分类号 _____ 密级 _____

UDC _____ 编号 _____

中国科学院大学
硕士学位论文

伪布尔问题的启发式方法及其应用

于宝新

指导教师 李勇坚 副研究员

中国科学院软件研究所

申请学位级别 硕士 学科专业名称 计算机应用技术

论文提交日期 2013年4月 论文答辩日期 2013年5月

培养单位 中国科学院软件研究所

学位授予单位 中国科学院大学

答辩委员会主席 _____

Typeset by LaTeX 2$_\varepsilon$ at April 3, 2013

With package `CASthesis` v0.2 of CTEX.ORG

Heuristic Approaches to Pseudo-Boolean problems and Its application to Power Estimation

Baoxin Yu

Supervisor:

Prof. Yongjian Li

Institute of Software
Chinese Academy of Sciences

April, 2013

Submitted in total fulfilment of the requirements for the degree of Master in Computer Application Technique

摘　　要

对于超大规模集成芯片，过高的瞬时功率可能会导致运行时错误，降低其可靠性；同时，高功率带来的诸如散热的问题，也会影响芯片的性能。于是，要设计高性能，高可靠性的电路，就一定要获得它的一个精确估计的最大瞬时功耗数据。估计组合逻辑电路的最大瞬时功耗是一件非常困难的事情，因为电路的功耗一般是其输入的强函数。这就意味着，要获得电路的最大瞬时功耗而要进行模拟的输入数量和输入的数目是成指数关系的，确切的说，当输入个数为N时，需要模拟的次数将会达到4^N。我们接下来介绍了一个简单但合理的电路功耗模型。同时规约到一类特殊的PB问题上，而后分别用三种启发式方法进行求解。最后对一组benchemark 进行了测试，实验结果显示，我们的方法是可行的，而且效果很好。

关键词： 伪布尔问题，电路峰值功耗，启发式方法

Abstract

Excessive instantaneous power consumption may cause run-time error, which may reduce the reliability of VLSI chips. At the same time, high power may also lead to heat dissipation problem, which may affects the performance of the chips. Hence, to design circuits with high reliability and performance, it's imperative to efficiently obtain a precise estimation of the maximum power dissipation. But, estimating maximum power dissipation for a CMOS logic network is no easy, because the power dissipated by the network is typically a strong function of the network's inputs. This implies that the number of simulations which must be performed in order to find the maximum power dissipation is exponential in the number of the inputs to the network. Rather, it's 4^N when the number of inputs is N. In our paper, we will introduce a simple but reasonable model of power dissipation. We will then make it a special Pseudo boolean problem, which will be solved by three heuristic algorithms. At last, we test the performance of our algorithms in a benchmark circuits, and the result shows that, our algorithms are viable and effective.

Keywords: Pseudo-Boolean Problem, Power-Estimation, Heuristic Algorithm

目　　录

表　　格

插　　图

第一章　引言/绪论

1.1　背景介绍

接下来，我们将依次介绍伪布尔函数、电路峰值能耗估计以及启发式方法的一些背景知识和最近的研究情况。

1.1.1　伪布尔函数

首先讲解一下伪布尔函数[31]。伪布尔函数一般都具形式：$\mathbf{B}^n \to \Re$，其中$\mathbf{B} = \{0, 1\}$，表示一个布尔域，n是一个非负的整数，叫做参数数量。任意伪布尔函数都可以唯一的写成一个多线性多项式的形式：

$$f(x) = a + \sum_i a_i x_i + \sum_{i<j} a_{ij} x_i x_j + \sum_{i<j<k} a_{ijk} x_i x_j x_k + \dots \tag{1.1}$$

我们知道，布尔可满足性问题很难高效求解。因为这和它的表示方法就有着很直接的联系。目前我们遇到的经典问题，没有一个是可以在这种表示方法下，在线性时间求解的。在伪布尔函数中有一类具有次模函数特性，因此可以用多项式时间的算法最小化它们。这使得某一类SAT问题存在另外一种表示方法，来克服常规表示法带来的求解困境。如果一个布尔函数，对所有的x和y具有性质1.2，那就具有次模函数特性。

$$f(x) + f(y) \geq f(x \vee y) + f(x \wedge y) \tag{1.2}$$

这类问题是伪布尔函数中最有研究价值的，这也是伪布尔函数存在的最大价值。近些年来，随着研究人员对伪布尔函数研究的深入，对伪布尔函数进行了更细致的划分，同时，针对不同类别的伪布尔函数，逐渐产生了一些优化的方法。如用封闭的函数表达式表示的函数的优化[4]。为了挖掘某类问题的可优化特性，科研人员做了大量的工作。对伪布尔函数的形式进行规约，设计一些高效的求解算法等等。但大多数SAT问题不具有次模函数等可优化的性质，而伪布尔函数更多的作为一种表示方法存在。

举一个伪布尔函数的小例子：布尔表达式$x \vee y \vee \neg z$，可以用如下方式等价表示$x' + y' + \overline{z'} \geq 1$，这里的$x$,y ,z都是布尔变量，而x' ,y' ,$\overline{z'}$ 则都是二元变量，其取值为0或者1。这里还有如下关于$\neg z$的定义，$\neg z = 1 - z$。这种表方法就是伪布尔表示法，即其中的自变量定义域为$\{0, 1\}$。

1.1.2 启发式方法

搜索问题，我们常使用蒙特卡洛法，一种简单的搜索算法。特别是我们对可行解空间的分布没有直观或者正确认识的时候。因为算法本身的随机性，因此会对解空间进行大量的无用搜索浪费搜索时间而效率低下。我们希望我们设计的算法能在最有希望到达最优解的节点上加以扩展，避开那些无用节点，从而降低算法在无用节点上浪费的搜索时间，提高算法搜索的效率。首要的目标就是识别和扩展解空间中的“有用节点”。这就是启发式算法的基本思想，它是对深度优先搜索的一种改进，即在当前节点上扩展搜索，同时根据一些启发式信息，在搜索的时候，只选择那些最佳的分支进行再搜索。启发式算法达到的效果的好坏，就是看我们能不能找到准确的启发式信息，以及是不是能准确的表示和利用启发式信息。

1.1.2.1 启发式方法的理论依据

启发式搜索算法作为一种基本搜索方法的扩展，主要的理论依据如下[8]：

1 > 无论是动物或者人，都善于利用一些搜索信息来帮助自己选择正确的搜索方向。如图1.1，猎狗在搜寻猎物的时候，会习惯的依靠在当前位置猎物遗留的气味确定猎物逃跑的可能方向；人在寻找一些遗失物件时，也总是使用一些遗留的“蛛丝马迹”来帮助自己。在我们的搜索系统中，这些线索系统的称为启发式信息。我们要做的核心工作就是，充分利用这些有可能体现可行解空间分部信息的线索来优化我们的搜索路径，减少搜索时间，提高搜索效率。

2 > 我们的搜索问题，往往只是需要一个解而不是全部解，即猎狗搜寻猎物的目标，不是要找到所有猎物的位置，而只是想找到某个或某一类而已；同时，我们往往要一个优化的解，而不是最优的解，即猎狗找的可能是一只大兔子，而不可能是林子里最大的兔子，因为这个对猎狗来说确实比较困难。

3 > 启发式信息可以避免某些领域中组合爆炸的问题。当我们要构造的可行解空间太大的时候，我们机器的性能可能无法满，这里不仅仅指内存，甚至是那些最廉价的计算资源。如果我们采用深度搜索去搜索某条路径的话，一边求解，一边构造，就能避免以上问题。

图 1.1: 布鲁诺和它的鼻子

1.1.2.2　启发式方法的表示方法

启发式函数可以看成是一种映射函数，它能把问题的当前状态描述成一种接近目标解的程度。使用这个映射函数，就可以通过这种“接近程度”来引导我们解决当前问题。在搜索算法的设计过程中，我们要考虑问题状态的各种特征，各种因素。怎样对我们关心的方面做出设计，怎样考虑对某些因素的加权等等。只要这些特征或因素有利于特征函数对当前节点是不是在我们通往优化解的路径上做出尽可能准确的估计，我们都要利用其来设计我们的启发式函数。启发式函数设计的好，对有效的引导我们的搜索过程有着至关重要的作用。但对启发式信息利用程度和启发式函数的复杂程度并没有直接的关系。有些情况下，非常简单的启发式函数搜索路径能够给出非常令人满意的估计，但对于其它的一些场合，还是需要非常复杂的启发式函数。因为我们的启发式函数是要根据问题的当前状态，确定用于继续求解问题所需要的信息，我们设计的启发式函数应该能够估计已找到的状态与目标状态的有利程度，这样启发式函数才能够有效的帮助我们寻找后继节点- 即下一步搜索的起点。启发式函数的最直接的作用就是，在我们扩展当前节点时，它能够帮助我们确定哪些节点应该从我们的搜索树中删除。

1.1.2.3 启发式方法的搜索方向

在接下来的求解过程中，并未涉及这方面的问题，但作为通用的启发式方法设计的一个因素，下面我们也作一简要介绍。在问题的可行解空间中，从某个状态开始，找一条到达目标状态的最好或者较好的路径，这就是我们搜索的目标。我们可以正向搜索，即从某一个状态单向搜索到达目标状态的路径。同时，理论上讲，如果我们能确定目标状态，可以逆向，或者从两个方向进行双向搜索。但是无论方向如何，我们要的是朝有利于达到目标状态的方向。这当然可以用我们的启发式函数来判定，即与目标状态的接近程度。由于本文问题限制，采用的是正向搜索。正向搜索的一个好处就是，更符合人们的思考过程，系统描述会更自然一些。或者说，更容易解释的结果状态。

1.1.3 峰值能耗估计

从集成电路诞生的那天起，芯片设计者就开始了对体积尽可能小的，同时集成规模又尽可能大的芯片的追求。经过几十年几代人的努力，半个多世纪以后，半导体制造工艺一跃达到了22纳米的水平[33]；同时，随着便携式设备种类和数量的激增，芯片制造商之间竞争越来越激烈。芯片的能耗和稳定性就是竞争最为激烈的领域之一。

设计额定能耗低的芯片有助于提高移动设备的续航能力，即移动设备在电池电量饱和状态下，连续工作时间长短的能力。芯片的能耗越小，在电池电量一定的情况下，设备的续航能力越强，越能博得人们对于该种电子产品的好感。如图1.2，苹果公司新款平板电脑$ipad3$在性能测试中显示，其续航时间为10小时；而微软公司推出的平板电脑$surface$的续航时间为8小时。

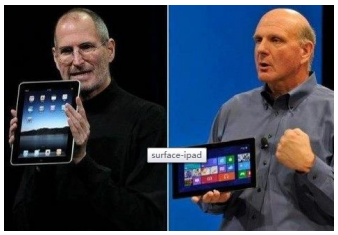

图 1.2: iPad和Surface

相比于平均功耗，设计稳定性高的芯片则更为重要。毕竟续航时间的延长可以通过应用更大电量的蓄电池来解决(我们暂时不考虑这种情况下成本的上

涨带来的负面效果)，而稳定性则关系到设备的性能和寿命。我们知道，芯片的瞬时功耗，从很大程度上影响着芯片的稳定性。因为高功耗带来的不仅仅是高瞬时电流，同时还有高散热。高瞬时电流会影响芯片稳定性，可能使芯片产生运行时错误，进而使芯片的性能退化[5]。同时高散热会使移动设备因为更大的散热设备而增大设计体积或者设计成本。同时持续的高散热甚至会损坏芯片，而使用户不得不花费更高的成本更换新设备。所以，在芯片设计完成之后，估计芯片的能耗，就成了一项重要而艰巨的任务。我们接下来的工作，就是对芯片能耗进行建模并计算。

1.2　相关工作

1.2.1　伪布尔问题

由于伪布尔表示法的使用，使得我们在多项式时间上，解决某些具有次模函数特性的布尔可满足性问题成为了一种可能。于是无论从理论还是实践上都有了长足的发展。本文只针对某一类特殊的伪布尔问约束问题，且不具备次模函数性质。这一类特殊的伪布尔约束问题的表示方法描述如下：

我们有如下的一个布尔函数f_i，参数为二元变量$x_1, x_2, x_3 \ldots x_n$，其中$i \in (0, m]$，我们要达到的目标是，寻找这组二元变量的一组赋值，使得如下的这个目标函数能达到最大值：

$$\sum_{i=1}^{m} W_i \times f_i x_1 x_2 x_3 \ldots x_n \tag{1.3}$$

这里的W_i和f_i相关，表示的是一个自然数，我们通常要求它的值大于0。同样举个例子，我们假设$m = 4$，二元变量为$\overrightarrow{x} =< x_1, x_2, x_3 >$,

$$\begin{aligned}
f_1\overline{x} &= \neg x_1 && W_1 = 1 \\
f_1\overline{x} &= \neg(x_1 \wedge x_2) && W_2 = 2 \\
f_1\overline{x} &= \neg(\neg x_1 \wedge (\neg(x_1 \wedge x_2))) && W_3 = 1 \\
f_1\overline{x} &= (\neg(x_1 \wedge x_2)) \vee x_3 && W_4 = 1
\end{aligned}$$

f_i对应的W值分别为$\{W_1 = 1, W_2 = 2, W_3 = 1, W_1 = 1\}$，我们希望在这种约束下，求解上面给出的目标函数1.3的最大值以及对应该最大值的二元变量的赋值。

我们能够通过计算得到这组二元变量的一组赋值$\{x_1 \to 0, x_2 \to 1, x_3 \to 0\}$，在这组赋值下，所有的$f$函数的值都为1，因此，我们目标函数最大值为5。而这种布尔变量的赋值就是我们要求解的“解”的形式。而给出的赋值和在此赋值下产生的权重，即是我们要求解的目标。很明显，以上这个PB问题是NP难的，它的时间复杂度和二元变量数目成指数关系。当二元变量的数目很大的时候，我们将很难精确计算目标函数的最大值。

1.2.2 启发式方法

启发式(heuristics)算法，来源于希腊语–发现(heuriskein)一词[29]，即在搜索算法中利用一些启发式的信息，提高算法的运行效率。

在事物的演化过程中，神奇的大自然构造了很多巧妙的方法和运行机制。人们从大自然的运行规律中得到启发，提出了很多解决实际问题正确且高效的方法。对于那些由于受到大自然运行规律或运行机制启发而得到的解决具体问题的经验、规则，进而得出的方法，人们习惯的称之为启发式方法。当然，一些启发式信息也来源于人们积累的丰富的工作经验。启发式方法诞生于上世纪四十年代，伴随着计算机技术的不断发展，取得了巨大的成就。计算机科学的两大基础目标，就是发现可证明的、执行效率高且可以得到最优解或次优解的算法。而启发式算法，通常能发现很不错的解，但同时也没有办法证明它不能得到较差的解；它通常可以在合理的时间给出解，但同样没有办法每次都可以以同样的速度得到解。

既然启发式方法有以上的这些不足，我们为什么还要去尝试设计和使用启发式的方法呢？

在计算机科学领域，针对NP难[22]问题，各种经过复杂优化的启发式方法，是我们解决那些在现实生活中存在，且必须要解决的问题的唯一的可行选择。当我们针对一些NP难问题求解时，那些我们习惯了的经典算法，在求解时花费的时间太长甚至完全不能在可接受的时间范围内给出优化解。这里的可接受范围，一种理解是，在这个时间范围内求解出的解，对于解决我们的问题还有意义。另外一种理解就是，可以预见的时间。总之，对于这些问题的求

解，使我们不得不求助于启发式方法。

尽管发展了这么多年，但启发式方法目前缺乏统一、完整的理论体系。同时，启发式算法各有优劣，大多数方法要结合具体的问题和实际的经验进行算子设计和参数的设置。这些问题不仅仅关系到算法的执行效率，甚至会影响到优化解的好坏。

但令欣慰的是，我们面对的大多数问题，都只要求我们计算一个优化解[1]，而不是一定需要最优解或者是全部的解。这让启发式方法这种退而求其次的方法广泛的应用于人工智能等领域。

1.2.3　峰值能耗估计问题

对于一个有N个原始输入的动态$CMOS$电路来说，要模拟获得电路的最大峰值能耗，需要输入的测试用例的规模为2^N个。而对于静态$CMOS$电路，由于电路当前状态对于上一时刻输入的依赖性，为了模拟获得该电路的最大的峰值功耗，需要输入的用例规模为2^{2N}个，即4^N个测试用例。如果是基于SAT描述的模型，该问题就是典型的NPC问题[14]，复杂度会相当的大而是求解变得不可能。同时，组合电路除了动静态之分，各种在硬件设计中各种技术的使用，使我们想设计一种简单通用，但同时又能准确模拟各种硬件能耗模型的工作变得更加困难。

比如为应对芯片电流泄漏问题而是用的$power-gating$(能量门[20])技术：这项技术将芯片分割成不同的功能块，每一个功能块在特定的电压下工作。当功能块暂停不使用的时候，可以切断该功能块的电源使之进入休眠。休眠晶体管用来连接功能块到电源或者地电压。在使用能量门技术的芯片中，功能模块被唤醒时诱发供电反应。这使得求解整块电路的峰值能耗变得困难，因为我们不得不计算唤醒某块芯片的能耗。同时在EDA领域[10]，在芯片设计技术的发展过程中，也有一些芯片设计者的经验得以总结和提炼，使得在设计阶段就考虑芯片能耗问题成为一种可能。

当然，即便是颇富挑战，这些年来，也有一些模型和基于这些模型的方法不断涌现出来。

近些年来，在处理SAT问题上的进步和一些高效的求解器的出现，特别是其在EDA领域的的广泛使用，使得在底层运用这些工具解决实际的SAT问题

[1]这里的优化解，往往并不是指问题的最优解，而只是一个好到可以接受的解

变成一种可能，毕竟$CMOS$组合电路的峰值能耗问题能很容易表示成SAT问题的形式。给定一组布尔变量和一组约束，目标是求解一组满足约束的布尔变量的赋值[3]。同时，由于SAT问题到PB问题的天然联系，使得$0-1$线性规划问题也成了一种可能的求解方式[3]，即$0-1ILP$问题。当使用线性规划的方法时，一些商用的优化工具，比如IBM公司的$CPLEX$商用数学规划优化软件得以在这些问题上使用。同时，还有一些基于确定性和统计的方法[5]被运用来估计最大的能耗，即使用ATG自动测试生成技术。还有一些针对更实际的电路建模进行计算的，即考虑到实际电路中能量门技术的使用，用$ATPG$自动测试模式生成技术[7]，无论从时间效率，还是从结果上，都有了长足的进步。同时也有前人应用了一些优化技术，和我们的方法相似，叫基因点优化，方法选择一个基因点，根据MPP[12]动态变化，也能求出还算不错的解。而本文采用启发式的方法，根据我们问题的特性，设计和试验了三种启发式方法，来求解峰值能耗的问题。

1.3 本文的贡献和组织结构

本文的主要贡献在于，设计和应用了三种启发式方法求解以上提到的电路的峰值能耗估计问题。本文介绍一种电路的能耗模型，同时在该能耗模型上，设计和改进算法并得出的一些试验效果。其中第一章主要依次介绍了伪布尔问题、启发式方法和电路峰值能耗估计等几个方面的背景以及近些年的发展情况和一些研究成果；第二章主要介绍一种简单的电路能耗模型，以及我们如何从电路文件计算得到该模型；第三章是本文的核心，主要介绍了根据问题的特性设计的三种启发式方法，包括最简单的爬山法，为了避免简单爬山法的贪心策略带来的局部性缺陷而采用的模拟退火算法，以及后来设计的更复杂的算法–遗传算法。在本章中，会详细介绍各个算法各个算子的设计方法及之所以这样设计的原因，同时也解释了一些对硬件认识的盲区带来的算法设计上的妥协和一些参数随机设定带来的理论缺陷。第四章介绍了三种启发式算法在一组测试用例上的实验结果以及和一些其它方法的对比。同时，横向比较本文的三种算法，针对得出的实验结果，还总结了一些经验，通过实验结果去认识问题，重新去审视上一章节中设计的算法。一些简单的设计却反而能得出相对较好的结果，而复杂的设计却没有表现出预期的效果。第五章主要介基

于$Forte$[2]平台和STE做的一些优化。在该章中，我们首先介绍了STE，然后介绍了一种基于STE的优化方法，尽管后来证明该优化方法并不适合我们芯片峰值能耗估计的问题及我们建立的能耗模型。第六章是在解决该问题的过程中总结的一些心得体会，并尝试了一些新的方法来在同样的模型下解决问题。基于时间原因，有些方法并没有去实现并观察实验效果，但给出了具体的想法和设计思路，希望抛砖引玉。

[2]Forte由因特尔公司开发使用

第二章　模型抽象

2.1　问题描述

估计$CMOS$逻辑电路的峰值能耗是困难的，因为该电路的能耗，通常是电路输入的强函数[19]。这也就意味着，如果我们要求解某个电路的峰值功耗，我们可能要仿真所有可能的输入情况，而这个仿真的数量，很可能庞大到我们根本无法模拟，因为它是和电路的输入个数成指数关系的。

总的来说，估计一个逻辑电路的功耗，关系到传播延时、时钟频率、技术参数、电路拓扑结构等等诸多因素。当然，在$CMOS$电路中，最困难的还得要数要搜索的输入向量或者输入向量序列空间规模的问题。如果要精确计算产生峰值功耗的输入序列，算法要模拟的输入向量的个数将远远超出我们的想象。

为了建立一个简单的，对于计算可行的模型，还要做一些必要的假设来解决一些实际电路中繁琐的问题，其中包括忽略一些可能对于计算无关紧要的东西。同时，还要保证建立的模型能够合理的模拟$CMOS$电路的能耗。

2.2　能量消耗模型

一个简单的、对计算可行，同时又能很方便的从电路文件生成的模型，就是本节要介绍的重点。如何在准确体现电路能量消耗的同时，又对电路进行合理的假设？接下来我们将逐一介绍。

2.2.1　模型的建立

首先，我们要做的就是，用一个简单但合理的模型来模拟逻辑电路中门的能量消耗行为。$CMOS$逻辑电路中，节点的电压变化会消耗能量，而且这种能量消耗形式是构成电路能耗的最主要部分。这告诉我们，要计算电路的能耗，我们要找到电路的瞬时响应。即确定组合逻辑电路的最大电路转化活动。这样我们可以通过计算此次电路转化活动中所有门的电容的总的变化量，进而可以计算出最大的那个。我们根据门在组合电路中的物理特性，给定不同的权重。通过该权重，我们想要在模型中展示的是，门节点发生电压变化时，释放能量

的多少，同时正比于其权重。释放的电量越多，其权重越大，反之则越小。根据经验，门节点能耗主要是其中的电容在输出转化的过程中的充放电行为造成的，即电容从V_{dd}到地电位或者相反时消耗的电量。这里的V_{dd}电压表示的是门的电源电压。在我们的模型中，门节点的权重正是和其中电容的负载电量正相关。根据经验，我们假设其中输入电容和其驱动的门的数量线性相关。这样一来，我们就能很方便的从电路的拓扑结构中计算每个门节点的权重了。而门节点的行为可以用输入向量表示的转化函数来表示。同时，这个转化函数的取得，也会在接下来的章节中介绍。Srinivas Devadas在[19]中介绍了各中转化算法。

对于动态$CMOS$逻辑电路，在每个时钟周期开始时，所有动态门的输出都是要重置的。这样，我们解决起来会简单一些，当前状态确定的话，逻辑门的转化活动就仅仅依赖于确定状态下的输入向量了。而对于静态$CMOS$逻辑电路中的节点的输出状态，不仅仅依赖于节点的当前状态下的输入向量，还依赖于当前状态。和动态$CMOS$电路不同的是，其当前状态并不会在每次输入前都进行重置。所以，其当前状态是依赖于上一时刻的输入向量的。因此，对于静态$CMOS$电路的峰值能耗问题，我们就不再仅仅是找到一个输入向量，而是要找到一个输入向量序列，这个序列按时序输入，使电路的峰值能耗达到最大，或者，按照我们的模型，等价的，让发生转化的门节点的权重和最大。这样，对于以上描述的加权逻辑模型来说，我们的结果依赖于两个输入向量组成的向量对。

说实话，一次求解一个向量，这没什么不好，但我们有个更绝妙的主意。我们在保证复杂度不发生变化的情况下，这样等效的处理我们的电路。我们构造这样一个新电路，其中每个新的门节点按如下方式重构：对原来的电路做一份拷贝，然后将原始电路中和拷贝电路中对应的节点门进行异或构成新电路的门节点，同时用该节点取代原始电路和拷贝电路中的门节点，并保有原始节点的权重。我们接下来要做的就是，计算新电路的一个输入，使这些异或后得到的节点发生转化产生的权重和最大。这里的新电路的输入包括原始电路的输入以及拷贝电路的输入两部分。我们可以这样理解：原始电路的输入作为t_1时刻电路的输入向量$\overrightarrow{V_1}$，使电路处于某一种状态；而拷贝电路的输入作为t_1+1时刻的输入$\overrightarrow{V_2}$，确定在该种状态下，经过这一时钟周期，在两个输入周期间发生转化活动的那些门节点，将在我们的新电路中输入高电平。

2.2.2　假设条件

当然，建模过程中做的一些简化，使我们不得不首先声明一些在建模过程中所作的必要的假设。这些假设使我们的模型在简化的同时也真实的体现了我们要处理的$CMOS$电路的能量消耗行为。

1 > 我们假设$CMOS$电路中门节点的唯一电容位于门的输出节点上。想一想我们上面的建模过程，我们对于门节点权重做的假设。我们假设门节点的能量消耗是因为其中的电容的充放电行为；之后我们根据经验假设其电容的负载电量是和其驱动门的数量成线性关系的；而门节点的权重又是通过其驱动门的数量来计算的。这一连串的假设有一个前提，就是我们的门节点只在输出节点上存在电容。事实上是，只有输出节点上的电容负责驱动后继的门节点；只有它上面的电容是我们关心的。如果其上还有一些其它的电容，那么在我们计算相应门节点的权重的时候将会异常复杂。

2 > 无论电流是从V_{dd}上的路径到电容，还是从电容到地电位，电压的变化都是从V_{dd}到地电位或者相反。这一点容易解释，我们不想把问题无谓的复杂化，所以，在我们的建模过程中，假设电容的充放电行为消耗的电量是相等的。而这假设也是十分合理的。

3 > 对于一个$CMOS$电路，原始输入向量在每次时钟循环开始，都要进行稳定化处理。我想这大概是关于传播延时等电路的实际物理特性的要求吧。

2.2.3　模型的公式化表示

我们将以具体例子解释我们的建模过程：

组合门i，在原始输入$\overrightarrow{x}$下的逻辑方程是f_i。当原始输入从$\overrightarrow{V_1}$变化到$\overrightarrow{V_2}$的时候，引发的状态变化所导致的能量损耗方程按如下方式表示：

$$P_g = 0.5 \times C \times \frac{{V_{dd}}^2}{T_{cycle}} \times (f_i\overrightarrow{V_1} \oplus f_i\overrightarrow{V_2}) \tag{2.1}$$

这里的P_g是组合逻辑门节点i的能量消耗；C是该门输出的电量；V_{dd}是该门节点的电源电压，T_{cycle}是每个时钟周期输出的转化次数；f_i是门节点i的布尔

函数，$\overrightarrow{V_1}$和$\overrightarrow{V_2}$是门节点i在t_1, t_2两个相邻时刻的输入向量。

按照我们之前的假设，电容的负载电量C和驱动的门节点的数量线性相关。因此，从以上做出的假设，为了使瞬时能耗尽可能的大，我们可以得出目标函数如下。

$$\sum_{i \in G} O_g \times (f_i \overrightarrow{V_1} \oplus f_i \overrightarrow{V_2}) \tag{2.2}$$

我们的目标是计算这样的向量序列$< \overrightarrow{V_1}, \overrightarrow{V_2} >$，使上面的目标函数的值尽可能趋于最大。这里的$G$是所有门节点的集合，$O_g$表示的是该门节点的加权值。这样，我们求解最大能耗的问题就变成了，在输入向量的$2N$维空间中寻找一个向量，使我们的目标函数值趋于最大。为了清晰起见，简化我们的目标函数如下：

$$\sum_{i=1}^{m} \times f_{i'} \overline{xy} \tag{2.3}$$

其中m表示门节点的个数，W_i表示门节点i的权重，$f_{i'}$表示门节点i的新的状态转换函数(即原始电路和拷贝电路对应节点进行异或之后的转换函数)$f_i \overrightarrow{V_1} \oplus f_i \overrightarrow{V_2}$，而$\overrightarrow{xy}$表示的是我们要求解的输入向量序列，输入序列序列向量的维度为$2N$，其中N为原始电路输入的个数。所以我们也捎带计算一下问题的复杂度为：$2^{2N} = 4^N$。

2.2.4 具体实例

具体实例的电路图如下：

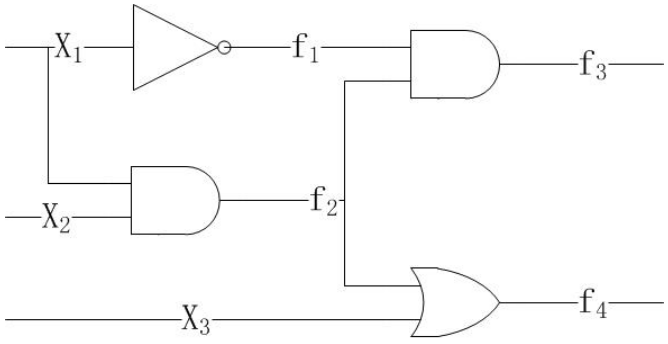

图 2.1: 原始电路拓扑

按照我们前面介绍的模型生成方法，对于该电路的原始建模如公式1.2.1。

同样，按照我们生成新的电路的方法，生成的新电路拓扑如下：

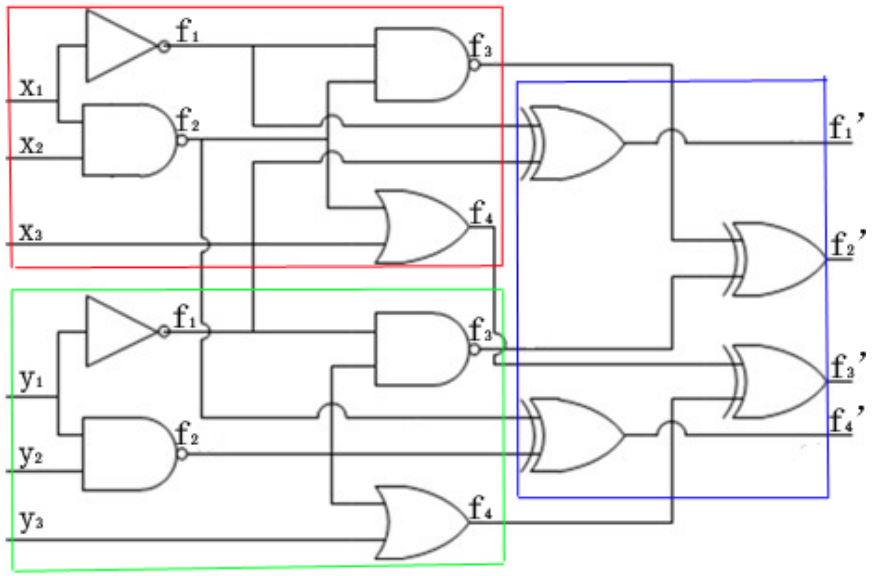

图 2.2: 新生成电路拓扑

电路的拓扑图的构成，分别是红色区域中的原始电路拓扑图；绿色区域内的拷贝电路拓扑图；蓝色区域内的，两份电路拓扑中，对应的门节点异或运算产生的新的拓扑节点。以上的电路拓扑图经过上面介绍的规则进行重新建模之后的到的新模型描述如下：

$$
\begin{aligned}
&f_1{}'\overline{xy} = \neg x_1 \oplus \neg y_1 && W_1 = 1\\
&f_1{}'\overline{xy} = \neg(x_1 \wedge x_2) \oplus \neg(y_1 \wedge y_2) && W_2 = 2\\
&f_1{}'\overline{xy} = \neg(\neg x_1 \wedge (\neg(x_1 \wedge x_2))) \oplus \neg(\neg y_1 \wedge (\neg(y_1 \wedge y_2))) && W_3 = 1\\
&f_1{}'\overline{xy} = ((\neg(x_1 \wedge x_2)) \vee x_3) \oplus ((\neg(y_1 \wedge y_2)) \vee y_3) && W_4 = 1
\end{aligned}
$$

各个新产生的门节点$f_i{}'$的权重维持不变，即仍然为$\{W_1 = 1, W_2 = 2, W_3 = 1, W_1 = 1\}$，上面的$\overline{xy} =< \overline{x}, \overline{y} >$，其中$\overline{x} =< x_1, x_2, x_3 >, \overline{y} =< y_1, y_2, y_3 >$。

这样的话，我们就将电路的峰值能耗估计问题就转化成了文章开始时所介绍的伪布尔问题的特例的描述形式。接下来我们就要用启发式方法解决这个问

题。即求解如下目标函数的一个优化解：

$$\sum_{i\leq 4} W_i \times (f_i\overrightarrow{V_1} \oplus f_i\overrightarrow{V_2}) \tag{2.4}$$

同样的小例子，和文章开始介绍的模型就有了比较大的变化，而单纯的给一组输入变量赋值变得不可行。当然，对这个小例子，我们也可以给出一组解如下：

最大权重为5[1]，对应的$< \overrightarrow{V_1}, \overrightarrow{V_2} >=< 1,1,0,0,1,1 >$，即两个时刻的向量分别是$t_1$时刻$\overrightarrow{V_1} =< 1,1,0 >$而$t_2$时刻$\overrightarrow{V_2} =< 0,1,1 >$。感兴趣的话可以手动验证一下。

[1]我们可以很肯定，即便我们是用启发式的方法求的的，因为5便是该电路能达到的权重的上界。

第三章　启发式算法

一些启发式算法的框架，可能大家都很熟识了，因为这些东西前人都做了研究和总结[6]。我们这里一共使用了三种启发式的算法来求解目标函数的优化解，从起初最简单的爬山法，到算子更多更复杂的遗传算法。我们完成了三个算法中算子的设计，同时对算法框架进行了修改。为了检验算法的效果，我们找到了一组benchmark作为实验数据对算法的效果进行了测试。在后面的章节中，我们还会展示多组实验结果，并对算法的实验效果及产生原因进行全面的分析。

3.1　电路文件预处理

电路的能耗模型是从*.bench*的电路文件中得到的，模型结构包括两部分，各门的逻辑方程及其权重，描述成*Forte*的表示形式如下所示：

$$weighted\_list < vector > :: integer \ ;$$
$$bool\_function\_list < vector > :: bexpr \ ;$$

由电路文件得到模型两部分结构的算法，参见[17]，这里不做过多介绍。

3.2　爬山法

3.2.1　爬山法背景

也许爬山法(*Hill Climbing Algorithms*)[30]叫瞎子爬山法会更贴切一些。我们可以想象，瞎子在爬山的时候会是一种什么样的场景：瞎子肯定是不会使用自己的眼睛，这一点我们是肯定的。没错，他会使用他的手杖去判别自己周围的高低情况，即通过探测局部的情况来爬山。我们需要澄清一点，瞎子可能永远也爬不到一个很高的山顶，因为他非常可能不知道山顶在哪儿，而只是能准确判断自己周围哪里更高一些而哪里更低一些。我们所谓的爬山法，就是我们只能通过局部信息的反馈进行的搜索算法，是一种贪心的搜索算法。他的过

程很简单，从当前解出发，算法每次从当前解的临近的位于可行空间的解集中选择一个最优解取代当前解，直到达到一个局部最优解。

3.2.2 算法描述

1 > 生成第一个候选解，如果满足局部条件，搜索停止，

2 > 从当前解生成其临近的解集合，

a) 使用评估函数评价候选解集合的所有解；

b) 选择其中最优解，保留其为临时最优解；

3 > 当前解优于临时最优解，搜索停止，否则用临时最优解替换当前解并转到第2 >步。

3.2.3 算子描述

首先，我们先将算法的框架展示一下:

Algorithm 1 $Hill\_Climbing\_Frame\, weighted\_list, bexpr\_list$

```
cur ← Initial();
best ← Evaluation(cur);
repeat
  neighbors ← Neightbor(cur);
  while ∃unEvaluated_neighbor do
    Evaluation(neighbor);
  end while
  loc_best ← Max(neighbors);
  loc_best_ass ← Evaluation(loc_best);
  if loc_best_ass > best then
    best ← loc_best;
    cur ← loc_best_ass;
  end if
until loc_best < best
return cur;
```

这个框架里使用了三个主要算子，它们分别是：

1 > *Initial*算子产生第一个位于解空间的候选解；

2 > *Evaluation*算子即评估算子，根据我们的目标函数，能对候选解的优劣进行评价，通常的使用规则是判断其与目标解的相似程度；

3 > *Neighbor*算子产生当前解的局部邻域，即新的候选解的集合，不同的产生规则，决定了不同的邻域结构，但一般都倾向于使用简单的产生规则。

在上面的章节中我们介绍过目标解的形式，即布尔向量和其赋值的形式：$< \overrightarrow{V_1}, \overrightarrow{V_2} >=< 1,1,0,0,1,1 >$，其中的$\overline{xy} =< x_1, x_2, x_3, y_1, y_2, y_3 >$为布尔向量，即新构造电路的输入，而→后的部分为其对应的赋值。

*Initial*算子：在我们问题的原始模型1.2.1中，目标解是一个被赋值了输入向量的序列$< \overrightarrow{V_1}, \overrightarrow{V_2} >$，表示两个连续时刻输入向量和其对应的输入值。当两个输入向量赋值完全相同时，我们也很容易求得目标函数的值为0，因为第二次输入和第一次输入之后，电路的状态完全不会改变。从我们的新电路模型中来看，没有一个门节点的输出会是1。

在我们要处理的新的模型2.2.3中，我们输入序列被作为同一时刻的输入向量的两部分。我们知道，爬山法是很依赖初始值的，所以我们不希望构造两个相同的向量作为算法初始值，评估结果很差(或者说太差了，因为评估函数的计算的权重和为0)。为了避免两部分完全一样，同时又避免完全的随机产生全部的变量赋值，充分利用原始和拷贝两部分输入的关系，我们按以下方法生成初始解：首先随机产生输入向量$\overrightarrow{V_1}$中所有布尔变量的赋值，然后逐个取反，并最为输入向量$\overrightarrow{V_2}$中对应位置上输入变量的赋值。

Algorithm 2 *Initial_Operator input_length*

```
vector < x̄, ȳ >:: bool ini;
for i = 1 to input_length do
    x_i ← random(0, 1);
    y_i ← 1 − x_i;
end for
return ini;
```

当然，在具体的算法中，对于长度为N的原始输入向量，我们是一次性生成所有原始输入的赋值：生成一个$[0, 2^N)$的一个随机数$rand$，然后将对应的二进制位上的值作为输入向量对应位置变量的赋值，同时，用另外一组赋值的二进制数为$2^N - 1 - rand$。尽管方法有差异，但我们知道，效果是一样的。因为，

逐个生成原始输入和同时生成全部输入，对于单个*input*来说，其值为0或者1是等概率的。这样的好处是，我们利用了新的电路模型是由原始模型和拷贝模型进行异或得到的这一特性。本质上，我们的目标是，在产生初始值之初就获得较大的目标值，这样可以加速收敛。但并不是说，更好的初始值就会产生更好的解。

Algorithm 3 *Evaluation_Operator* $weighted\_list, bexpr\_list, assign$

```
weight ← 0;
length ← bexpr_list.length();
for i = 1 to length do
  if substitute(bexpr_list.at(i), assign) = true then
    weight ← weight + weighted_list.at(i);
  end if
end for
return weight;
```

*Evaluation*算子是用来评估当前解的优劣的，这里的策略很简单，即把目标函数作为评估算子使用。同时澄清一点：在算法执行之前，问题并没有给定优化解的接受范围，所以没有办法给出以相似程度为结束条件的算子。这里的评估函数并不是以评估当前解与目标解的相似程度为目标而只是单纯的希望得到更大的值。

Algorithm 4 *Neighbor_Operator ini*

```
vector < assigns >:: assignment neighbors;
length ← ini.length();
for i = 1 to length do
  neighbor ← ¬ini.at(i);
  neighbors.insert(neighbor);
end for
return neighbors;
```

至于上面的*Neighbor*算子：所谓*Neighbor*，就是你的邻居，这里该算子的目的，就是生成当前解在解空间的邻域，即我们要探测的局部区域的候选解集合。比如一维的区域上，我们可以在这一维度上对数值(进制大于2)进行增大或减小来产生两个邻域。由于我们这里使用的是布尔变量，我们所有变量值都是二元的，所以我们在每个维度上进行的计算不是增大或者减少，而是取反。这

样我们对每个维度上的赋值进行取反之后，我们就会得到很多的新的候选解，具体的数目是N，每个赋值都作为当前赋值的一个邻域。

3.2.4　爬山法的缺陷

我们用图3.1来说明：

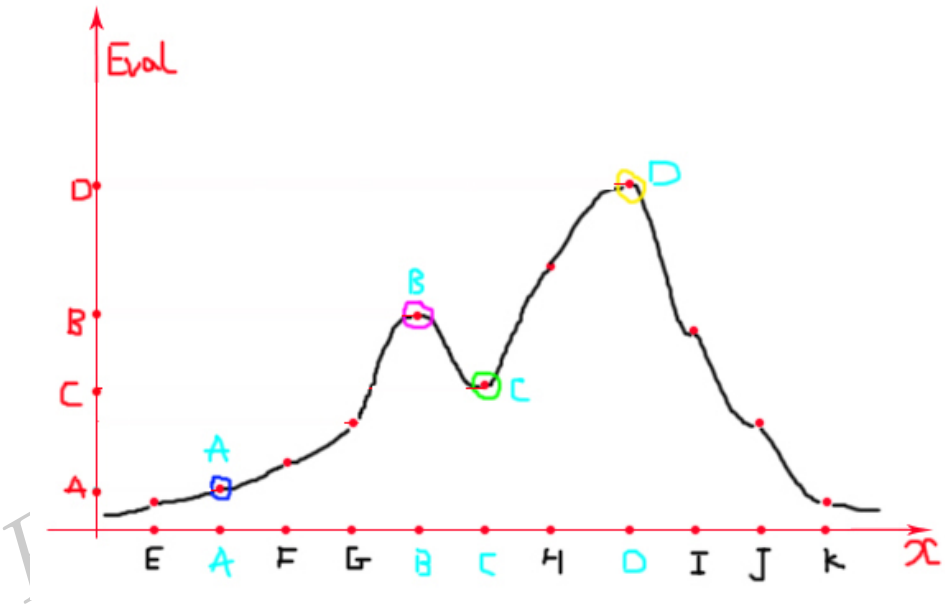

图 3.1: 一维爬山法图例

我们这里主要讲解$Hill\ Climbing$算法的贪心策略与生俱来的缺陷：

首先是局部性缺陷，即如上图，假设A是初始化点，按照以上描述的算法，当算法执行到达B后，发现领域G和C的$Eval$值都比当前值小，算法随即执行完毕并返回一个局部最大值。这个过程说明，算法得到的解太依赖于初始值，而且贪心算法本身就带有局部性的缺陷。如果我们的算法能达到C，进而就可以从C就能到达D这个全局最优解。接下来我们介绍模拟退火算法，首要的目的就是客服局部性的缺陷，减少优化解对初始解过分的依赖性。

3.3 模拟退火算法

3.3.1 算法背景

模拟退火(*Simulated Annealing*)源于冶金学的专有名词退火：即将材料加热后再经特定的速率进行冷却，目的是增大晶粒的体积，并且减少晶格中的缺陷[32]。材料中的原子原来会停留在使内能有局部最小值的位置，加热使能量变大，原子会离开原来位置，从而随机在其它位置中移动。退火冷却时速度较慢，使得原子有较多的可能可以找到内能比原先更低的位置。

我们上面介绍了，瞎子爬山法使用的是贪心策略，每次都只会选则当前能够达到的最优解。当选无可选时，算法即结束，因此只能搜索到局部最优解。模拟退火算法算的上是爬山法的一种改进，它本身使用的也是一种贪心策略，但模拟退火算法在搜索过程中加入了随机因素[1]，即以一定的概率接受一个比当前解要差的解。这就使模拟退火算法具有了一种能够跳出局部最优解的能力，从而使得算法有可能达到全局最优。这里没有确切的证据显示，在有限的时间内，模拟退火算法可以达到全局最优；甚至也没有确切的证据，在有限的时间内，模拟退火算法能找到比爬山法更好的解。但我们可以理解的是，它的确从算法本身设计中，就具有了客服了爬山法死板的局部性特性的能力。

3.3.2 算法描述

1 > 生成当前解，评估当前解，并设置跳出条件。

2 > 通过产生函数从当前解产生一个位于解空间的解loc（这里的产生函数的变化方法通常也很简单，但产生解的方法同样决定了的邻域结构）。

3 > 评估当前解和新产生解，计算目标函数差Δt。

4 > Metropolis法则：$\Delta t > 0$则接受loc为当前解，否则以概率$e^{-\frac{\Delta t}{T}}$为概率接受当前解。如果不满足迭代次数则转到第2 >步，否则跳出，算法执行完毕。

3.3.3 算子描述

和$Hill\ Climbing$算法相比，模拟退火算法的$Initial$3.2.3和$Evaluation$3.2.3算子都是一样的策略，只有$Neighbor$算子是不同的，这是因为在$Hill\ Climbing$中

的$Neighbor$算子是产生一个位于解空间中解的集合，而这里只是产生一个位于解空间的解。

其中$Neighbor$算子可以由爬山法的算子稍加改动，不是对每一位取反，而是随机的选择一位进行取反。这里比较关键的是随机概率的确定，如果设计的过于苛刻，则模拟退火算法就会退化成贪心的爬山法。而如果随机概率设计的过于宽松，那么，算法将很难收敛。放在我们的框架下，由于跳出条件是以执行的循环的次数决定的，就会表现为得到比爬山法更差的解。的我们接下来就着重介绍一下模拟退火算法的算法框架：

Algorithm 5 $Simulated\_Annealing\_Frame\ weighted\_list, bexpr\_list$

```
cur ← Initial();
step ← 0;
T ← Evaluation(cur);
while step < MaxStep do
  loc = Neighbor(cur);
  Δt ← T − Evaluation(loc);
  if (Δt > 0) ∨ random() > pro(Δt, T) then
    T ← Evaluation(loc);
    cur ← loc;
  end if
  step ← step + 1;
end while
return cur;
```

Algorithm 6 $Neighbor\_Operator\ ini$

```
length ← ini.length();
i ← random(1, length)
return ¬ini.at(i);
```

再以上图为例3.1对比爬山法和模拟退火算法：我们之前介绍了，当爬山法到达B的局部最优后，算法会终止。而模拟退火算法则会以一定的概率跳转到C，然后经过H点到达全局最优的D点，而这就是我们想看到的。但注意的是，我并没有确定的说，算法执行一定会达到D点，同样的执行也可能达到E点，进而出现更差的解也说不定。

3.4 遗传算法

我们已经接触了两种启发式算法，不论从算法本身的复杂度上来看，还是从算子上来看，上面的两种算法都算不上复杂。我们这里更多的，以贴近问题的应用为主，因为启发式算法本是就是很开放的课题，可以挖掘的点很多，针对不同的问题，除了框架，算子本身的设计很灵活。我们能做的就是让我们的算子，尽可能的体现问题的特性，达到更好的计算效果。接下来我们介绍一种相对来说更复杂的算法：遗传算法(*Genetic Algorithms*)。这种复杂的算法能更大程度上的提高算子的灵活性，同时更多的应用概率，通过设置大量的参数来调整算法的性能。

3.4.1 算法背景

遗传算法[28]和以上两种算法一样，是计算数学中用于解决最优化的搜索算法，不仅有着深厚的生物学基础，而且还有强大的数学背景[26]。遗传算法是进化算法的一种，从名字我们也能够看出来。其最初的设计理念就是借鉴了达尔文进化生物学中的一些现象而发展起来的，这些现象包括了遗传，杂交，突变，自然选择甚至殖民入侵和迁徙[2]等，具有鲜明的认知学意义和坚实的生物学基础。我们稍后会介绍，这些神奇的进化机制会怎样以算子的形式体现在我们的算法中。

遗传算法通常实现为一定数量的候选解的抽象表示的种群向更好的解的进化的模拟[24]。我们把候选解抽象为种群中的个体。个体表示为一个变量序列，进而被抽象为染色体。染色体的表示方法和待解决的问题是有联系的，不同的问题，编码方式稍有不同。通常的染色体的表示方法和我们的解的表示方式很相似，为2进制0/1串[27]。进化从由随机个体组成的种群开始，之后在每代发生。在每一代中，种群中每个个体的适应度会被评估，从而从当前种群中，根据其适应度的大小，随机选择多个个体，这个阶段我们称之为父代的选择阶段，而选中的个体就组成了相对优化的种群。当然选择不是完全按照适应度来进行的，而是依据遗传规则：适应度高的个体，选中的几率高；适应度低的个体，选中的几率低。这种规则能用来避免产生局部最优解。然后在一定的交配率和变异率下，通过遗传和变异产生新的个体，新个体再融入到种群中作为新的种群。如此往复若干代之后，种群整体向着整体适应度高的方向发展，进而完成进化的过程。算法完成条件，可以是进化的次数限制，人干预，满足条件

的个体已经找到等等，我们也要具体问题来具体设计。当然，算法本身是很灵活的，很多参数和算子都可以根据问题的不同做相应的调整。

3.4.2　算法描述

我们根据我们要处理的PB问题的具体特点对算法框架进行一些必要的，更切合我们问题特性的修改：

传统的遗传算法中，一般采用两性生殖，交叉遗传的方法，即父母双亲会将自己的染色体分成两段，分别遗传给两个后代。同时新产生的两个后代会取代父母双亲，而没有生殖后代的个体保留。在我们的问题中，个体虽然也是二进制编码的0/1串，每一位表示一个输入，但由于电路的拓扑结构在中间门多的时候变得很复杂，所以不能很简单的分成两部分来分别进行遗传，因为基因在本问题上互相依赖，而传统的基因，其片段与片段之间的独立性更高。但在本问题中，单纯的交叉遗传会打破个体适应度高的性质，而使产生的个体不再保有双亲高适应度的性征。同时，由于不使用两性交叉生殖，子代不再单纯取代双亲。

1 > 初始种群生成，并评估每个个体；

2 > 对种群中的个体做以下处理：

a) 以一定的比例将不同适应度的个体分成三类;

b) 适应度最高的一类，进行单性生殖;

i. 评估每个基因的优劣；

ii. 对最差的两个基因进行反转，这样会生成三个新个体。

c) 对适应度居中的一类，也进行单性生殖，并变异;

i. 评估每个基因的优劣；

ii. 对最差的一个基因进行反转；

iii. 变异，变异采用的是对选中的基因进行反转，而变异的基因个数会随着遗传的次数逐渐降低，产生两个个体。

d) 对使用度最低的一类，只进行单纯的变异。变异的基因个数随着遗传次数的增加而降低;

e) 对新个体进行评估，同时去掉因为反转和变异产生的重复个体，同时对新个体进行评估;

3 > 淘汰部分适应度低的个体，保持种群数量。如果遗传次数达到要求，算法终止并给出最好的个体；否则转向第2 >步。

3.4.3 算子描述

和上面一样，我们首先介绍我们算法的框架：

Algorithm 7 *Genetic_Algorithm weighted_list, bexpr_list*

```
gen_num ← Generation(inputs.length(), bexpr_list.length());
popu_num ← Population(input.length());
vector < assigns >:: assignment population;
while population.num < popu_num do
  indiv ← Initial();
  if ¬population.find(indiv) then
    population.insert(indiv);
  end if
end while
for i = 1 to popu_num do
  Evaluation(population.at(i));
end for
while generation < gen_num do
  Order(population);        Order them by weight
  Apomixis(population.front());
  Apom_Mut(populaton.mid());
  Mutation(population.left());
  Uniq(papulation);
  Selection(Papulation);
end while
return population.at(0);
```


我们接下来逐一介绍参数和算子：

首先介绍第一个参数：遗传的代数——即要进化多少次。这里的设置是和问题的规模成线性关系的，即便是我们并没有找到很好的线性模型来模拟两者的关系，即，比例是多少的时候能更快的同时更准确的求解得到优化解。因为还有一个很严重的影响问题规模的因素就是中间节点的个数。我们这里单纯的

将这两个参数分别加一个权重因子来平衡。我们可以如下设定：

$$gum\_num = \lfloor 0.1 \times N \rfloor + \lfloor 0.01 \times m \rfloor$$

这里的N表示输入向量的长度，而m表示中间门节点的个数。通常在电路中，中间门节点的个数要远小于输入的个数，于是，我们这样设定，给两部分影响复杂度的部分以权重。这只是一种没有理论根据的设定，因为这毕竟是一个数值，我们需要的是，在这个数值尽可能小的时候，得到更优秀的解。

接下来是种群数量因子，这个和进化次数一样，越多越好，同时和问题规模相关。根据我们的观察，问题的输入规模都在$N \in (50, 500)$之间，是一个比较好的区间，我们这样假设：

$$popu\_num = N$$

*Evaluation*算子：至于评估函数，我们仍然使用和爬山法以及模拟退火算法一样的方法，即使用问题的目标函数3.2.3。

*Initial*算子：这里是要找到*popu_num*个随机个体。我们的策略是选择无重复的个体，爬山法的*Initial*3.2.3算子每次产生一个个体，这里我们循环调用该算子，直到产生*popu_num* 个无重复的随机个体为止。

Algorithm 8 *Initial popu_num*

```
vector < assigns >:: assignment population;
while population.num < popu_num do
  indiv ← Initial();
  if ¬population.find(indiv) then
    population.insert(indiv);
  end if
end while
return population;
```

接下来我们给出一个很重要的定义：“坏基因”。

“坏基因”可以这么来描述：那些控制个体优良性状但去没有使当前个体展现出优良的性状的基因。哪些基因控制了优良的性状？基因在个体中怎样的状态，才算是体现了优良性状？接下来，通过两个计算策略给出“坏基因”的公式定义。

我们首先定义一个*input*的权重：我们共使用了一下三种策略：

$$
\begin{aligned}
&The\ Weight\ Of\ Input:\\
&Policy1: \quad \sum_{input \in f_i} W_i\\
&Policy2: \quad \sum_{input \in f_i} 1\\
&Policy3: \quad \sum_{input \in f_i} \frac{W_i}{\|f_i\|}
\end{aligned}
\tag{3.1}
$$

即输入的权重可以由包含它的所有门的权重总和决定；同时也可以按照包含它的门的总数来定；再就是用将包含它的门的权重按其包含的门中变量的个数平均分配，然后求和。这里$\|f_i\|$的意思是f_i门中输入的个数。这里的输入的权重，和具体变量的赋值无关。

权衡起见，我们采用以上的策略1，同时，这里没有对策略二、策略三的对比数据。

我们定义一个*input*的有效权重。相应于以上的策略，我们也有以下三种对应的策略：

$$
\begin{aligned}
&The\ effective\ Weight\ of\ Input:\\
&Policy1: \quad \sum_{f_i=true \wedge input \in f_i} W_i\\
&Policy2: \quad \sum_{f_i=true \wedge input \in f_i} 1\\
&Policy3: \quad \sum_{f_i=true \wedge input \in f_i} \frac{W_i}{\|f_i\|}
\end{aligned}
\tag{3.2}
$$

所谓有效权重，需要给定一个赋值，然后计算，在该组赋值中，该基因到底是好的是坏的。同一个输入/基因，在不同的赋值中可以表现出好的或坏的“性状”。

我们同样应用的是策略1。

接下来我们定义坏基因：所谓坏基因，就是该基因的权重与有效权重的差过于大。这只是我们的一个策略，相应的，我们可以用有效权重占权重的比例来定义其好坏。

我们这里的意思是，如果改变该基因，即将其值进行反转，那么我们可能在权重上获得更多的上升空间。这里是假设，因为有些值为0的f_i的值不会因为我们改变了一个变量而改变，同时也可能那些值为1的f_i的值会变成0从而使$weight$值变小。我们这里只是采用这种评价机制。

接下来是我们的单性遗传算子和突变算子：

有了基因的评价算子，那么单性生殖算子就很好给出了：

$Apomixis$算子：

Algorithm 9 *Apomixis_Operator indiv num*

```
children ← null;
while ∃unWeighted_gene do
  Policy1(gene);
end while
Order(gene); Order by the weight − eval_weight
children ← Reproduce(num, genes)
return children;
```


所谓num是用来控制其生成子代的数量的，单性生殖的个体，$num = 2$表示生成三个子代个体。生成法则，就是找到最差的2个基因，分别取反得到两个，全部取反，又得到一个。

这里之所以设置一个参数，是因为在另一部分中还会用到单性生殖(比如半单性生殖)。我们控制种群走向的方式是：让好个体(适应度高的)多产生子代，而差个体少产生子代。这里的单性生殖在我们的算法中叫$Total_gen$，意思就是完全是单性生殖。而一部分父代个体则进行半单性生殖半变异，在算法中叫$half_gen$。剩下还有一部分个体只进行变异，在算法中叫$Mutation$。三种生殖方法产生的子代个体的数目要依次减少。这些在算法中都是可控的，但我们并没有什么确切的参数设定规则。如果产生的子代过多，则会使种群更快的收敛。当然这里还有变异的概念在里面，它也是我们用来调整收敛速度的。如果变异率过低，则会导致快速收敛，而过高则会使种群出现波动而影响

收敛速度。那接下来我们就先介绍介绍变异算子，因为半单性生殖算子也要用到它，该算子包含一个遗传代数的参数，遗传次数越多，变异率越低。以下就是$Mutation$算子：

Algorithm 10 $Mutation\ indiv, gene\_num$

```
pro ← Pro(gene_num);
while mut do
  i ← Random(1, N)
  Inverse(indiv.at(i));
  mut ← Dicing(pro);
end while
return indiv;
```

接下来的这个算子，是上面提到过的半单性生殖半变异算子，即$half_gen$算子。因为这个算子不仅仅应用到了变异算子，还用到了单性算子，产生了两个子代个体。其中之一是由突变得到的，而另外一个个体是单性生殖反转最坏的一个基因得到的。我们的定义如下：

Algorithm 11 $Apom\_Mut\ indiv$

```
pair < child, child >:: assign children ← null;
children.fst ← Mutation(indiv);
children.snd ← Apomixis(indiv, 1);
return children;
```

这样我们就产生了两个新个体。至于处理适应度最低的那部分个体，我们采用完全的变异处理，即用$Mutation$算子产生单个个体。

算法的框架和各个算子的描述就完成了。关于一些理论论证算法性能方面的东西，这里没有涉及，只是在算法原来的框架上，结合问题的特性，对解空间进行合理编码，同时对一些算子进行了必要的修改，使之更贴近问题的本质。我们不能保证算法在有效的时间内，能够达到全局最优。同时，对于单性生殖的应用，从算法策略上看，也不一定是有效的。因为评价基因好坏的策略同样是经验和理解使然。遗传算法中，人为干预的参数太多，加上诸多算子的设计策略问题，甚至解的编码问题，都严重影响着最优解的效果。在算子的设计上，应用尽可能多的问题本身的特征。至于问题的编码，N维二元向量和染色体结构有着天然的联系，但却没有给出二元变量的排序策略。总之，设计的成功与否，还要看看算法的执行效果吧。

3.5　算法总结：设计

我们最初设计了爬山法，一种简单的贪心算法。尝试之后，居然能很快的求解出“还算不错”的解，于是，在接下来的工作中，改进该算法就成了一个很好的主意。熟悉它贪心的天然劣势，自然而然的会想到使用模拟退火算法进行改进。模拟退火算法是被证明了的，依概率收敛[32]于最优解的一种方法。但问题是，我们能不能在有效时间内得到最优解。在随机参数的设定上，偏于苛刻的话，那模拟退火算法就会退化成爬山法，而过于宽松的话，模拟退火算法就会出现严重的波动，又因为我们的跳出条件设置的是探测的步数，这很可能会使算法得到更差的解。再就是最复杂的遗传算法，遗传算法的复杂在于众多的算子和参数。它就是一个初始值为一个种群的模拟退火算法。所以，我们看到了变异率，以及传统遗传算法中的交配率等条件，都是为了避免遗传算法过快收敛于局部最优解。总之，遗传算法给了我们足够多的应用启发式信息和自己“好主意”的自由。那我们从下一章给出的实验结果，比较一下三种算法，看看分析的到底是不是有道理。

第四章　实验结果展示

4.1　测试用例

在本问题上，有一组常用基准测试，评估算法最好的办法就是用试验结果说话。这组基准测试的复杂度详情见下表：

表 4.1: 测试用例规模

bench	c432	c499	c880	c1355	c1908
Input	72	82	120	82	66
Gates	160	202	383	546	880
bench	c2670	c3540	c5315	c6288	c7552
Input	314	100	356	64	412
Gates	1193	1669	2307	2416	3512

我们前面已经介绍了，问题的复杂度和输入的规模成指数关系，上表的$Input$就是输入的规模。但算法中会重复计算中间节点在给定赋值的情况下的结果值。所以，中间节点数目，也就是表中的$Gate$值对效率也会有很显著的影响。

首先先看一下，三种算法从上面一组测试用例的电路文件中得出的计算结果[1]，如下表所示：

我们观察图4.1算法执行结果的比较。横坐标是文件名，按输入规模进行排序[3]。纵坐标表示在该文件计算所得结果：这里并没有列举算法在所有电路文件上的执行结果，而且增加了两列来表示HL和GA在多次计算过程中得出的最差结果。为了增加彼此间的区分度，每组数据都减去了一个固定的值，使得表格中甚至有一些负数值的出现[4]。

[1]是多次重复计算取的最优值。

[2]由于$c17$电路规模太小，我们不再罗列。

[3]输入规模相等时，按门节点数目排序

[4]比如$c880$列，就是每个值都减去400后的结果，由于HL_min最坏时出现结果为318，则其相对值就成了-82.

表 4.2: 算法实验结果[2]

.bench	c432	c499	c880	c1355	c1908
HL	198	215	423	415	970
SA	167	205	413	460	947
GA	191	212	419	464	968
.bench	c2670	c3540	c5315	c6288	c7552
HL	1338	1533	2887	2529	3886
SA	1224	1516	2587	2551	3422
GA	1259	1531	2739	2733	3457

从图4.1我们可以发现，四个最优值中，有三个最优值是用HL算法求解得到的，而只有一组解是由GA得到的。这里也顺便说一句，我们可以从表4.2中发现，其实11组最优解中，有10组是HL得到的，这绝非偶然，只有一组是GA得到的。SA则相对来说平庸了一些。当然，它存在的意义，更多的是作为一个从HL到GA的过渡。再看出现的最差结果，我们发现，所有的四组最差结果都是HL求解得到的，而GA则相对稳定，不会求解得到很差的结果。这也印证了我们前面关于HL过分依赖初始值的判断。局部最优解间可能差别较大，这导致了其结果波动太大，太不稳定。这个从HL_max以及HL_min之间巨大的空隙就很明显能够看出来，因为最差的结果也无一例外的都出现在了HL的执行结果[5]中。而从总体结果来看，GA算法表现比较差的原因，我感觉就是算子设计和参数设定的不合理导致的。但GA表现比HL要稳定的多，我们也不难从图中GA_min同GA_max中较小的空隙中发现这一点。

单纯从常规算法的时间复杂度上来看，问题是NP难的。通过重复计算统计执行时间，再从耗时上来看一看，三种算法的执行效率究竟怎么样。由于SA算法并没有表现出比HL更好的效果，这有违该算法的设计初衷，所以SA只是作为了一个由HL到GA的过渡算法进行讨论，而没有对大文件进行重复性测试。这里主要是统计了HL和GA对于所有电路文件多次执行的时间消耗，其中一些较大的电路文件执行一次时间过长，比如$c7552.bench$。所以，两种算法对它的统计次数较少，可能会稍不准确。但大部分文件，执行次数都有

[5]由于HL算法执行耗时最少，所以被执行的次数最多，而SA最无关紧要，所以执行的次数最少。理论上讲，HL出现最差结果的概率也最大，但我们忽略这一点

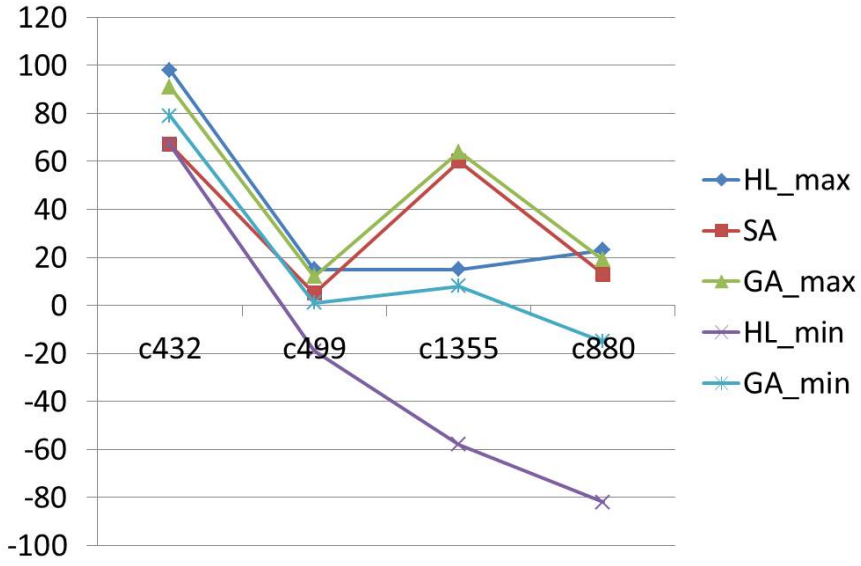

图 4.1: 算法效果对比图

十数次甚至数十次之多，应该会比较准确的展示算法的效果。

下表是三个算法统计的执行一次运算的最长[6]耗时：

表 4.3: 三种算法的统计耗时[7]

.bench	c432	c499	c880	c1355	c1908
HL	5.14	3.42	41.26	7.04	27.01
SA	12.50	5.04	73.24	12.5	43.97
GA	22.61	8.07	139.12	19.52	81.00
.bench	c2670	c3540	c5315	c6288	c7552
HL	937.39	175.31	2521.2	209.22	6884.91
SA	1381.28	262.56	3466.59	276.8	7938.79
GA	1837.63	345.82	4144.56	615.71	9304.45

备注：时间单位:s 机器配置(系统:$vm\ fedora,\ cpu:i7\ 2600$, 主存:$4G$)

整体的效果就是，HL算法速度最快，而GA算法速度最慢。这里要说明的一点是，算法的执行时间的长短是可以通过修改参数来调节的。比如GA算

[6]这里的最长耗时不是理论时间，而是统计时间，是多次运行程序得出的时间

[7]由于$c17$电路规模太小，我们不再罗列。

法就可以通过调节种群数量和遗传的代数来降低计算时间。但这样无疑会让GA的执行结果从整体上变差。

先提醒下，接下来我们会发现一件令人很诧异的事情。首先，前面不断的说，不断的提，我们处理的问题是NP难的问题，时间复杂度和电路输入的规模成指数级的关系。但接下来的图4.2却可能让我们大失所望。算法的执行时间和另外一组数据紧密相关，而和输入貌似没有任何关系。

自习看一下三条线的走势。这里只对HL算法的计算结果进行了罗列。看一看在HL算法中，执行时间到底和什么最相关，是输入结点个数？还是门的个数？这里为了让图展示的效果更明显，我们也对$c7552.bench$的数据做了一些等价的处理：在$c7522$这一点上同时压缩了三个指标。稍后我们会解释，为什么最初讨论的NP问题，变得貌似和输入的个数完全没有关系了！

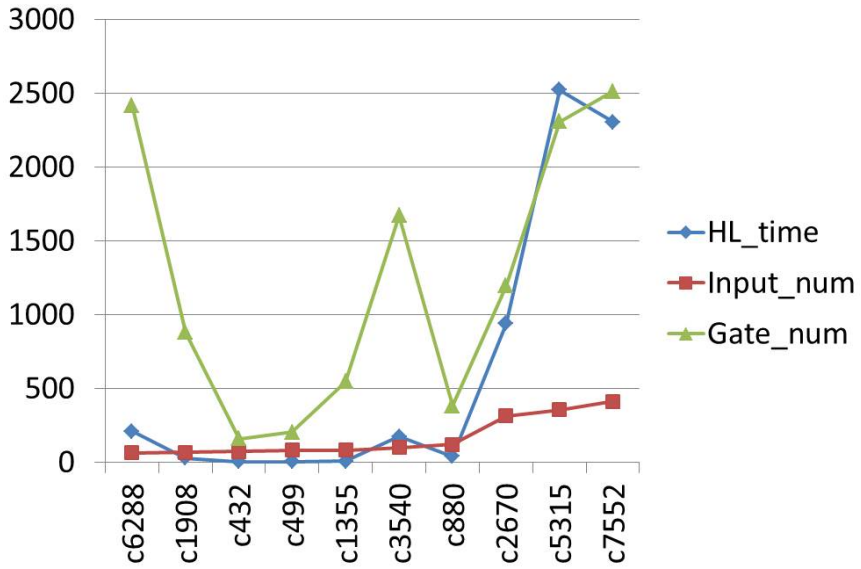

图 4.2: 时间和谁更相关？[8]

看吧！算法的执行时间和中间门节点的个数的走势大体相似，而跟输入个数的走势，则不是很一致。我们先解释下上面遇到的奇怪问题：启发式方法最根本的优点在于能求解我们传统算法不能求解的NP问题。既然要解决它，就不要按照经典算法的思路来计算了。简单说一下：我们设计的爬山法，大部分的时间，其实都花费在了重复计算$Evaluation$算子的值上，$Forte$内置的BDD

[8]我们将$c7552$的耗时和中间节点的个数分别减去了7000和1000

会花费大量的时间求解某个逻辑方程在当前的赋值下是不是为$true$；而至于输入的个数，从算法中来看，除了增加了解的长度，让逻辑方程变的更长以外，对算法的效率没有很大的影响。启发式算法和经典算法在复杂度规模计算上，还是有差别的。好吧，我想你粗略明白了。

4.2　数据比较

表4.4展示了和前面介绍的几种方法的比较结果。这里不考虑时间因素，因为大家可能由于试验时间，机器配置，平台的不同，单纯比较可能并不能得出准确的结果。同时，有些数据的准确性有待考证。再次，一些数据并没有给出准确的时间及时间单位，现在也无法考证作者的数据情况。基于以上原因，这里只有运算得出的最优结果的比较。对于一些可能有误的数据，我们也通过一些手段进行了验证，甚至包括求解了某些电路的精确的最大值[9]。

如上面我们分析的，除了几组问题数据外，爬山法的计算结果还是很有竞争力的。我们主要是使用$Forte$平台，$Forte$中集成了BDD。为了更好的试验算法的效率，我们使用了基于C的$CUDD$提供的BDD接口，用C语言重写了三个算法，算法的在执行时间上有了显著的提高。

还是横向比较本文的三种算法，首先我们发现，大部分最优结果出现在最简单的算法爬山法中。而我们对其优化而使用的模拟退火算法和遗传算法反而得出的结果较差。

对于模拟退火算法，有一个很关键的地方，就是恰当的设置随机选择的选择条件。选择条件过于宽松，那么就有很大可能向更差的结果做出选择；而如果过于苛刻，那么就又会失去它的意义，因为很可能会使其退化成爬山法而重新陷入局部最优的窘境。这里还有一些条件原因，即便把随机条件设置的很苛刻，也可能不会有爬山法一样的优化解，因为我们改变了算法框架的跳出条件。爬山法是贪心的，其跳出条件设置为遇到一个局部最优解时。而模拟退火算法的跳出条件则是执行的次数，这里的次数是很难界定的，而且可能随着初始值的不同，而有很大差异。由于不了解电路的解空间分布，导致我们在设置参数时是盲目而没有依据的。

[9]一些研究人员使用基于SMT的$solver - Z3$，运用二分法求解出了$c499$电路的精确解，结果发现$Assim$给出的结果比精确解要大。而另外一组解，$Assim$给出的结果已经超过了电路权重的上界

[10]点击这里进行下载数据lcs.ios.ac.cn/~lyj238/hillClimbing.html

表 4.4: 数据比较[10]

.bench	c432	c499	c880	c1355	c1908
HL	198	215	423	415	970
SA	167	205	413	460	947
GA	191	212	419	464	968
MAX_p	183	196	388	368	898
SIM	169	208	333	399	962
MiniSat+	274*	526*	352	578	737
PBS4	274*	526*	355	579	745
Galena	274*	526*	360	579	742
CPLEX	274*	544*	367	588	754
.bench	c2670	c3540	c5315	c6288	c7552
HL	1338	1533	2887	2557	3886
SA	1224	1516	2587	2551	3422
GA	1259	1531	2739	2733	3457
MAX_p	1161	1347	2556	2911	3556
SIM	1191	1462	2573	2647	3387
MiniSat+	963	1456	2205	2833	3156
PBS4	989	1486	2248	2833	3229
Galena	973	1496	2232	2833	3227
CPLEX	1014	1529	2386	2833	3306

对于遗传算法原因就很简单了，那就是，我们没有很好的用算子表示出电路的特性，以至于我们设计的算法在使用启发式信息时的方法是不准确甚至是错误的。将个体抽象为染色体是比较通用的做法，但是，染色体片段之间的依赖性很小，每个片段表示不同的性征。交叉遗传会使得父本的优良性状得到保持，子代个体的一些性状还可能会加强，这就会使得子代个体完成向更完善个体的进化。而电路的输入之间，片段的概念无从谈起，单个输入与单个输入之间的依赖太大，这也促使了单性生殖算子的出现。但是，单性生殖会使父代的优良性状得到保持么？这又要涉及对于好坏基因评定的策略。前面我们也介绍过，对于策略，也是受基因之间依赖性限制的。单纯的更改单个基因，更可能

获得高适应度个体而不是一定会获得高适应度个体。但并不是说两种算法和爬山法比较就一无是处。其实爬山法过于依赖初始值的缺点，显现在执行结果上就是，不同试验，得出的最优值分布很分散，差距很大。而遗传算法的结果分布则比较好，不会出现很差的值。这些是通过算法的重复执行而观察到的结果，其实也正是算法特性在执行中的反映。

在接下来的两章中，我们将分别介绍一些起初尝试使用的优化方法以及对于精确求解峰值能耗估计问题的一些想法和演示的例子，以供大家参考。

第五章　优化工作

这里叫优化工作，是因为我们曾经想通过一些前期的处理，降低算法的复杂度。但实验结果却表现的很差，这也让我们停止了这里的工作。但这项技术却是有很大的意义的，这作为我们做过的研究的一部分写到这里。

在介绍之前之前，我们先介绍一下STE和$Forte$工具：

5.1　STE(符号轨迹计算)介绍

5.1.1　轨迹符号计算

STE[11]是一种新型的基于符号模拟的模型检测方法。它将符号模拟技术的电路建模能力与时序逻辑的模型检测技术的自动分析能力结合起来，使得对于数字电路的验证效率得到了相当大的提高。所谓的符号化指的是，在轨迹公式中允许符号变量的存在。在STE中的所有模拟用例都是由符号化变量所组成的公式来索引的，对于符号化轨迹公式的一次计算，就能达到模拟所有用例计算的目的。因为对符号轨迹公式的计算覆盖了所有的模拟用例。这从很大程度上减少了模拟计算的次数。

5.1.2　轨迹计算逻辑

STE的受限时序逻辑是它其在数据路径电路验证中成功的一个关键因素，因为STE因其变得高效。轨迹公式的构造是简单的，STE的语法规则[21]如下所示：

$$
\begin{aligned}
f, g := {} & n \ is \ 0 && \text{-节点}n\text{值为0} \\
| \ & n \ is \ 1 && \text{-节点}n\text{值为1} \\
| \ & f \ and \ g && \text{-公式的交} \\
| \ & p \rightarrow g && \text{-当}P\text{为真时}f\text{被断言} \\
| \ & Nf && \text{-}f\text{在下个时刻为真}
\end{aligned}
$$

当f和g表示公式，n表示电路上的节点，P是命题公式，我们通常叫$guard$。最基本的轨迹公式是'$n\ is\ 0$'和'$n\ is\ 1$'。表示的是节点n的值为0或者值为1。and操作符表示轨迹公式的交。$P \rightarrow f$弱化子公式f，即假设P为真的时候子公式f是满足的。最后的是Nf，它表示的是在下一时间点上f为真。

$Guard$是变量能出现的唯一的地方。它们构成命题公式。这么说的话，貌似有些严格。比如我们看如下公式'$n\ is\ b$'(这里的b是一个变量)，这里的变量好像不是出现在$guard$ 里的。我们的语法上有一些微调，即允许变量和节点进行关联。

$$n\ is\ P \triangleq P \rightarrow (n\ is\ 1)\ and\ \neg P \rightarrow (n\ is\ 0) \tag{5.1}$$

在这里n表示节点，P表示命题公式。

我们定义σ序列为一个电路的轨迹。ϕ是$guard$中的变量的一组赋值，当一个序列σ满足轨迹公式f的定义如下：

$$\begin{aligned}
\phi,\sigma \models n\ is\ 0 &\triangleq \sigma(0) = \top\ or\ \sigma\ 0\ n = 0 \\
\phi,\sigma \models n\ is\ 1 &\triangleq \sigma(0) = \top\ or\ \sigma\ 0\ n = 1 \\
\phi,\sigma \models f\ and\ g &\triangleq \phi,\sigma \models f\ and\ \phi,\sigma \models g \\
\phi,\sigma \models P \rightarrow g &\triangleq \phi \models P\ implies \phi,\sigma \models f \\
\phi,\sigma \models Nf &\triangleq \phi,\sigma_1 \models f
\end{aligned}$$

这里$\phi \models P$的意思是，赋值ϕ对P中变量的赋值使命题公式P为真。

5.1.3 三元逻辑和偏序空间

这里要讲到抽象的技术。所谓抽象[17]，就是用布尔变量来符号化索引，同时，用布尔逻辑公式计算用来表示可达的抽象状态集合的结果集。这种机制就是“符号索引”[21]。所谓的时序逻辑即，所检测的STE断言性质为一个受限的时态逻辑，该逻辑仅仅能描述系统在特定有限时序逻辑关系内的性质，其描述公式的形式为$A \Rightarrow C$ 。其直观的含义就是：若给定的输入时序满

足$A(antecedent)$，则形影的输入时序会满足$C(consequence)$。给定电路M，给定断言$A \Rightarrow C$，我们就是要验证该电路的所有轨迹是否会满足该断言。由于值X的抽象性，我们只要计算出M满足的性质A的最小轨迹(相对于前面的偏序关系而言)，该公式往往被称为A的定义轨迹。只要检测到该轨迹同时满足C，那么M的所有轨迹都将满足$A \Rightarrow C$。正式由于被检测的轨迹只有一条，所以通过轨迹计算检测的性质效率非常之高。STE的核心是[11]，在抽象的格中，将电路中的每个节点赋予$\{0,\ 1,\ X\}$集合中的一个值，其中值X是表示一个未知的值或我们不关心的值。对应的电路的每个节点复制后，我们就得到了一个布尔电路状态集合的抽象。我们称其为抽象的，是因为它模糊的表示了布尔状态集合族的状态。只要我们将其中的X值用0或者1进行替换，我们就会得到布尔电路的一个具体状态。这些抽象集合就形成了一个由节点值包含的信息量为序的格[11]。为了实现基于X的数据抽象，STE提供了一种灵活的方法，我们称之为“符号索引技术”，它使得我们能够对数据进行部分抽象。这项技术充分利用了STE的偏序状态空间，被广泛的应用于数字电路验证领域，如存储器(这里的偏序关系更加的显著，同时能更好的用手动的方式建立正确的符号索引的编码，这都使得能在很大程度上体现这项技术的作用)。但这项技术的一个劣势在于，在很多的实际应用中，用户需要手动的建立正确的符号索引的编码[17][13]。对于自动化实现“符号索引技术”，有这样一些工作。在存储器验证以外的其它问题上，比如Round-robin Arbiter[34][35](ATM(Asynchronous Transfer Mode)[15]等高速网络路由器性能的关键因素）验证上发挥更好的作用。

我们将在接下来的章节中介绍我们这方面的工作，同时介绍当初我们认为它对于我们$power$峰值估计的意义。

5.2　Forte工具

STE是硬件模型检测领域一个很强大的技术，STE已经在中等到大规模工业级硬件设计的验证中展示了其强大的应用前景[16][25][23]。其在工业界所引起的影响要远远超过了在学术界。目前已经在$Intel$, $Motorola$和IBM广泛应用。比如在$Intel$公司，STE用来验证一个浮点型算数单元和一个复杂的IA指令长度的解码单元，并和$IEEE$ 754标准进行了比较[13]。同时，$Intel$还开发了集成了fl语言以及公式$Solver$的形式化验证工具$Forte$[9]。我们接下来的工

作，都主要在$Forte$工具上展开。

5.3 符号索引技术

5.3.1 符号索引技术的意义

我们知道，我们求解问题的复杂度是和输入向量的维度成指数关系的，如果我们能通过某种技术，在保留问题原来的所有约束的同时，减少原始输入的个数，那么我们的问题就会从复杂度上发生变化，如果效果好的话，能从很大程度上降低时间复杂度。

$Symbolic\ Indexing$可以处理STE逻辑公式。这项技术就是用来挖掘那些偏序的状态空间，并且减少我们的逻辑公式中的BDD变量的个数[21]。减少变量的个数，对于NP难问题的求解是至关重要的。

5.3.2 举个小例子

我们用如下的三输入的and门来解释：

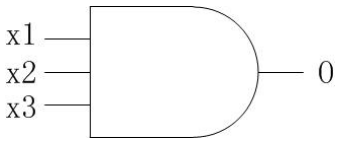

图 5.1: 3-and门

如果我们用STE做验证，那么的模型如下：

$$\begin{aligned} &\models (x_1\ is\ a)\ and\ (x_2\ is\ b)\ and\ (x_3\ is\ c) \\ &\Rightarrow (O\ is\ (a \wedge b \wedge c)) \end{aligned} \tag{5.2}$$

我们验证的原始公式的性质如下:

$$
\begin{aligned}
&\models \neg a \rightarrow (x_1\ is\ 0)\ and\ a \rightarrow (x_1\ is\ 1)\ and \\
&\quad \neg b \rightarrow (x_2\ is\ 0)\ and\ b \rightarrow (x_2\ is\ 1)\ and \\
&\quad \neg c \rightarrow (x_3\ is\ 0)\ and\ c \rightarrow (x_3\ is\ 1) \\
&\Rightarrow \\
&\quad \neg a \vee \neg b \vee \neg c \rightarrow (O\ is\ 0)\ and\ a \wedge b \wedge c \rightarrow (O\ is\ 1)
\end{aligned}
$$

在这里共使用了三个变量，每个变量代表该门的一个输入。这样做的好处是，每个变量互相独立，而且是唯一的。这样可以符号化的模拟电路的行为，看我们的门电路是不是能正常运行。

前面我们讨论了STE中的X，它是用来在验证电路性质时，减少性质中变量使用个数的手段。我们要模拟电路的几次运行状态是这样的：

$$
\begin{aligned}
(x_1\ is\ 0) &\rightarrow (O\ is\ 0) \\
(x_2\ is\ 0) &\rightarrow (O\ is\ 0) \\
(x_3\ is\ 0) &\rightarrow (O\ is\ 0) \\
(x_1\ is\ 0) \wedge (x_1\ is\ 0) \wedge (x_3\ is\ 1) &\rightarrow (O\ is\ 1)
\end{aligned}
\tag{5.3}
$$

这里表达的意思是，如果有一个输入是0，那么不管其它两个输入是什么(X)，输出结果都将是0。而只有当三个输入都为1的时候，输出才会是1。我们这里只有4个用例需要验证，而在我们的原始公式中，一共使用了三个符号变量$\{a,\ b,\ c\}$。而事实上是，只需要两个变量，就能覆盖所有的四个用例了。

符号索引技术，用布尔变量来枚举所有的用例。它的意义就是我们要用最少的变量，来覆盖我们原来的性质。再看上面介绍的$3-and$门的例子，这里一共有4个用例需要检测，我们使用变量p和q来索引这4个用例。我们可以检测如下的轨迹断言来验证所有用例：

$$
\begin{aligned}
&\models p \wedge q \rightarrow (x_1\ is\ 0)\ and \\
&\quad p \wedge \neg q \rightarrow (x_2\ is\ 0)\ and \\
&\quad \neg p \wedge q \rightarrow (x_3\ is\ 0) \\
&\quad \neg p \wedge \neg q \rightarrow (x_1\ is\ 1) \wedge (x_2\ is\ 1) \wedge (x_3\ is\ 1) \\
&\Rightarrow \\
&\quad p \vee q \rightarrow (O\ is\ 0)\ and\ \neg p \wedge \neg q \rightarrow (O\ is\ 1)
\end{aligned}
$$

如果以上的这个公式是为真的，那么我们就很确定，我们原是的公式也为真。这就是我们符号索引技术的优势，在这个$3-and$门的例子中，我们将符号变量的个数由三个降低为两个。当然，在一些实际的例子中，变量的减少程度可能会更显著，这从很大程度上降低了我们验证的时间复杂度。要时刻记得，我们的问题可是NP难的。当然还有一些工作要做，就是索引的转化，即用新的索引变量的自动化转换。

我们的目标是，将该技术用来化简$Power$模型中的门电路转换函数，减少这些转换函数中使用的输入的个数。后来我们发现，由于$Sara\ Adams$在论文[17]中提供的$automatic_build_abstraction$函数在转化索引的过程中的冗余，使我们转化后的结果不但没有改善，反而更糟糕了。即我们的输入变量的个数比转换完成后的新的索引变量的个数反而少。其中有一些原因，就是我们的电路，往往有大量的节点门，在转化的时候，节点门越多，转化的效果越差，这使得我们不得不在$Power$模型的前期优化中放弃该方法。尽管如此，我还是学到了不少东西。

第六章　进一步的工作

我们的工作主要集中在三种启发式算法的设计上。同时，为了脱离对$Forte$工具的依赖，我们尝试使用$CUDD$和C对三种算法进行了重新编写，以期达到更快的速度，同时控制对内存的使用规模。该工作已经完成，但由于时间原因，并没有对所有的测试用例进行重复性测试。同时有了一些新的想法，能借助其它工具，使得问题求解时，能从复杂度上得到显著的降低。

6.1　电路拆分

电路拆分无疑是一种很好的方法，因为我们可以很好的理解它对求解电路的复杂度的降低效果，即大电路拆分成小电路分别求解。我们举一下例子进行说明：

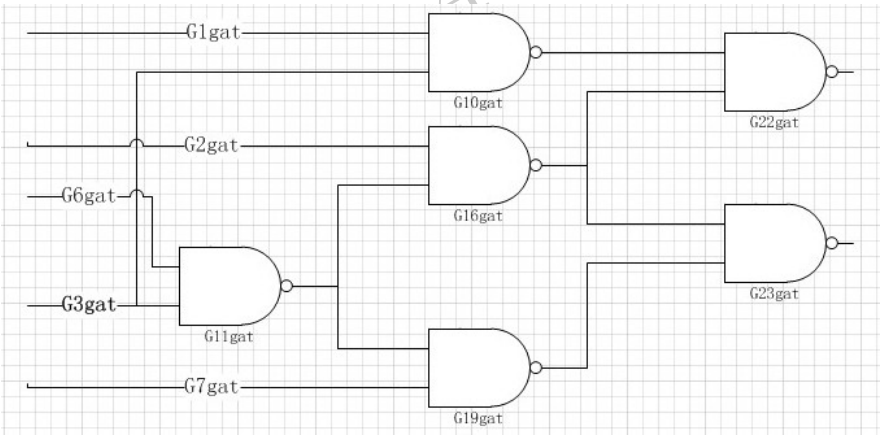

图 6.1: $c17$电路拓扑图[1]

如果想计算以上电路的峰值能耗，我们可以很容易计算出问题的复杂度规模。原始电路异或上原始电路的一份拷贝，输入的总个数为$2 \times N = 10$，则一共有$2^{10} = 1024$ 个用例需要计算。接下来首先介绍一下电路的拆分方式：从电路的输出开始，将电路的输出分成两部分。我们的目标是，通过拆分电路的输出，将电路的输入等价的拆分成两部分。这两部分必不可免会有交集，而电路

[1] 由c17.bench得到。

拆分目标就是：使电路拆分的两部分的输入尽可能相等，同时使两部分的交集尽可能的小。假设这里有三个集合，$|Set_1|$表示部分1包含的输入，$|Set_2|$表示部分2包含的输入，$|Set_3|$ 表示两个分割部分的交集输入。经过拆分之后，我们知道$|Set_1| + |Set_2| - |Set_3| = 2 \times N$，这里$|Set|$表示$Set$集合中输入的个数。我们假设$|Set_1| - |Set_3| = a$而$|Set_2| - |Set_3| = b$，同时$|Set_3| = c$。

6.2 拆分算法

接下来介绍一下电路的拆分算法，框架如下：

Algorithm 12 *Split_Operator bexpr_list*

```
max_weight ← 0;
assignment ← Initial();
\\The complicity is 2^c
while ∃unTested_val do
  \\The complicity is 2^a
  best_1 ← Solve(Set_1);
  \\The complicity is 2^b
  best_2 ← Solve(Set_2);
  cur_best ← best_1 + best_2;
  if cur_best > best then
    best ← cur_best;
    assignment ← cur_assign
  end if
end while
return assignment;
```


这里只是一个框架，给出框架的原因，是为了更清晰的计算电路拆分完成之后的复杂度规模：

$$2^c \times (2^a + 2^b) \tag{6.1}$$

以上为拆分后新的复杂度计算公式。接下来两张图展示了上图6.1拆分的结果，为的是计算拆分后的复杂度规模。

以上是电路拓扑结构的上半部分结构。因为电路只有两个输出，我们就很自然的将两个输出分成两部分，于是就得到了以上的结果图。

[2]上半部分指的是从原始拓扑图6.1中相对位置。

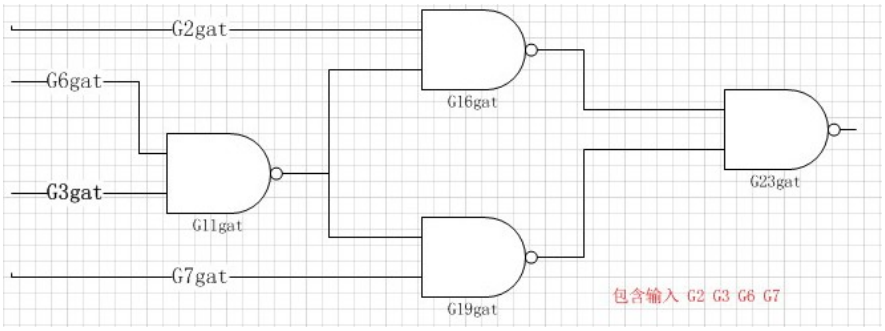

图 6.2: $c17$电路拓扑图的上半部分[2]

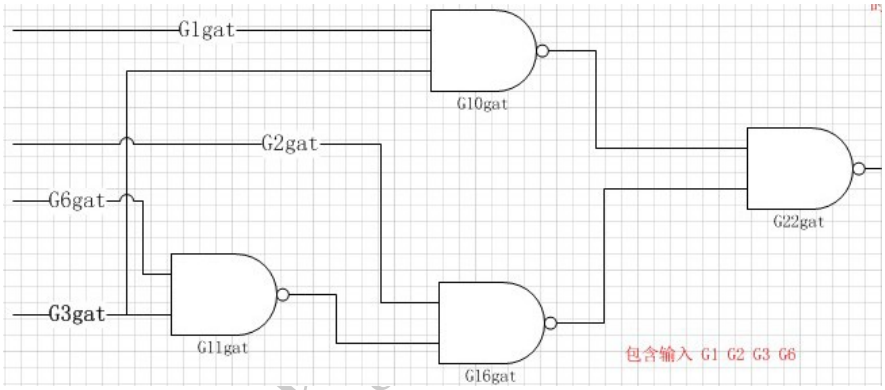

图 6.3: $c17$电路拓扑图的下半部分[3]

来看一下三个Sat的内容。因为这里是原始电路的拆分结果，新电路拆分后的拓扑也很容易得到：

$$
\begin{aligned}
Set_1 &= \{G_2, G_3, G_6, G_7\} & |Set_1| &= 4 \\
Set_1 &= \{G_1, G_2, G_6, G_6\} & |Set_2| &= 4 \\
Set_1 &= \{G_2, G_3, G_6\} & |Set_3| &= 3
\end{aligned}
$$

通过上图及公式可得原始电路中$\{a = 1, b = 1, c = 3\}$。那么在和拷贝电路异或后的新电路中，新的参数为$\{a' = 2, b' = 2, c' = 6\}$。带入复杂度计算公式就可以得到$2^6 \times (2^2 + 2^2) = 512$[4]。这样，新的结果出现了，而计算拆分电路的

[3]下半部分指的是从原始拓扑图6.1中相对位置。

[4]这里的a'表示的是新电路中集合1中的参数个数，因为包括了拷贝电路部分的输入，其它两个参看

用例只有原先电路用例的一半。即便只减少了一个参数，但作为典型的NP问题，这在问题复杂度上的降低可是很明显的。

6.3 算法缺陷

这种方法的一个缺陷是，要找到一个良好的划分结果是比较困难的，特别是当输出的规模比较大的时候。举个例子：一个有M个输出的电路，将每个输出都表示成一个输入集合的形式。这样，我们想把M个集合分成两部分，而且希望划分得到的两部分规模相当，同时交集尽可能小。更确切的说就是，让公式6.1的值最小。不要小看这个问题，集合划分也是NP难的划分问题。即我们如果想计算精确解的话，也会相当复杂。如果电路的输出比输入规模小的多，我们可以尝试划分电路。但如果输出规模与输入规模相当，甚至比输入的规模还大的时候，我们可以尝试随机划分。因为，如果这时候想计算最优划分的话，那就得不偿失了。像上面的例子6.1，输出只有两个，规模远小于输入的规模10。而试一下后就发现，确实是能从一定程度上提高效率。

同时，这种方法还有另外一个缺陷，就是算法的划分结果，有一个极限值。按照上面介绍的方法，问题最后的时间复杂度会是$2^c \times (2^a + 2^b)$，而该复杂度是有一个下限的：$2^c \times (2^a + 2^b) > Max(2^{a+c}, 2^{b+c})$。也就划分后的最好结果，一定是受到某个输出包含的输入个数限制的。还以上面的例子说明，M个输出代表的输入集合分别为$\{S_1, S_2, S_3 \ldots S_M\}$，而$Max\{|S_1|, |S_2|, |S_3| \ldots |S_M|\} = bound$[5]。结果就是，我们优化后问题的时间复杂度不会好于2^{bound}。回来看我们用启发式方法解决的那组测试用例的情况。接下来将分别给出各测试用例的$bound$值。可以很清晰的看到这样的问题：即便是算法进行了最优的划分，但大部分电路划分后结果的复杂度规模，也不是我们目前的计算机水平下的计算能力能达到的，这里只展示结果如表6.1所示。

看看这些结果：即便是我们进行了最优的划分，用上面介绍的方法，也不可能在有效的时间内计算出精确的结果。所以上面介绍的划分方法并不是针对所有电路都切实可行的。但同样可以看到的是，$c6228$的$bound$值只有64，可见对一些特殊的大电路，简单的划分算法还不算是一无是处。下面的方法要借助于一些高效的$SMT\ solver$来解决，我们这里只介绍算法的框架而不对具体的

[5]这里的Max表示从M个数字中取最大值
[6]由于$c17$电路规模太小，我们不再罗列。

表 6.1: 计算出的各电路的*lower bound*[6]

.bench	c432	c499	c880	c1355	c1908
Inputs	72	82	90	82	66
.bench	c2670	c3540	c5315	c6288	c7552
Inputs	238	100	134	64	388

实现细节进行讨论。

6.4　基于SMT的$Solver$的使用 Z_3

Z_3是微软公司开发的一款基于SMT的求解器，效率很高。如算法6.2介绍的，计算划分后子电路的值，可以依靠其来求解。当输入规模小于100时，其求解效率很高。但当输入规模略有提高之后，其效率下降过于明显，求解变得不太可能。所以，我们尝试将电路重复划分成Z_3能够求解的规模，然后借助Z_3来求解精确解[7]。我们知道算法6.2有一个缺点就是，当一次划分之后，复杂度并没有降低到我们能求解的规模，那该怎么办？

6.5　回溯法和SMT $Solver$的使用

这里我们也提供一种解决办法，即应用回溯法的框架来拆分和求解，同时为了提高性能设计一些必要的剪枝策略。我们接下来简单介绍一下算法的思想和几个必要的剪枝策略：我们按什么回溯？

Algorithm 13 *Backtracking_Level inputs*

```
while ∃unCounted_input do
    Policy2(input);
end while
Order(input); Order by the weight
return inputs;
```

这里计算的目的是，给定所有*inputs* 一个序，作为在赋值过程中回溯的层次。我们只是假定这么做是有效的，因为我们通常是这么理解问题的。而至

[7]这里，研究人员进行了对c499的测试，他采用二分法，可以用Z_3很快的求得精确解，我们不用它处理大电路，因为输入增多之后，它的效率下降太明显

于效果，在没有试验结果的情况下，我们只能说，好吧，这只是我们的一种策略。同时，关于回溯的条件，我们设计了两个：其一就是，所有的$inputs$都被赋值或者所有的f_i都已求解出具体值[8]。此时确定是否更新最优解和对应的赋值。另外一种情况是，即使并不是所有的$inputs$都给出了赋值，但是在当前赋值下，已经求解出的权重和，加上剩余未求解出的所有门的权重和[9]，依然没有目前的最优解好的时候我们选择回溯。因为很显然，继续计算下去已经没有任何意义了。

我们知道，即便是剪枝，复杂度也是有极限的，我们并没有去尝试这种方法，因为对于像$c7522$那样的大电路，这样的剪枝算法对它的求解起到的效果肯定是不那么显著的。也就是说，即便是有好的剪枝策略，我们也不可能在有效时间内计算出其精确最优解。因为我们算法本身就没有真正的解决时间复杂度上的问题。所以，在目前计算机的计算能力下，我们依然不可能求解得到如此大的电路的峰值能耗的精确解。

[8]非X值

[9]我们求解的即，在当前给定的部分$input$赋值下能达到的权重的一个$upper\ bound$，如果上界都没有当前的优化解大，显然，继续对$input$进行赋值已经没意义了。

第七章　总结

本文主要介绍了电路峰值能耗问题的一般解法，并在Srinivas Devadas的能耗模型基础上，设计了三种启发式的算法，同时用一组benchmark测试了算法的效果。因为在该能耗模型上描述的目标函数，是输入的强函数，如果要通过仿真计算电路峰值能耗的精确解，在电路规模上升之后，将是一件不可能完成的事情。于是想到用启发式的搜索算法在该模型下，计算电路的优化解。

其中爬山法是尝试求解该问题设计的第一个启发式算法，框架和算子都比较简单，而且能在较短时间内给出benchmark中每个电路文件的解。有了这样成功的尝试，稍后就有了优化爬山法的想法。进而为了克服爬山法贪心策略带来的局部性缺陷。于是开始在爬山法的基础上设计和实现模拟退火算法。为了尝试更好的优化，在算法上最大化对启发式信息的利用程度，开始尝试设计参数和算子都更加复杂的，但控制更加灵活的遗传算法。同时根据电路峰值功耗问题本身的性质，对遗传算法的框架进行了一些细微的调整，并设计了更贴近问题性质的算子。测试结果显示，爬山法无论是从最后的计算结果，还是从计算时间的消耗上，都比较有优势。但由于其算法框架本身的局部性缺陷，使之过于依赖于初始值。表现在算法的执行上就是，算法计算的权重值波动太大，所得的计算结果中所有的最差结果也都是由爬山法得到的。而模拟退火算法由于只有一种结束条件，而对于结束条件的设置，没有找到更贴近问题的策略，使得其并没有像最初设计的那样，得到比爬山法更好的解。我猜测的原因是，权重值在我们设计的邻域中，变化频繁，但整体上峰值与峰值之间的差距并不大。这样，即便是模拟退火算法能跳出一个局部邻域，一种情况是很快进入了另一个差别不大的峰值领域，或者在一个较大峰值的邻域中，到达峰值之前就结束并跳出了。至于遗传算法，并没有得出更好的解，同时又是三个算法中最耗时的一个。这样的设计结果有悖我们最初的设计意图。究其原因，一方面，我们对于解的编码形式，可能不适合用遗传算法计算；另一方面原因，就是各个算子设计上的问题。特别是遗传算子及变异算子的设计。其子代的产生规则，并没有使父代的“优良性状”得到保持。但遗传算法也并不是一无是处，和爬山法相比，遗传算法的优化解分布要均衡很多，即尽管没有得出最好的解，但所有解的差距并不大，整体分布要比爬山法好得多。

通过以上三个算法的设计，使我对问题的认识有深入了一步。在组内的讨论中，先后尝试了脱离$Forte$平台的限制，用$C+CUDD$重写了三个算法。同时还尝试了电路拆分的方法，尽管最后被认定，即便进行了最优的划分，对于该benchmark中的大电路仍然是不可求解的。同时还参看了一些基于SMT的公式求解器，讨论了一些可能可行的方法。

附录 A　c17.bench原始文件

combinational logic example "c17"

♯ ——————————————————————————-

♯

♯ total number of lines in the netlist 13

♯

♯ lines from primary input gates 5

♯ lines from primary output gates 2

♯ lines from interior gate outputs 4

♯

♯ avg_fanin = 2.00, max_fanin = 2

♯ avg_fanout = 1.33, max_fanout = 2

♯

$INPUT(G1gat)$

$INPUT(G2gat)$

$INPUT(G3gat)$

$INPUT(G6gat)$

$INPUT(G7gat)$

$OUTPUT(G22gat)$

$OUTPUT(G23gat)$

$G10gat = nand(G1gat, G3gat)$

$G11gat = nand(G3gat, G6gat)$

$G16gat = nand(G2gat, G11gat)$

$G19gat = nand(G11gat, G7gat)$

$G22gat = nand(G10gat, G16gat)$

$G23gat = nand(G16gat, G19gat)$

附录 B　c17.bench模型

new combinational logic example "c17"

♯ ——————————————————————————-

♯

♯ total number of lines in the netlist 34

♯

♯ lines from primary input gates 10

♯ lines from primary output gates 6

♯ lines from interior gate outputs 12

♯

♯ avg_fanin = 2.00, max_fanin = 2

♯ avg_fanout = 1.33, max_fanout = 2

♯

$INPUT(G1gat)$

$INPUT(G2gat)$

$INPUT(G3gat)$

$INPUT(G6gat)$

$INPUT(G7gat)$

$INPUT(*DUP*G1gat)$

$INPUT(*DUP*G2gat)$

$INPUT(*DUP*G3gat)$

$INPUT(*DUP*G6gat)$

$INPUT(*DUP*G7gat)$

$OUTPUT(NewG10gat)$

$OUTPUT(NewG11gat)$

$OUTPUT(NewG16gat)$

$OUTPUT(NewG19gat)$

$OUTPUT(NewG22gat)$

$OUTPUT(NewG23gat)$

$$
\begin{aligned}
G10gat &= nand(G1gat, G3gat)\\
G11gat &= nand(G3gat, G6gat)\\
G16gat &= nand(G2gat, G11gat)\\
G19gat &= nand(G11gat, G7gat)\\
G22gat &= nand(G10gat, G16gat)\\
G23gat &= nand(G16gat, G19gat)\\
*DUP*G10gat &= nand(*DUP*G1gat, *DUP*G3gat)\\
*DUP*G11gat &= nand(*DUP*G3gat, *DUP*G6gat)\\
*DUP*G16gat &= nand(*DUP*G2gat, *DUP*G11gat)\\
*DUP*G19gat &= nand(*DUP*G11gat, *DUP*G7gat)\\
*DUP*G22gat &= nand(*DUP*G10gat, *DUP*G16gat)\\
*DUP*G23gat &= nand(*DUP*G16gat, *DUP*G19gat)\\
NewG10gat &= xor(G10gat, *DUP*G10gat)\\
NewG11gat &= xor(G10gat, *DUP*G11gat)\\
NewG16gat &= xor(G10gat, *DUP*G16gat)\\
NewG19gat &= xor(G10gat, *DUP*G19gat)\\
NewG22gat &= xor(G10gat, *DUP*G22gat)\\
NewG23gat &= xor(G10gat, *DUP*G23gat)
\end{aligned}
$$

参考文献

[1] Kirkpatrick S, Gelatt C.D, Vecchi M.P. *Optimization by Simulated Annealing.* Number 671-680. Science, 1983.

[2] Akbari, Ziarati. A multilevel evolutionary algorithm for optimizing numerical functions. *Proceedings of IJIEC 2*, pages 419–430, 2010.

[3] Assim Sagahyroon, Fadi A.Aloul. Using sat-based techniques in power estimation. *Microelectronics Journal*, pages 706–715, 2007.

[4] Boros, Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 2002.

[5] Chuan-Yu Wang, Kaushik Roy. Maximum power estimation for cmos circuits using deterministic and statistical approaches. *VLSI'98 Very Large Scale Integertion Systems*, pages 134–140, 1998.

[6] Daniel Kahneman, Amos Tversky, Paul Slovic, eds. *Judgment under Uncertainty: Heuristics and Biases.* Cambridge University Press, 1982.

[7] Fei Li, Lei He. Maximum current estimation considering power gating. *ISPD'01 Proceedings*, pages 106–111, 2001.

[8] Fuxi.Zhu. Artificial intelligence based tutorial - edition 2. *Tsinghua University Press*, pages 30–35, 2011.

[9] Intel. *Forte from Intel.* http://www.intel.com/software/products/ opensource/tools1/verification.

[10] Kirti Sikri Desai. *EDA Innovation through Merger and Acquisitions.* EDA Cafe. Retrieved March 23, 2010.

[11] Koen Claessen, Jan-Willem Roorda. An introduction to symbolic trajectory evaluation. *SFM'06 Proceedings of the 6th international conference on*

Formal Methods for the Design of Computer, Communication, and Software Systems, pages 56–77, 2006.

[12] Michael S.Hsiao. Peak power estimation using genetic spot optimization for large vlsi circuits. *Proceedings of Design, Automation and Test in Europe Conference and Exibition*, 1999.

[13] M.Pandey, R.Raimi, R.E.Bryant, M.S.Abadir. Design automation conference. *Design Automation Conference*, pages 167–172, 1997.

[14] M.R.Garey, D.S.Johnson. *A Guide to the Theory of NP-Completeness.* Computers and Intractability, 1979.

[15] P.Curzon. The formal verification of an atm network. *13th Annual ACM Symposium on Principles of Distriuted Computing*, page 392, 1994.

[16] R.E.Bryant. A methodology for hardware verification based on logic simulation. *Proceeding Journal of the ACM*, pages 299–328.

[17] Sara Adams, Magnus Bjork, Tom Melham. Automatic abstraction in symbolic trajectory evaluation. *FMCAD'07 Proceedings of the Formal Methods in Computer Aided Design*, pages 137–135, 2007.

[18] Seger, C.J.H., R.E. Formal verification by symbolic evaluation of partially ordered trajectories. *Formal Methods in System Design*, pages 147–189, 1995.

[19] Srinivas Devadas, Kurt Keutzer, Jacob White. Estimation of power dissipation in cmos combinational circuits using boolean function manipulation. *Computer Aided Design*, pages 373–383, 1992.

[20] S.Thompson, P.Packan, M.Bohr.MOS scaling. Transistor challenges for the 21st century. *Intel Tchnology Journal*, 1998.

[21] Thomas F.Melham, Robert B.Jones. Abstraction by symbolic indexing transformation. *FMCAD'02 Proceedings of the Formal Methods in Computer Aided Design*, pages 1–18, 2002.

[22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to algorithms - edition 2.* Number 841-864. The MIT Press, 2001.

[23] T.Schubert. High level fromal verification of next-generation microprocessors. *Proceedings of the 40th conference on defign automation*, pages 1–6, 2003.

[24] Vose, Michael D. *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press.

[25] W.Dally, B.Towels. Route packets, not wires: on-chip interconnection network. *Proceedings of Design Automation Conference 2001*, pages 684–689, 2001.

[26] Wenxiu. Zhang, Yi. Liang. *Mathematical foundation of Genetic Algorithms.* Xi'an Jiaotong University Press, 2000.

[27] Whitley, Darrell. A genetic algorithm tutorial. *Statistics and Computing*, pages 65–85, 1994.

[28] wikipedia. *Genetic Algorithm.* http://en.wikipedia.org/wiki/Genetic_ algorithm.

[29] wikipedia. *Heuristic Approach.* http://en.wikipedia.org/wiki/Heuristic.

[30] wikipedia. *Hill Climbing Algorithm.* http://en.wikipedia.org/wiki/Hill_ climbing.

[31] wikipedia. *Pseudo-Boolean Function.* http://en.wikipedia.org/wiki/Pseudo-Boolean _function.

[32] wikipedia. *Simulated Annealing Algorithm.* http://en.wikipedia.org/wiki/ Simulated_annealing.

[33] wikipedia. *Very large scale integration.* http://en.wikipedia.org/wiki/ Very-large-scale_integration.

[34] Y.Li, N.Zeng, X.Song. Enhanced symbolic simulation of a round-robin arbiter. *Proceedings of International Conference on Computer Design*, pages 102–107, 2011.

[35] Y.Li, N.Zeng, X.Song. Combining symmetry reduction with generalized symbolic trajectory evaluation. *Journal of Computer*, 2012.

发表文章目录

[1] Y.Li, B.Yu and X.Song. A Hill Climbing Approach to Optimization on Pseudo-Boolean Functions and Its Application to Power Estimation. On submitting, 2013.

[2] Y.Li, B.Yu and X.Song. A SMT Approach to Optimization on Pseudo-Boolean Functions and Its Application to Power Estimation. On submitting, 2013.

简　　历

基本情况

于宝新，男，河北省保定市易县人，1987 年 10 月出生，未婚，中国科学院软件研究院在读硕士研究生。

教育状况

2006 年 9 月至 2010 年 6 月，吉林大学软件学院，本科，专业：软件工程。

2010 年 9 月至 2013 年 7 月，中国科学院软件研究所，硕士研究生，专业：计算机技术。

工作经历

无。

研究兴趣

数据挖掘，信息检索，自然语言处理，分布式。

联系方式

通讯地址：北京市 8718 信箱，中科院软件所计算机科学国家重点实验室

邮编：100190

E-mail: yubaoxin@hotmail.com

致　　谢

值此论文完成之际，谨在此向多年来给予我关心和帮助的老师、同学、朋友和家人表示衷心的感谢!

……

谨把本文献给我最敬爱的父亲!