DOI:10.16451/j.cnki.issn1003-6059.2014.09.012

VLSI 标准单元布局问题的增强型混合遗传模拟退火算法*

陈雄峰[1] 吴景岚[1,2] 朱文兴[2]

[1](闽江学院 计算机科学系 福州 350108)
[2](福州大学 离散数学与理论计算机科学研究中心 福州 350003)

摘 要 提出有效处理百万个 VLSI 标准单元布局问题的混合遗传模拟退火算法. 首先采用小规模种群、动态更新种群和交叉局部化策略,并协调全局与局部搜索,使遗传算法可处理超大规模标准单元布局问题. 然后为进一步提高算法进化效率和布局结果质量,将爬山和模拟退火方法引入遗传算法框架及其算子内部流程,设计高效的线网 – 循环交叉算子和局部搜索算法. 标准单元阵列布局侧重使用爬山法,非阵列布局侧重使用模拟退火方法. Peko suite3、Peko suite4 和 ISPD04 标准测试电路的实验结果表明,该算法可在合理运行时间内有效提高布局结果质量.

关键词 混合遗传算法,模拟退火,标准单元布局,线网 – 循环交叉算子,局部搜索
中图法分类号 TP 18

An Enhanced Hybrid Genetic Simulated Annealing Algorithm for VLSI Standard Cell Placement

CHEN Xiong-Feng[1], WU Jing-Lan[1,2], ZHU Wen-Xing[2]
[1](*Department of Computer Science*, *Minjiang University*, *Fuzhou* 350108)
[2](*Center for Discrete Mathematics and Theoretical Computer Science*, *Fuzhou University*, *Fuzhou* 350003)

ABSTRACT

A hybrid genetic simulated annealing algorithm is presented for solving the problem of VLSI standard cell placement with up to millions of cells. Firstly, to make genetic algorithm be capable of handling very large scale of standard cell placement, the strategies of small size population, dynamic updating population, and crossover localization are adopted, and the global search and local search of genetic algorithm are coordinated. Then, by introducing hill climbing (HC) and simulated annealing (SA) into the framework of genetic algorithm and the internal procedure of its operators, an effective crossover operator named Net Cycle Crossover and local search algorithms for the placement problem are designed to further improve the evolutionary efficiency of the algorithm and the quality of its placement results. In the algorithm procedure, HC method and SA method focus on array placement and non-array placement respectively. The experimental results on Peko suite3, Peko suite4 and ISPD04 benchmark circuits show

* 国家自然科学基金项目(No. 61170308)资助
收稿日期:2013 – 11 – 23;修回日期:2014 – 04 – 07
作者简介 陈雄峰,男,1969 年生,硕士,副教授,主要研究方向为智能计算. E-mail:chenxf2000126@126.com. 吴景岚,女,1969 年生,博士研究生,主要研究方向为组合优化算法及其在超大规模集成电路物理设计中的应用. 朱文兴(通讯作者),男,1968 年生,博士,教授,主要研究方向为最优化理论与算法、超大规模集成电路物理设计中的优化算法. E-mail:wxzhu@fzu.edu.cn.

that the proposed algorithm can handle array and non-array placements with 10,000 ~ 1,600,000 cells and 10,000 ~ 210,000 cells respectively, and can effectively improve the quality of placement results in a reasonable running time.

Key Words Hybrid Genetic Algorithm, Simulated Annealing, Standard Cell Placement, Net Cycle Crossover, Local Search

1 引 言

求解各类大规模乃至超大规模 NP 难组合优化问题的混合型启发式算法是当前该领域的一个研究热点[1-4]. 未采用针对性策略的情况下,遗传和模拟退火算法可有效处理组合优化问题的规模通常小于 2 万[5-12]. 文献[2]采用针对性策略,使混合型遗传算法可处理 1 万~20 万规模旅行商问题,并在合理运行时间内获得部分超过当时最优的问题解.

超大规模集成电路(Very Large Scale Integration, VLSI)标准单元指具有相等高度、相等或不等宽度的电路单元. 标准单元等宽和不等宽意味着大小相同和不相同,布局模式分别为阵列和非阵列[13-15]. 合法的标准单元布局要求一个电路的所有单元都被放置在已划定的布局行上且互不重叠. 标准单元阵列布局的所有单元大小相同,任意两个单元交换位置不影响整个布局的合法性;非阵列布局则不然,合法布局进行单元位置交换后就不一定合法. 标准单元阵列布局时任意一个单元排列都可对应一个合法布局,问题可行域总是连通的;而非阵列布局时有部分单元排列不能对应合法布局,问题具有非连通可行域. 因此,本文将标准单元布局问题细分为阵列和非阵列布局问题,分别对二者采用不同的针对性策略,以便能更为有效地提高算法的优化性能.

目前主要的标准单元布局算法有解析算法、启发式算法和划分算法等,其中,启发式算法包括遗传算法和模拟退火算法等. 遗传算法同时混合其他进化方法以及结合局部搜索是目前的主流方法之一[5,9-13,15]. 但迄今为止包括[5],[9]~[13],[15]在内的文献表明,此类算法对标准单元布局的问题基本上没有采用针对性策略,通常仅能处理几十至几千个单元规模电路,且布局结果质量均没有超过现有最新的解析算法[16-17].

基于种群和交叉机制的遗传算法具有较强的全局搜索能力但局部搜索能力不足,基于单个解个体和顺序邻域搜索的爬山和模拟退火方法可获得一定范围内高质量的优解但全局搜索能力不足[1,3,18-19]. 爬山法的优势是搜索效率高,但要求好的可行解位于当前可行解附近才能继续搜索最优解,应用于可行域连通的问题时才能体现其优势. 模拟退火方法搜索过程灵活,可接受不好的解,能自行跳出局部最优解,适用于具有非连通可行域的复杂问题,其缺点是计算时间长且效率相对较低. 本文首先在种群和交叉算子等方面采用针对性策略,在有效利用遗传算法全局搜索优势的基础上,将爬山和模拟退火方法引入遗传算法框架及其主要算子内部流程,同时针对阵列布局问题可行域总是连通的特性侧重使用爬山法,针对非阵列布局问题具有非连通可行域的特性侧重使用模拟退火方法,以增强其局部搜索能力,进而提高算法的进化效率和布局结果质量.

2 问题表示与算法基础

2.1 问题定义

布局前的电路设计已确定标准单元个数 n 以及每个单元 v_i 的高度 h 和宽度 w_i,线网个数 m 以及每个线网 e_j 所连接的单元或固定端点(Pins),矩形布局区域的高度 H 和宽度 W,在布局区域上划分的布局行数 s 和布局行高度 h_r 等相关信息,其中,一个单元或固定端点可属于多个线网,一个线网可连接多个单元或固定端点.

布局时只能将单元放置在某一布局行上,要求单元之间不能重叠,确定单元中心位置的坐标(x_i, y_i). 通常以最小化布局的所有线网半周长之和(Half-Perimeter Wire-Length, HPWL)为主要优化目标,因为优化 HPWL 可间接优化可布线性、时延、功耗等其他指标[13,16,20],HPWL 减小幅度达 2% 即可视为显著提高布局质量[20]. 一个线网 e_j 的半周长是指包围其连接的所有单元或固定端点最小矩形的半周长,可表示为

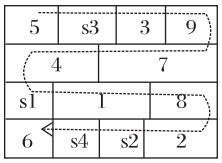
$$HPWL(e_j) = \max_{s,t\in e_j}|x_s - x_t| + \max_{s,t\in e_j}|y_s - y_t| = \max_{s\in e_j} x_s - \min_{s\in e_j} x_s + \max_{s\in e_j} y_s - \min_{s\in e_j} y_s. \quad (1)$$

一个布局 P 的所有线网半周长之和可表示为

$$HPWL(P)=\sum_{1\leqslant j\leqslant m}HPWL(e_j)$$
$$=\sum_{1\leqslant j\leqslant m}\left(\max_{s\in e_j}x_s-\min_{s\in e_j}x_s+\max_{s\in e_j}y_s-\min_{s\in e_j}y_s\right).\quad(2)$$

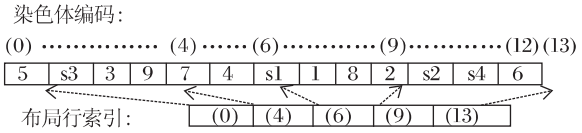
2.2　布局表示与染色体编码

一个二维布局如图1(a)所示,其中1,2,…,9为标准单元编号,s1、s2、s3、s4为标准单元之间的间隔区域即布线单元编号.为能高效实现交叉操作,遗传算法中的个体染色体编码通常采用一维形式[1,2,5,13,19],如图1(b)所示.以蛇形排列可使得二维布局中物理位置靠近,特别是蛇形弯曲处的单元在一维染色体中也能彼此靠近,利于提高交叉进化效果.图1(b)中"布局行索引"用于存放二维布局每一行在一维染色体上的开始位置(下标从0开始),使得单元编号在二维布局所处位置的行、列下标与其在一维染色体所处位置的下标之间可进行快速换算或索引查找.这样实现算法可直接用一维而无须用二维数据结构存放二维布局,大大降低算法的空间和时间复杂度.采用这一数据结构是实现算法可处理超大规模布局问题的有效策略之一.



(a)2 维布局

(a)2D placement



(b)1 维染色体编码

(b)1D chromosome code

图1　布局与染色体编码示意图

Fig. 1　Sketch map of placement and chromosome code

2.3　目标函数和适应值函数

一个布局 P 的 $HPWL(P)$ 越小表示其布局质量越高.算法中模拟退火方法的目标函数直接设定为 $HPWL(P)$,布局的适应值函数定义为 $HPWL(P)$ 的反比例函数,表示为

$$fitness(P)=\left[HPWL_{Theoretical\ Opt.}\mid HPWL_{Expected\ Opt.}\right]/HPWL(P),\quad(3)$$

其中,$HPWL_{Theoretical\ Opt.}$ 为某一电路实例理论上最优布局的 $HPWL$,没有理论最优布局的可自行选取一个期望最优值 $HPWL_{Expected\ Opt.}$.这一适应值函数可直观反映当前布局质量与理论最优或期望最优布局质量的比例关系.$HPWL_{Theoretical\ Opt.}$ 和 $HPWL_{Expected\ Opt.}$ 二者取值只要保证不同规模布局的适应值可稳定在相同区间如(0.0,1.5),以便根据当前布局结果质量水平统一协调控制算法的全局与局部搜索,此外没有其他特别限制.

3　算法设计

3.1　算法框架

算法以混合型遗传算法为框架,针对标准单元阵列和非阵列布局的不同特性,分别侧重使用爬山法和模拟退火方法,同时随机使用两种不同策略的局部搜索,具体描述如下.

算法1　Framework of the proposed Hybrid Genetic Simulated Annealing Algorithm for Standard Cell Placement

输入　附有布局所需信息的电路

输出　优化布局的所有电路单元位置坐标

begin

Initialize placement population;

repeat

Choice parents;

Net Cycle Crossover;

Select offspring by Hill Climbing method;

Select individuals for local search, randomly use step 1 or step 2 with probability P_{Net_Win}:

step 1　Net Local Search, accept neighbor by SA method (*SA_Accept_Less*());

step 2　Window Local Search, accept neighbor by Hill Climbing method for array placement or SA method (*SA_Accept_Less*()) for non-array placement;

Update population except the *best* individual if *population lost diversity*;

until　Satisfy termination condition;

Output optimal placement;

end // 算法结束

其中,"Net Cycle Crossover"为线网-循环交叉

算子(Net Cycle Crossover, Net_CX),详见 3.3 节;"Net Local Search"为线网单元聚集模拟退火局部搜索,详见3.4.1 节;"Window Local Search"为窗口穷尽局部搜索,详见 3.4.2 节;*SA_Accept_Less*( ) 为模拟退火方法接受函数,详见 3.2 节. 交叉算子和局部搜索内部流程中处理阵列和非阵列布局时,也分别侧重使用爬山法和模拟退火方法,详见 3.2 ~ 3.4 节相关介绍. 针对非阵列布局侧重使用模拟退火方法的设计策略体现尤为突出的有效性,详见 4.3 节的实验分析. 下面介绍使遗传算法可处理大规模乃至超大规模问题的针对性策略.

1)种群. 采用小规模种群(5 ~ 10 个),而后使用动态更新种群及下文 2)和 4)所述双亲与后代选择和交叉局部化策略,以保持小规模种群的多样性,提高小规模种群的进化性能. 动态更新种群是指种群一旦失去多样性,如"所有 $fitness(P_i)/fitness(P_{best}) > 99.5\%$",就更新种群个体,更新时用新初始个体替换除当前最优外的其他所有个体. 选用文献[16]第一阶段非线性规划方法生成的非法布局,应用就近原则将其合法化,即将其每个单元放置在已划定的布局行上且单元之间互不重叠,以此作为组成初始种群的主要个体. 与随机生成个体组成的初始种群相比,这种初始种群决定更好的初始搜索区域. 需进一步优化初始搜索区域时加入 1 ~ 2 个经文献[16]或文献[17]解析算法合法化或优化的布局个体.

2)双亲与后代选择. 在每一代开始时,对种群个体重新进行随机排列,依次逐个选择当前个体为母亲个体,其循环相邻的下一个个体为父亲个体. 后代选择采用竞争性和爬山法结合策略,即让一对双亲进行多次不同的交叉生成多个(20 ~ 40)不同的孩子个体,以第一次出现优于母亲个体或最优的孩子个体为候选后代个体,如果候选后代个体优于母亲个体则替换母亲个体,否则不替换. 这一策略可使小规模种群保持更长时间的多样性和寻优能力.

3)全局与局部搜索. 随着当前布局结果质量的提高而逐步加大进行局部搜索的个体范围,以协调算法的全局和局部搜索. 以概率 P_{Net_Win} 随机使用两种局部搜索,实验表明处理阵列布局时,P_{Net_Win} 取值 15% 左右,整体上性能最好;处理非阵列布局时,P_{Net_Win} 取值 50% ~ 90%,性能都较好.

4)交叉局部化. 采用文献[2]交叉局部化的思想,每次进行交叉的单元个数限定为远小于布局单元的总个数,具体设定见 3.3 节相关内容. 这样一方面大大降低交叉算子的时间复杂度,利于遗传算法的高效实现,另一方面也利于保持交叉生成后代个体之间的差异性.

3.2 模拟退火方法要素

模拟退火方法的要素包括初始状态选择、目标函数、邻域函数、状态接受(转移)概率、初始温度设定与降温方案和算法终止条件等. 邻域函数生成新状态的接受概率通常使用 Metropolis 准则,即接受概率 $p = \exp(-\Delta E/T)$,其中 ΔE 为目标函数的增加值,当 $\Delta E \leqslant 0$ 或 $\Delta E < 0$ 时邻域新状态全部接受,$\Delta E > 0$ 或 $\Delta E \geqslant 0$ 时接受概率与 ΔE 的大小和当前温度参数 T 有关,当 $T \approx 0$ 时模拟退火方法退化成为爬山法. 这里介绍本文算法框架及其主要算子内部所使用的接受函数、初始温度设定与降温方案,其他要素需要依据问题不同方面特征进行更有针对性设计,详见3.3 和 3.4 节相关介绍.

1)接受函数. 算法中模拟退火方法接受函数根据不同邻域函数所生成候选状态的不同特点细分为两种,具体描述如下.

算法 2 Acceptance function [*SA_Accept_Less*(*old_ind*, *new_ind*, *CUR_TEM*) | *SA_Accept_Less* Equal(*old_ind*, *new_ind*, *CUR_TEM*)]

输入 当前个体 *old_ind*,后代或邻域个体 *new_ind*,当前温度 *CUR_TEM*

输出 被接受的新当前个体 *old_ind*

```
// 算法框架上后代选择时使用"<",交叉、局部搜索内部算法流程使用"<="
begin
  if (HPWL(new_ind) [<|<=] HPWL(old _ind))
    old_ind = new_ind;
  else
    if (rand() < exp(-(HPWL(new_ind) -
      HPWL(old_ind))/CUR_TEM))
      old_ind = new_ind;
    end if
  end if
  return old_ind;
end
```

2)初始温度. 初始温度设定与初始布局质量和单元长度均有关[6,18]. 算法中初始温度 T_0 通过实验设定为:(1)处理阵列布局时,$T_0 = -\min CellWidth/\ln(0.5)$;(2)处理非阵列布局时,$T_0 = -\min CellWidth/\ln(0.2)$,其中 $\min CellWidth$ 为最小单元宽度.

3)降温方案. 算法中退火过程的降温方案也是通过实验确定. 非阵列布局以一定时间间隔内,当前最优布局中"接受 $\Delta HPWL \geqslant 0$ 次数 / 接受 $\Delta HPWL < 0$ 次数"的比例值作为降温判断条件,相对更为精确;阵列布局则因模拟退火方法作用较小而难以确定合理的比例值,直接使用"遗传进化代数"作为降温判断条件. 为提高收敛速度,在退火过程的中间和后期阶段每次降幅应保持稳定有效,因此分阶段逐步减小降温系数,具体方案如下:

(1) $CUR\_TEM = 0.93CUR\_TEM$, $CUR\_TEM > 0.6T_0$;

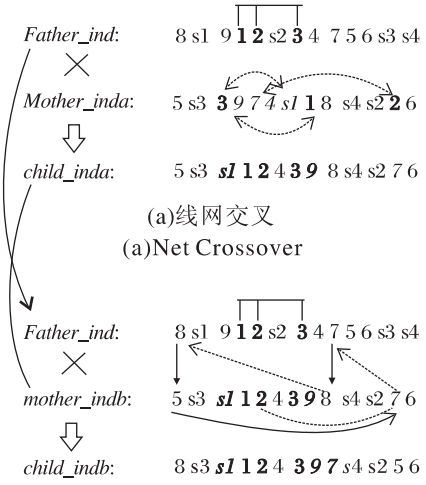
(2) $CUR\_TEM = 0.89CUR\_TEM$, $0.2T_0 < CUR\_TEM \leqslant 0.6T_0$;

(3) $CUR\_TEM = 0.84CUR\_TEM$, $CUR\_TEM \leqslant 0.2T_0$.

3.3　线网-循环交叉算子

遗传算法的要素包括染色体编码、适应值估算、双亲与后代选择、交叉算子和变异算子等,其中交叉是其搜索问题解空间的主要算子. 目前标准单元布局问题的主要遗传交叉方法有部分匹配交叉(Partially-Matched Crossover, PMX)、循环交叉(Cycle Crossover, CX)和相邻交叉(Neighbor Crossover, NX)等,其中最为常用的是 PMX,CX 也有相对较好的性能[5,9,13,21]. 针对 PMX、CX 和 NX 进行实验比较分析,结果表明,要获得更高质量的布局结果,特别是超大规模电路的布局结果,必须根据问题及其优化目标的特征,有针对性地设计具有更好收敛性的交叉算子. 采用直接交换、交叉片段内单元不相邻以及片段长度局部化的设计思想可有效提高交叉算子的收敛性能. 本文依此思想以及阵列和非阵列布局的不同特性,设计一种线网与循环结合的交叉算子,简称线网-循环交叉算子(Net Cycle Crossover, Net_CX).

Net_CX 交叉过程如图 2 所示. 图 2 中 *father_ind*、*mother_ind* 分别指参与交叉的父亲个体和母亲个体染色体,*child_ind* 为交叉后生成的孩子个体染色体. 图 2(a)中 *father_ind* 上 1、2、3 为将要交叉到 *mother_inda* 的一个线网所连接的单元. 以 *father_ind* 上 1、2、3 所在位置为双亲进行交叉的对应位置,*mother_inda* 上 9、7、s1 分别为其对应位置上的单元. 交叉时采用与 PMX 相同机制,对应位置单元直接交换,同时解决交叉后单元重复或缺失的问题. 交叉过程可简化为直接在 *mother_inda* 上进行单元交换,即其对应位置上的单元 9、7、s1 分别与单元 1、2、3 直接进行位置交换,如图 2(a)双箭头虚线所示.



(a)线网交叉
(a)Net Crossover

(b)长度为 3 的循环交叉
(b)Cycle crossover with a length of 3

图 2　Net_CX 示意图
Fig. 2　Net_CX diagram

处理阵列布局时,仅进行上述线网交叉(Net Crossover, NetX);处理非阵列布局时,NetX 过程中每进行一对单元交换,马上就从被交换单元新位置开始限定长度的 CX,如图 2(b)所示为单元 2 与 7 交换后从单元 7 新位置开始进行长度为 3 的 CX. 图 2(b)中 *father_ind* 与图 2(a)中相同,图 2(a)单元交换生成的孩子个体 *child_inda* 直接作为 CX 的母亲个体 *mother_indb*,CX 生成的孩子个体 *child_indb* 又直接作为下一次交换单元的母亲个体 *mother_inda*,如此交替进行 NetX 的单元交换和 CX 直到完成一个线网交叉,而后进行下一个线网的交叉. NetX 每次单元交换后再进行 CX 是处理非阵列布局问题的针对性策略,因为单元交换后的非阵列布局不一定合法,被交换单元的新位置不一定合理,如图 2(a) *child_inda* 中单元 7 远离交叉前的相邻单元 4、9. 而非阵列布局的合法性判断又过于耗时,每次单元交换都进行布局合法性判断,这在时间上是不可行的. 交换后再进行限定长度的 CX,以较小的时间代价为每个被交换单元寻找到相对更为合理的新位置,如图 2(b) *child_indb* 中单元 7 相对靠近交叉前的相邻单元 4、9. 如此可大大增加交叉后获得合法非阵列布局的可能性,从而提高交叉的收敛性能和进化效率. 其有效性实验结果详见 4.2 节. Net_CX 算法具体描述如下.

算法 3　*Net_Cycle_Crossover*(*mother_ind*, *father_ind*, *crossover_netnumber*, *CUR_TEM*)

输入　母亲个体 *mother_ind*，父亲个体 *father_ind*，一次交叉的线网个数 *crossover_netnumber*，当前温度 *CUR_TEM*

输出　*mother_ind* 和 *father_ind* 交叉生成的孩子个体 *child_ind*

```
begin
  net_crossed 置 = EMPTY, child_ind = old_ind = mother_ind;
  for (repeat = 1  to crossover_netnumber)
     current_net = Randomly Select from Adjoin (net_crossed 置) or All First ;
     // 当前线网中单元逐个进行直接交换，而后非阵列布局从被交换单元开始循环交叉
    for (each cell C_i in current_net)
       new_ind = Swap_CellPosition(old_ind, father_ind, C_i); // 如图2(a)
       // 阵列布局使用爬山法，非阵列布局使用模拟退火方法
      if (placement style is array)
        if (HPWL(new_ind) < HPWL(child_ind))
          child_ind = new_ind;
        end if// 爬山法
      else
         child_ind = SA_Accept_LessEqual(child_ind,new_ind, CUR_TEM);//模拟退火方法
         //非阵列布局时从被交换单元开始继续循环交叉，以使其获得更合理的位置
         C_cycle =the cell which be swapped to C_i;
         old_ind = new_ind;// 交换后的新个体作为新的母亲个体
         new_ind =Cycle_Crossover(old_ind, father_ind, C_cycle, cycle_length); // 如图2(b)
         child_ind =SA_Accept_LessEqual(child_ind, new_ind, CUR_TEM);
      end if
        old_ind = new_ind; // 交换或循环交叉后的新个体作为新的母亲个体
    end for
  end for
  if (child_ind = Legalize_Placement(child_ind) is legal)
     return child_ind;
  else
     return NULL;
  end if
end
```

算法 3 中可连续进行多个线网的交叉. 尽可能选择相互有连接的多个线网，如此孩子个体可更有效继承由若干个相互连接线网组成的部分优良布局. 连续交叉的线网个数 *crossover_netnumber* 和 CX 的循环长度 *cycle_length* 可通过实验确定. 处理阵列布局时 *crossover_netnumber* 取值 15 ~ 150；处理非阵列布局时 *crossover_netnumber* 取值 1 ~ 8，母亲个体质量越低取值越大. *cycle_length* 取值 2 ~ 2 000，电路规模越大取值范围越大. 在其取值范围内随着布局结果质量提高而逐步减小，以逐步增强 Net_CX 的局部搜索能力. 实验表明这一取值方案可使 Net_CX 具有相对平衡的全局和局部搜索效果，既有较高的搜索效率又不易陷入局部优解.

3.4　局部搜索

3.4.1　线网单元聚集模拟退火局部搜索

线网单元聚集模拟退火局部搜索不仅在搜索后选择新状态且在其邻域函数 *Net_CellGathertoMid*( ) 内部处理流程中使用模拟退火方法. *Net_CellGathertoMid*( ) 根据标准单元布局问题和算法优化目标的特点进行有针对性的设计，以贪心思想把一个线网所连接的单元往其所有单元的中间位置聚集，聚集过程如图 3 所示，其中第 1、2 行分别表示单元个数为奇、偶数两种情况. 算法具体描述如下.

算法 4　*Net_Local_Search*(*current_ind*, *SA_netnumber noimprovment_count*, *CUR_TEM*)

输入　当前个体 *current_ind*，一次邻域搜索所使用的线网个数 *SA_netnumber*，搜索过程中未能获得更好邻域个体的次数 *noimprovment_count*，当前温度 *CUR_TEM*

输出　被接受为新当前个体的邻域个体 *current_ind*

```
begin
  neighbor_ind =old_ind =current_ind;
  count = 0;
  // 连续进行多次邻域搜索，直到获得更好的可能性较小时为止
  while (count < noimprovment_count)
   // 以模拟退火方法进行邻域搜索，生成当前个体的一个邻域个体
   for (repeat =1 to SA_netnumber)
      current_net =Randomly Select from All Nets;
      // 一个线网的单元往所有单元的中间位置聚集，如图 3 所示
    new_ind =Net_CellGathertoMid(old_ind, current_net,CUR_TEM) with acceptance
```

```
                function SA_Accept_LessEqual();
      old_ind = new_ind;
    end for
    if (neighbor_ind = Legalize_Placement(new_
      ind) is legal)
     if (neighbor_ind is not better than current_
       ind) count = count + 1; end if
     if (neighbor_ind is better than best_ind)
       count = 0; end if // 重新开始计数
     // 与此同时实现算法 1 中的后代选择
     current_ind = SA_Accept_Less(current_ind,
     neighbor_ind, CUR_TEM);
    end if
   end while
   return current_ind;
end
```

算法 4 中每一次局部搜索时可进行连续多次的邻域搜索,直到未能获得优于当前最优布局且邻域搜索 *noimprovment_count* 次未能获得优于当前布局的邻域个体为止. 每次邻域搜索特别是处理阵列布局时又可连续进行多个线网(*SA_netnumber*)的单元聚集,通过实验观察随机选择线网效果更好. 在实验算法中取 *noimprovment_count* = 3,阵列和非阵列布局的 *SA_netnumber* 取值分别为 15 和 1.

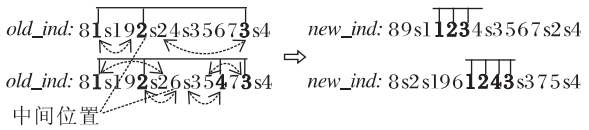


图 3　线网单元聚集示意图

Fig. 3　Sketch map of net cells gathering

3.4.2　窗口穷尽局部搜索

窗口穷尽局部搜索时在二维布局上随机选择窗口,而后在窗口内进行穷尽搜索. 实验表明4个~5个单元的穷尽搜索效率最高,模拟退火方法搜索邻域范围不宜过大,因此窗口大小设定为 4 × 5,搜索又细分为列和行穷尽搜索两个阶段. 算法具体描述如下.

算法5　*Win_Local_Search* (*current_ind*, *win_number*, *CUR_TEM*)

输入　当前个体 *current_ind*,一次局部搜索所使用的窗口个数 *win_number*,当前温度 *CUR_TEM*

输出　被接受为新当前个体的邻域个体 *current_ind*

```
begin
  old_ind = current _ind;
  // 连续进行多个窗口穷尽搜索
  for (repeat =1 to win_number)
    begin_row,end_row,begin_col,end_col =
    RandLocate_SearchWindow(current_ind);
    // 先对所有列,后对所有行分别进行穷尽搜索
    for (search_direction = "COL" to "ROW")
    // 阵列布局用爬山法,非阵列布局用模拟退火
    方法
     if (placement style is array)
       neighbor_ind = FullSeach_AllEachbyEach
                      (search_direction,
                      begin_row,
                      end_row,begin_col,end_col,
                      old_ind) with acceptance
                      function HillClimbing_Accept
                      ();
      if (HPWL(neighbor_ind) <
        HPWL(current_ind)) // 爬山法
       current_ind = neighbor_ind;
       // 实现算法 1 中的后代选择
      end if
     else
       new_ind = FullSeach_AllEachbyEach(search_
                 direction, begin_row, end_row,
                 begin_col, end_col, old_ind) with
                 acceptance function SA_Accept_
                 LessEqual();
      if (neighbor_ind = Legalize_Placement(new_
        ind) is legal )
        // 实现算法 1 中的后代选择
        current_ind = SA_Accept_Less(current_ind,
        neighbor_ind, CUR_TEM);
      end if
     end if
     old_ind = current_ind; // 列搜索的结果作
     为行搜索的开始
    end for
  end for
  return current_ind;
end
```

算法 5 中每一次局部搜索可连续搜索多个窗口,窗口个数 *win_number* 可通过实验确定. 为适应阵列和非阵列布局的不同特性,以及爬山法和模拟退火方法的不同特点,阵列和非阵列布局分别采用

不同的 *win_number* 设定方案. 处理阵列布局时 *win_number* 设定为约每 1 万个单元 3 个~10 个;处理非阵列布局时 *win_number* 设定为 1 个~3 个. 当前最优布局个体的 *win_number* 随着电路规模和布局结果质量的提高逐步增多;其他布局个体的 *win_number* 固定为最大值,目的是使其能尽快提高布局质量,以利于交叉时生成更高质量的孩子布局个体.

4 实验与结果分析

算法使用 C 语言编程实现,测试环境为 Windows XP, Intel Core2 CPU 2.10GHz, RAM 2.0GB. 阵列布局使用 UCLA VLSI CAD LAB 公布的标准测试电路实例 Peko suite3 和 Peko suite4 [22],非阵列布局使用 ISPD04 ibm [23],其中 ISPD04 与 Peko suite3 规模完全相同,Peko suite4 规模是 Peko suite3 的 10 倍. 由 Peko suite3 和 Peko suite4 可知其理论上的 *HPWL* 最优值 [22],详见表 1、表 2 中 "*Theoretical Opt. HPWL*",而 ISPD04 则未知. 为统一二者适应值函数的取值范围以便于算法的实现,依据 2.3 节式(3)适应值函数定义,参照 Peko suite3 理论上 *HPWL* 最优值和二者初始布局 *HPWL*,为 ISPD04 的每个测试例子设定一个期望获得的 *HPWL* 最优值,即

$$\begin{aligned} & HPWL_{Expected\ Opt.} \\ \approx & HPWL(P_{ISPD04\ Initial})* fitness(P_{Peko\ Initial}) \\ = & HPWL(P_{ISPD04\ Initial})* \frac{HPWL_{Theoretical\ Opt.}}{HPWL(P_{Peko\ Initial})}, \end{aligned}$$

取值详见表 1 中 "*Expected Opt. HPWL*".

4.1 算法性能

算法设定种群规模 $N_p = 5$,交叉概率 $P_c = 0.8$,无变异算子,结束条件为 "100 分钟时间间隔 *HPWL* 进化减小量小于 0.01×10^6(阵列布局)或 0.001×10^6(非阵列布局)",其他参数设置参见上文中相应章节. 所有测试例子均可在合理的运行时间内(少于 24h)获得明显优于使用相同初始非法布局的文献[16]和文献[17]解析算法布局结果,实验结果如表 1 和表 2 所列,其中,"*Analytical algorithm opt. HPWL*" 是文献[16]和文献[17]解析算法所获得的 5 个不同布局结果 *HPWL* 的平均值;"*HPWL*" 是本文算法所获得的 5 个不同布局结果 *HPWL* 的平均值;"*HPWLimpr.*" 是 "*HPWL*" 较之于 "*Analytical algorithm opt. HPWL*" 的减小程度.

表 1 Peko suite3 和 ISPD04 ibm 测试结果

Table 1 Test results of Peko suite3 and ISPD04 ibm

阵列						非阵列					
电路	单元个数	*Theoretical opt. HPWL* ($\times 10^6$)	*Analytical algorithm opt. HPWL* ($\times 10^6$)	*HPWL* ($\times 10^6$)	*HPWL impr.* /%	电路	单元个数	*Expected opt. HPWL* ($\times 10^6$)	*Analytical algorithm opt. HPWL* ($\times 10^6$)	*HPWL* ($\times 10^6$)	*HPWL impr.* /%
Peko01	12506	0.822	1.01	0.90	10.89	ibm01	12506	1.290	1.619	1.585	2.10
Peko02	19342	1.27	1.59	1.44	9.43	ibm02	19342	2.775	3.458	3.394	1.85
Peko03	22853	1.51	1.90	1.72	9.47	ibm03	22853	3.560	4.470	4.379	2.04
Peko04	27220	1.76	2.21	1.98	10.07	ibm04	27220	4.360	5.472	5.357	2.07
Peko05	28146	1.95	2.46	2.29	6.91	ibm05	28146	7.510	9.357	9.225	1.41
Peko06	32332	2.07	2.57	2.33	9.39	ibm06	32332	3.827	4.825	4.707	2.45
Peko07	45639	2.89	3.58	3.23	9.78	ibm07	45639	6.423	8.041	7.915	1.57
Peko08	51023	3.15	3.93	3.60	8.40	ibm08	51023	6.878	8.609	8.403	2.38
Peko09	53110	3.65	4.51	4.19	7.10	ibm09	53110	7.235	9.022	8.886	1.51
Peko10	68685	4.75	5.89	5.49	8.79	ibm10	68685	13.537	16.998	16.652	2.04
Peko11	70152	4.72	5.88	5.38	8.50	ibm11	70152	10.809	13.572	13.291	2.07
Peko12	70439	5.02	6.20	5.74	7.42	ibm12	70439	17.120	21.549	21.107	2.05
Peko13	83709	5.89	7.37	6.77	8.14	ibm13	83709	13.017	16.263	16.015	1.52
Peko14	147088	9.03	11.55	10.93	5.37	ibm14	147088	24.864	31.211	30.634	1.85
Peko15	161187	11.60	14.59	13.89	4.78	ibm15	161187	28.704	37.395	36.613	2.09
Peko16	182980	12.50	15.71	14.94	4.90	ibm16	182980	33.427	41.842	41.144	1.67
Peko17	184750	13.50	16.87	16.11	5.10	ibm17	184752	49.010	60.883	59.230	2.72
Peko18	210341	13.20	16.61	15.75	5.18	ibm18	210341	31.296	39.318	38.645	1.71

表 2　Peko suite 4 中 3 个不同规模测试实例的测试结果

Table 2　Test results of 3 benchmarks with different scale in Peko suite4

电路	单元个数	*Theoretical opt. HPWL* ($\times 10^6$)	*Analytical algorithm opt. HPWL* ($\times 10^6$)	*HPWL* ($\times 10^6$)	*HPWL impr.* /%
Peko07	456390	28.9	34.91	33.50	4.04
Peko13	837090	58.9	72.19	69.67	3.49
Peko15	1611870	116.0	147.65	141.79	3.97

算法可有效处理阵列和非阵列布局电路的实例均达到超大规模,分别为 1 万~160 万和 1 万~21 万个单元,"HPWLimpr." 分别为 3.49% ~10.89% 和 1.41% ~2.72%. 实验说明,2.2 节所述一维存储结构有效降低算法的空间复杂度,为算法实现可处理超大规模问题提供必要的基础条件. 小规模种群、动态更新种群和交叉局部化策略是降低算法时间和空间复杂度,进而扩大处理问题规模的主要有效策略. 根据问题的特点进行针对性设计是改进交叉算子和局部搜索性能,提高布局结果质量的有效途径.

尽管侧重使用模拟退火方法的设计策略大幅提高了针对非阵列布局的算法收敛性(参见 4.3 节),但与针对阵列布局相比,算法性能仍有较为明显的差距. 主要原因是后代选择时要进行非阵列布局合法性判断,舍弃非法的后代个体. 实验观察交叉算子 Net_CX 中针对非阵列布局的 CX 机制可有效提高获得合法后代个体的概率,但不能降低非阵列布局合法性判断的复杂度. 以降低非阵列布局合法性判断的复杂度为目标,设计更为高效的交叉算子和局部搜索,将进一步提高针对非阵列布局算法性能.

4.2　交叉算子 Net_CX 收敛性能

为进一步验证 Net_CX 针对性设计策略的有效性,使用与 4.1 节相同参数和结束条件,执行算法全过程,分别使用 Net_CX、NetX 和 CX 对不同规模电路实例 ISPD04 中 ibm01、ibm04、ibm07、ibm10、ibm13、ibm16、ibm18 的收敛性进行测试. 所有测试结果均体现出 Net_CX 和 NetX 收敛性大大优于 CX,Net_CX 收敛性优于 NetX. 图 4 和图 5 分别为对 ibm01、ibm18 电路实例的收敛性比较.

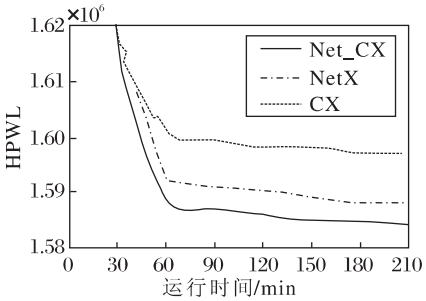


图 4　对 ibm01 电路的交叉算子收敛性比较

Fig. 4　Convergence comparison of crossover operators for circuit ibm01

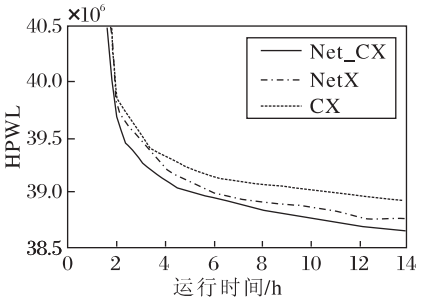


图 5　对 ibm18 电路的交叉算子收敛性比较

Fig. 5　Convergence comparison of crossover operators for circuit ibm18

4.3　模拟退火方法对算法收敛性的作用

为进一步验证针对非阵列布局侧重使用模拟退火(SA)方法设计策略的有效性,使用与 4.1 节相同参数、初始种群和算法基本框架,分别采用不使用 SA、仅算法框架使用 SA 和本文算法,针对 Peko suite 3(阵列)和 ISPD04 ibm(非阵列)不同规模电路实例 ibm01、ibm04、ibm07、ibm10、ibm13、ibm16、ibm18,进行算法收敛性测试. 运行时间与 4.1 节测试相同,测试结果如表 3 所列,其中,"$\Delta HPWL$" 为各算法布局结果 *HPWL* 较之于 "*Analytical algorithm opt. HPWL*"(如表 1 所列)的减小量,"*ΔHPWL impr.*" 为两种使用 SA 算法较之于不使用 SA 算法的 "$\Delta HPWL$" 的提高程度,可表示为

$$\Delta HPWL = HPWL_{Analytical\ algorithm\ opt.} - HPWL, \quad (4)$$

$$\Delta HPWL\ impr. = \frac{(\Delta HPWL - \Delta HPWL_{Without\ SA})}{\Delta HPWL_{Without\ SA}} \times 100\%. \quad (5)$$

测试结果表明,处理阵列和非阵列布局时 SA 对算法收敛性的作用明显不同,具体分析如下.

1)处理阵列布局时,SA 对使用相对较低质量初始种群的 1 万~10 万个单元规模电路实例有一定的作用,"*ΔHPWL impr.*" 为 3.57% ~8.70%,而对其中使用更高质量初始种群的 10 万个单元以上规模电路实例几乎没有作用. 处理 4 万~10 万个单元相对较大规模电路时,仅算法框架使用 SA 的算法的 "*ΔHPWL impr.*" 为 3.57% ~4.35%,主要算子内部流程使用 SA 的本文算法 "*ΔHPWL impr.*" 为 7.14% ~8.70%,二者

一一对比，本文算法“$\Delta HPWL\ impr.$”高出4% 左右.

2）处理非阵列布局时，SA 可大幅度提高算法收敛性能，而且电路规模越大提高幅度越大. 仅算法框架使用 SA 的算法的“$\Delta HPWL\ impr.$”为 16.00% ~ 138.82%，主要算子内部流程使用 SA 的本文算法的“$\Delta HPWL\ impr.$”为36.00% ~194.51%，二者一一对比，本文算法“$\Delta HPWL\ impr.$”高出约20% ~56%，同样是电路规模越大高出越多.

表 3 不同 SA 使用强度的 3 种算法收敛性比较

Table 3 Convergence comparison of 3 algorithms with different SA intensity

阵列				非阵列			
电路	不使用 SA $\Delta HPWL$ ($\times 10^6$)	仅算法框架使用 SA $\Delta HPWL\ impr.$ /%	本文算法 $\Delta HPWL\ impr.$ /%	电路	不使用 SA $\Delta HPWL$ ($\times 10^6$)	仅算法框架使用 SA $\Delta HPWL\ impr.$ /%	本文算法 $\Delta HPWL\ impr.$ /%
Peko01	0.105	4.76	4.76	ibm01	0.025	16.00	36.00
Peko04	0.22	4.55	4.55	ibm04	0.069	37.68	63.77
Peko07	0.23	4.35	8.70	ibm07	0.070	50.00	80.00
Peko10	0.37	4.05	8.11	ibm10	0.165	73.94	109.70
Peko13	0.56	3.57	7.14	ibm13	0.109	84.16	125.69
Peko16	0.86	0.00	0.00	ibm16	0.237	138.82	194.51
Peko18	0.86	0.00	0.00	ibm18	0.238	126.47	182.77
平均值		3.04	4.75			75.3	113.21

5 结 束 语

本文提出一种可有效处理百万个 VLSI 标准单元布局问题的混合遗传模拟退火算法. 首先采用小规模种群和交叉局部化策略有效降低时间和空间复杂度，使用动态更新种群等策略提高小规模种群的进化性能，协调全局和局部搜索以节省运行时间，使遗传算法可处理超大规模标准单元布局问题. 而后在有效利用遗传算法全局搜索优势的基础上，将爬山和模拟退火方法引入遗传算法框架及其算子内部流程，以增强其局部搜索能力. 针对标准单元布局问题及其优化目标的二维性质，设计新型高效的线网-循环交叉算子 Net_CX 和局部搜索算法. 与此同时针对阵列和非阵列布局的不同特性，分别侧重使用爬山法和模拟退火方法，有效提高算法的收敛性能，进而提高布局结果质量. 实验结果表明，本文算法可在合理运行时间内获得明显优于现有解析算法的布局结果. 与不使用和仅算法框架使用模拟退火方法的遗传算法相比，本文算法能更为有效地提高布局结果质量，针对非阵列布局侧重使用模拟退火方法的设计策略的有效性尤为明显. 这些针对性策略或思想也可为其他类型大规模组合优化问题的启发式算法研究提供借鉴.

后续研究的重点将通过引入其他启发式方法如禁忌搜索等，设计更为高效的交叉、局部搜索等主要算子，以使算法能获得更高质量的非阵列布局结果. 继续扩大算法处理标准单元布局问题规模，以及使之可以处理混合单元布局问题也是后续主要研究方向.

参 考 文 献

[1] Tamilarasi A, Anantha Kumar T. An Enhanced Genetic Algorithm with Simulated Annealing for Job-Shop Scheduling. International Journal of Engineering, Science and Technology, 2010, 2(1): 144 - 151

[2] Nagata Y, Kobayashi S. A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem. INFORMS Journal on Computing, 2013, 25(2): 346 - 363

[3] Baghel M, Agrawal S, Silakari S. Survey of Metaheuristic Algorithms for Combinatorial Optimization. International Journal of Computer Applications, 2012, 58(19): 21 - 31

[4] Guo W Z, Chen G L, Xiong N, *et al*. Hybrid Particle Swarm Optimization Algorithm for VLSI Circuit Partitioning. Journal of Software, 2011, 22(5): 833 - 842 (in Chinese)
(郭文忠，陈国龙，Xiong Naixu，等. 求解 VLSI 电路划分问题的混合粒子群优化算法. 软件学报，2011，22(5)：833 - 842)

[5] Kaur J, Kaur M. A Survey on Various Genetic Approaches for Standard Cell Placement. International Journal of Computer Technology and Applications, 2013, 4(3): 533 - 536

[6] Suman B, Kumar P. A Survey of Simulated Annealing as a Tool for Single and Multiobjective Optimization. Journal of the Operational Research Society, 2006, 57(10): 1143 - 1160

[7] Chen J L, Zhu W X, Ali M M. A Hybrid Simulated Annealing Al-

gorithm for Nonslicing VLSI Floorplanning. IEEE Trans on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2011, 41(4): 544 – 553

[8] Chen J L, Zhu W X, Peng Z. A Heuristic Algorithm for the Strip Packing Problem. Journal of Heuristics, 2012, 18(4): 677 – 697

[9] Bunglowala A, Singhi B, Verma A. Multi-objective Optimization of Standard Cell Placement Using Memetic Algorithm. International Journal of Computer Applications, 2011, 19(7): 31 – 34

[10] Tang M L, Yao X. A Memetic Algorithm for VLSI Floorplanning. IEEE Trans on Systems, Man, and Cybernetics, Part B: Cybernetics, 2007, 37(1): 62 – 69

[11] Markov I L, Jin H, Kim M C. Progress and Challenges in VLSI Placement Research // Proc of the IEEE/ACM International Conference on Computer-Aided Design. San Jose, USA, 2012: 275 – 282

[12] Chopra V, Singh A. Multi-objective Genetic Algorithm for Testing MCNC FPGA Benchmark Circuits. International Journal of Computer Science and Communication Engineering, 2013, 2(1): 74 – 78

[13] Wang L T, Chang Y W, Cheng K T. Electronic Design Automation: Synthesis, Verification, and Test. San Francisco, USA: Morgan Kaufmann, 2009

[14] Khawam S, Nousias I, Milward M, *et al*. The Reconfigurable Instruction Cell Array. IEEE Trans on Very Large Scale Integration (VLSI) Systems, 2008, 16(1): 75 – 85

[15] Li K. Placement & Constrained Placement Optimization in VLSI Physical Design. Ph. D. Dissertation. Chengdu, China: University of Electronic Science and Technology of China, 2010 (in Chinese)
(李 康. VLSI 物理设计中布局及有约束的布局优化. 博士学位论文. 成都:电子科技大学, 2010)

[16] Chen J L, Zhu W X. An Analytical Placer for VLSI Standard Cell Placement. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 2012, 31(8): 1208 – 1221

[17] Chang Y W, Jiang Z W, Chen T C. Essential Issues in Analytical Placement Algorithms. IPSJ Trans on System LSI Design Methodology, 2009, 2: 145 – 166

[18] Pattanaik S, Bhoi S P, Mohanty R. Simulated Annealing Based Placement Algorithms and Research Challenges: A Survey. Journal of Global Research in Computer Science, 2012, 3(6): 33 – 37

[19] Garcia-Martínez C, Lozano M. Local Search Based on Genetic Algorithms // Siarry P, Michalewicz Z, eds. Advances in Metaheuristics for Hard Optimization. Berlin, Germany: Springer, 2008: 199 – 221

[20] Kim M C, Viswanathan N, Alpert C J, *et al*. MAPLE: Multilevel Adaptive Placement for Mixed-size Designs // Proc of the ACM International Symposium on Physical Design. Napa, USA, 2012: 193 – 200

[21] Subbaraj P, Sankar S S, Anand S. Parallel Genetic Algorithm for VLSI Standard Cell Placement // Proc of the International Conference on Advances in Computing, Control, and Telecommunication Technologies. Trivandrum, India, 2009: 80 – 84

[22] Chang C C, Xie M. PEKO Suite (Placement Example with Known Optimal Wire Length) [EB/OL]. [2013 – 03 – 20]. http://cadlab.cs.ucla.edu/~pubbench/placement/dw.htm

[23] Viswanathan N, Chu C. ISPD04 IBM Standard Cell Benchmarks with Pads [EB/OL]. [2013 – 03 – 20]. http://vlsicad.eecs.umich.edu/BK/Slots/cache/www.public.iastate.edu/~nataraj/ISPD04_Bench.html