1. **编码基础**

头部/尾部

* Permitive 函数符（函数：均值方差等+运算符:加减乘除）
* Termitive 终结符

头部：函数符+终结符

尾部：终结符

终结符（输入）：a,b

函数符：+-×÷

+-aabaabbbb

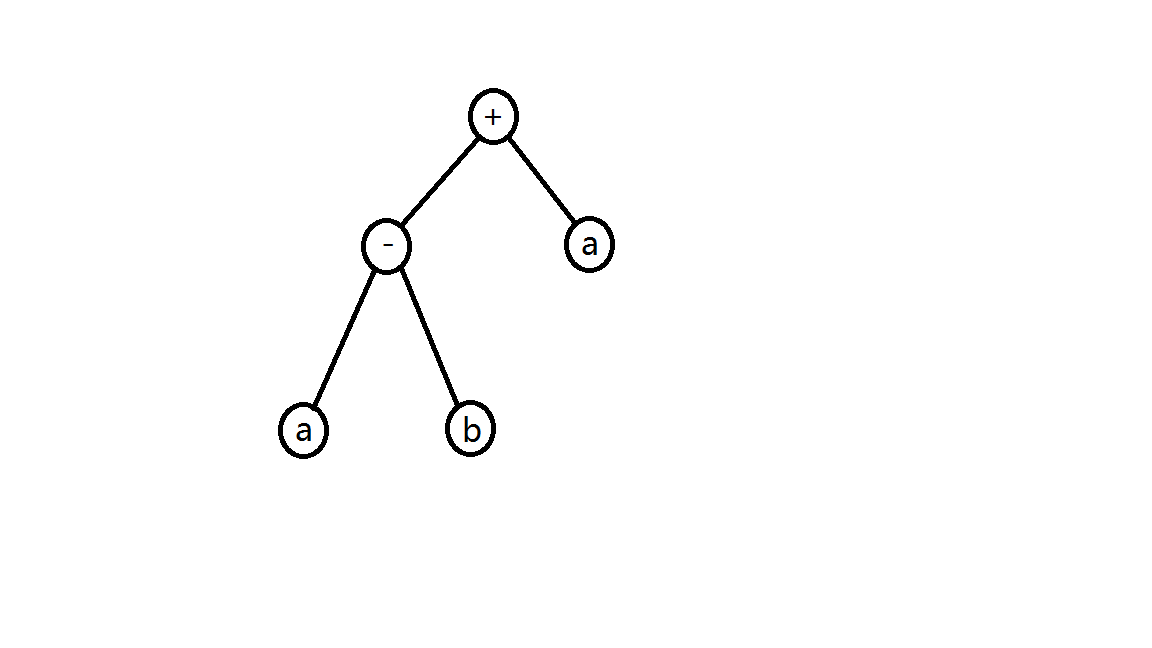


Fig Expression Tree

1. **编码规则**

尾部长度：t= h×(n-1)+1

* n：所需要变量数最多的函数参数个数
* h：头部长度（用于限制函数的复杂度）

若h=10,n=2→t=11，基因长度为21(二叉树)

设置头部和尾部的意图在于使基因链无论进行怎样的遗传操作都能产生可识别的函数式

* 当基因的第一个元素来自于终结符集T中，此时编码区域的长度为1
* 当基因头部的所有元素均来自于函数符集F，且对应的操作数均为最大操作数时，则编码区域长度达到最大，与基因的长度相等

编码区域（ET）+非编码区域

\* b+a-aQab+//+b+babbabbbababbaaa

任意遗传操作合理

\* b+a-a\*ab+//+b+babbabbbababbaaa

GEP中，变化的不是基因的长度，而是编码区域的长度

1. **算例**

**终结符输入量:**

1. Close
2. Open
3. High
4. Low
5. Volume

**函数符:**

1. Std7
2. Average7
3. Sum7
4. Cor7
5. Diff
6. protectedDiv
7. add
8. sub
9. mul
10. neg

**输出函数种群:**

* Mean7(Volume)
* Sum7(add(Open, Low))
* Mean7(neg(sub(Close, Volume)))
* Mean7(add(Volume, sub(Volume, High)))
* sub(Mean7(neg(Open)), Open)
* add(protectedDiv(mul(Close, Close), mul(Volume, Close)), Diff(Volume))
* Diff(High)
* mul(Cor7(Cor7(Volume, Open), Diff(Open)), Sum7(add(High, Open)))
* Diff(Open)
* Mean7(protectedDiv(Open, High))
* Cor7(Close, Volume)
* neg(Std7(Cor7(Low, High)))
* protectedDiv(Mean7(Mean7(High)), sub(High, add(Low, Volume)))
* Cor7(Mean7(Low), Volume)
* Std7(protectedDiv(Close, Low))

……

1. **评价个体**

**快速**的评价方式

减少迭代时间

**进化过程**

复制→变异

**复制：轮盘赌**

选择算子根据适应度和比例选择（赌轮选择方法）来随机确定下一代的个体

复制算子只复制选择算子所选择的个体

这些个体组成了下一代的初始群体

**变异：**

在单个染色体上，对染色体的每一位进行随机测试，如果满足变异的概率，则重新产生该位的编码

头部变异时可以选择所有的符号

尾部变异时只能选择终结符

**插串：**

随机在基因中选择一段子串, 然后将该子串插入到头部的随机指定的一个位置(但不能是第0个位置), 将头部的其他符号向后顺延, 超过头部长度的编码将被截去。

根插串

**单点重组：**

单交叉点

**双点重组：**

双交叉点

……

**工作安排**

假设：函数表达式相似的函数→因子表现相近

* 加入回测的评价部分
* 通过RL赋予算法记忆能力

**工作内容汇报**

1. 基因遗传进化算法

* 找到局部最优
* 每一次运行程序的最后一代种群相似度非常高
* 每次运行程序后可以找到一个非常有效的结构

组合上述结构：

1. 暴力遍历全部空间

* 保证生成的公式没有重复
* 排除一些无意义的公式，如sub(Close,Close) ; Cor7(Volume,Volume)等
* 优化评价函数，每日剔除因子值小于0.1或大于0.9的股票以去除涨跌停的影响；计算2013-2018每一年的因子分布图的极值

1. python多进程运算

* 多进程=多个python在同时运行，分别调用每一个CPU内核
* 结果表明有一定运算加速，并且还有提升空间

**结果分析**

* 进化算法每次运行可以找到近似局部最优解
* 直接遍历可以随机找到一些相关性较低的有效因子
* 找到的因子普遍非常奇怪，量纲不一致，不属于某种类型的因子

**下一步工作**

* 寻找更有效的遍历函数空间的算法
* 增加输入函数的个数（max,min等），增加输入参数（turnover等）
* 继续提升并行计算的效率