主要内存空间使用量是主索引。它们都在主索引中为每个箱子维护一个索引项。EB的每个索引项需要包含文件最小块ID(20 Byte),一个文件签名(整个文件的哈希,20 Byte)和一个箱子地址(10 Byte),共50 Byte。假设EB存储了n个文件,平均每个箱子有f_1个文件(f_1是常数)。那么,EB的空间复杂度是$O(50 \times n/f_1)$,即$O(n)$。

相似索引的每个索引项包含一个相似哈希(8 Byte)、16个标签数量(32 Byte)、一个文件签名(20 Byte)和一个箱子地址(10 Byte),共70 Byte。每个标签数量需要用2 Byte来存储。假设相似索引存储了n个文件,平均每个箱子有f_2个文件(f_2是常数)。那么,相似索引的空间复杂度是$O(70 \times n/f_2)$,即$O(n)$。因此,相似索引与EB具有相同的空间复杂度$O(n)$。

2 实验结果

本章评估相似索引的重复数据删除率和内存使用量。本文实现了相似索引,收集了一个真实世界的数据集,用它来比较相似索引和EB。这个数据集是Linux源代码档案,叫Linux。它包含Linux1.2.0至Linux2.5.75的源代码版本,一共564个版本。Linux中的大多数文件是小文件,一般是几十KB。Linux代表了主要由小文件组成的高冗余数据负载。Linux数据集包含3 186 361个文件,共35.678 GB。

2.1 重复数据删除率的比较

重复数据删除率是原始数据量除以实际存储的数据量,如式(3)所示[2]。重复数据删除率可以衡量存储空间使用量,重复数据删除率越大越好,说明存储空间使用量越少。

$$\text{重复数据删除率} = \frac{\text{原始数据量}}{\text{存储使用量}} \tag{3}$$

图3比较了相似索引、EB和传统重复数据删除的重复数据删除率,表明相似索引的重复数据删除率明显优于EB。传统重复数据删除[1]在内存中维护所有已存储数据块的索引,因此可以找到所有的重复数据块,具有最高的重复数据删除率。但是,它需要非常多的内存,在实际应用中是不可行的[2]。对于Linux数据集,传统重复数据删除算法的重复数据删除率是21.38。当$t \geq 0.02$时,相似索引的重复数据删除率比EB好。另外,随着t的增长,相似索引的重复数据删除率也随之增长。当$t = 0.2$时,相似索引的重复数据删除率接近传统重复数据删除;当$t \geq 0.1$时,相似索引的重复数据删除率不再有明显的提高。此时,相似索引的重复数据删除率比EB高24.8%。t取较小的值可以使箱子的尺寸较小。箱子的平均尺寸较小对性能有好处,可以减少从磁盘读取的数据量。

2.2 索引内存使用量的比较

对于Linux数据集,传统重复数据删除的块索引的内存使用量是11.625 MB,EB索引的内存使用量是6.397 MB。如图4所示,相似索引的内存使用量比EB少很多。当$t \geq 0.05$时,它的内存使用量小于EB的内存使用量的0.5%。此外,相似哈希的内存使用量随着t的增加而降低。当$t > 0.1$时,相似哈希的内存使用量不再有明显的降低。此时,它只占EB的内存使用量的0.265%。

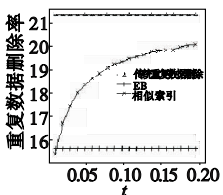

图3 重复数据删除率的比较(Linux)

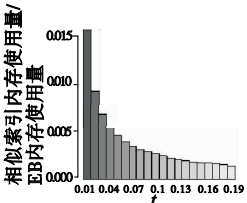

图4 内存使用量的比较(Linux)

基于块的重复数据删除的吞吐量主要取决于I/O访问量。对于非常大的数据量,传统重复数据删除处理文件的每个一个数据块都需要访问一次磁盘,导致极低的吞吐量。相比之下,EB处理一个文件只需要访问磁盘一次,具有高吞吐量[4]。相似索引与EB一样,处理一个文件只需要访问磁盘一次,与EB有类似的吞吐量,本文不再详述。

2.3 索引数据存储空间的比较

除了存储在内存中的第一级索引,相似索引和EB都需要一些永久磁盘存储空间来存储其第二级索引,也就是箱子。它们不需要临时索引数据存储空间。箱子包含文件的数据块的信息被存储在磁盘上。相似索引和EB都是通过内存中的索引快速找到包含相似文件的数据块的信息的箱子,再从箱子里找到重复的数据块。对于每一个文件,相似索引和EB只需要找到一个箱子并把文件的数据块的信息存到这个箱子里。相似索引与EB使用不同的方法确定目标箱子,但是,它们存储

的内容(文件的数据块信息)是相同的。因此,相似索引和 EB 需要的索引数据存储空间是相同的。

3　结束语

块查找磁盘瓶颈问题是基于块的重复数据删除的关键问题。本文提出了相似索引来改进 extreme binning 的重复数据删除率,并且降低内存使用量。相似索引是基于相似哈希的二级索引,适用于以小文件为主的数据负载的重复数据删除。实验结果表明,当相似容忍度是 0.1 时,相似索引的重复数据删除率比 EB 高 24.8%,并且其内存使用量仅仅是 EB 的 0.265%。相似索引的存储使用量和内存使用量都比 EB 少。

参考文献:

[1] ESHGHI K,LILLIBRIDGE M,WILCOCK L,*et al*. Jumbo store: providing efficient incremental upload and versioning for a utility rendering service[C]//Proc of the 5th USENIX Conference on File and Storage Technologies. Berkeley: USENIX,2007:123-138.

[2] ZHU B,LI Kai,PATTERSON H. Avoiding the disk bottleneck in the data domain deduplication file system[C]// Proc of the 6th USENIX Conference on File and Storage Technologies. Berkeley: USENIX, 2008:269-282.

[3] LILLIBRIDGE M,ESHGHI K,BHAGWAT D,*et al*. Sparse indexing: large scale, inline deduplication using sampling and locality[C]// Proc of the 7th Conference on File and Storage Technologies. Berkeley:USENIX,2009:111-123.

[4] BHAGWAT D,ESHGHI K,LONG D,*et al*. Extreme binning: scalable,parallel deduplication for chunk-based file backup[C]// Proc of IEEE International Symposium on Modeling,Analysis & Simulation of Computer and Telecommunication Systems. Washington DC: IEEE Computer Society, 2009:1-9.

[5] ARONOVICH L,ASHER R,BACHMAT E,*et al*. The design of a similarity based deduplication system[C]// Proc of SYSTOR : The Israeli Experimental Systems Conference. New York: ACM Press, 2009: 6.

[6] ROMANSKI B,HELDT L,KILIAN W,*et al*. Anchor-driven subchunk deduplication[C]// Proc of SYSTOR 2011: The Israeli Experimental Systems Conference. New York: ACM Press,2011: 16.

[7] ZHANG Zhi-ke,BHAGWAT D,LITWIN W,*et al*. Improved deduplication through parallel binning[C]// Proc of the 31st IEEE International Performance Computing and Communications Conference. Washington DC: IEEE Compurter Society,2012: 130-141.

[8] ZHANG Zhi-ke,JIANG Ze-jun,LIU Zhi-qiang,*et al*. LHs: a novel method of information retrieval avoiding an index using linear hashing with key groups in deduplication[C]// Proc of International Conference on Machine Learning and Cybernetics. Washington DC: IEEE Compurter Society,2012: 1312-1318.

[9] DUBNICKI C,GRYZ L,HELDT L,*et al*. Hydrastor: a scalable secondary storage[C]// Proc of the 7th Conference on File and Storage Technologies. Berkeley:USENIX,2009:97-210.

[10] UNGUREANU C,ATKIN B,ARANYA A,*et al*. Hydrafs: a high-throughput file system for the hydrastor content-addressable storage system[C]// Proc of the 8th USENIX Conference on File and Storage Technologies. Berkeley: USENIX,2010: 225-238.

[11] DONG Wei,DOUGLIS F,LI Kai,*et al*. Tradeoffs in scalable data routing for deduplication clusters[C]// Proc of the 9th USENIX Conference on File and Storage Technologies. Berkeley: USENIX,2011: 15-29.

[12] SADOWSKI C,LEVIN G. Simihash: hash-based similarity detection [R]. Santa Cruz: University of California at Santa Cruz,2011.

[13] RABIN M. Fingerprinting by random polynomials[R]. Cambridge: Harvard University,1981.

[14] FORMAN G,ESHGHI K,CHIOCCHETTI S. Finding similar files in large document repositories[C]// Proc of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. New York: ACM Press,2005: 394-400.

(上接第 3602 页)机制的驱动下,MFA 更容易跳出进化停滞状态。当发生进化变慢或者停滞时,MFA 能通过子群之间的信息传递,牵引进化受阻的萤火虫重新进入进化状态,具备一定的进化复苏能力,从而能够更好地满足局部搜索和全局精度的全面寻优要求,具有更好的求解效率。

4　结束语

本文对常规萤火虫算法进行了分析,并提出了一种基于多种群学习机制的萤火虫算法(MFA),通过设置不同模式参数的萤火虫子群,增加寻优萤火虫个体的多样性。在子群之间构建学习机制,帮助进化停滞的子群重新进入进化过程,当种群最优个体也无法提供更好的寻优信息时,利用高斯变异激发自身的再进化能力。利用多个标准测试函数对 MFA 进行了性能对比测试,从实验结果可以看出,MFA 能有效改进 FA 存在的进化停滞问题,并在全局精度和收敛速度等方面都具有较好的求解效果。今后的研究内容主要有:a)对 MFA 算法的理论分析;b)不同子群设置模式对种群多样性的影响研究。

参考文献:

[1] YANG Xin-she. Nature-inspired metaheuristic algorithms[M]. [S. l.]:Luniver Press,2008: 83-96.

[2] GANDOMI A H,YANG Xin-she,ALAVI A H. Mixed variable structural optimization using firefly algorithm[J]. Computers & Structures,2011,89(23):2325-2336.

[3] SAYADI M K,HAFEZALKOTOB A,NAINI S G J. Firefly-inspired algorithm for discrete optimization problems:an application to manufacturing cell formation[J]. Journal of Manufacturing Systems, 2013,32(1):78-84.

[4] SRIVATSAVA P R,MALLIKARJUN B,YANG Xin-she. Optimal test sequence generation using firefly algorithm [J]. Swarm and Evolutionary Computation,2013,8(1):44-53.

[5] KAZEM A ,SHARIFI E , HUSSAIN F K,*et al*. Support vector regression with chaos-based firefly algorithm for stock market price forecasting[J]. Applied Soft Computing,2013,13(2):947-958.

[6] CHANDRASEKARAN K,SIMON S P. Network and reliability constrained unit commitment problem using binary real coded firefly algorithm[J]. International Journal of Electrical Power & Energy Systems,2012,43(1):921-932.

[7] COELHO L D S,MARIANI V C. Firefly algorithm approach based on chaotic Tinkerbell map applied to multivariable PID controller tuning [J]. Computers & Mathematics with Applications,2012,64(8): 2371-2382.

[8] HORONG M-H. Vector quantization using the firefly algorithm for image compression[J]. Expert Systems with Applications,2012,39 (1):1078-1091.

[9] SENTHILNATH J,OMKAR S N,MANI V. Clustering using firefly algorithm:performance study[J]. Swarm and Evolutionary Computation, 2011,1(3):164-171.

[10] YANG Xin-she,DEB S. Eagle strategy using lévy walk and firefly algorithms for stochastic optimization[J]. Studies in Computational Intelligence,2010,28(4):101-111.

[11] 刘长平,叶春明. 一种新颖的仿生群智能优化算法:萤火虫算法[J]. 计算机应用研究,2011,28(9):3295-3297.

[12] GANDOMI A H, YANG X S,TALATAHARI S ,*et al* . Firefly algorithm with chaos[J]. Communications in Nonlinear Science and Numerical Simulation,2013,18(1): 89-98.