CHAPTER 7 REVIEW OF ITERATIVE ALGORITHMS FOR PLACEMENT

Jing Lee

Department of Electronic Engineering

Southern Taiwan University of Technology

Tainan, Taiwan 701, R.O.C.

Email: leejing@mail.stut.edu.tw

Placement problem has been proven to be **NP hard**. It is impractical to perform an exhaustive search to find the optimal solution for a large circuit. Thus, algorithms based on heuristic rationales are commonly employed. An ideal approach is to get an initial solution using a constructive algorithm and then uses an iterative algorithm to improve it. So, it is desirable to develop better improvement algorithms. In this chapter, a brief review of iterative algorithms is given.

7.1 General Iterative Improvement Algorithm

Considering any optimization problem, suppose one wish to find a solution that minimizes the cost function E( ) subjected to certain constraints. Solutions that satisfy the constraints are called **feasible solutions** and the feasible solution with minimum cost is called an **optimal solution**.

Many different iterative placement techniques exist. Even though they differ substantially, they all share the same underlay structure and have four main phases: **INITIAL_SOLUTION**, **PERTURB**, **ACCEPT**, and **ADAPT**. The generic form of iterative improvement algorithms can be represented as follows:

```
procedure Generic_Iterative( )
S ← INITIAL_SOLUTION( )
Initialize heuristic parameters
    repeat
        repeat
            New_S ← PERTURB(S)
            if  ACCEPT(New_S, S) then S ← New_S endif
        until "time to adapt parameters"
        ADAPT(parameters)
    until "terminating criterion"
return(S)
end Generic_Iterative
```

7.1.1 INITIAL SOLUTION( )

The initial solution may be any feasible solution. Generally, it is either generated through some randomizing mechanism, (i.e., a random feasible solution is used) or it is generated using some other heuristic for the problem being solved. Many reports [Han76, Nah86, Ros90] indicated that better final solutions are obtained when one starts with a good initial solution.

7.1.2 PERTURB(S)

This is a function that generates a new feasible solution from the current solution. There are many perturbation functions in literature.

7.1.2.1 Interchange Methods

Interchange methods select some components and then change their positions. In **single movement**, an object is selected randomly and moved to a random location in S. For example, consider an initial linear permutation, $<v_1, v_2, v_3, v_4, v_5>$ and v_4 is selected to move to position 1, a new permutation, $<v_4, v_1, v_2, v_3, v_5>$, is obtained. Single movement method is discussed in [Nah86].

In **pairwise interchange** each component is selected in turn to be the primary component and is exchanged on i trial basis with every other component. All n(n-1)/2 possible pairs of components are trial interchange in a cycle. Placement systems using pairwise interchange are discussed in [Han76, Sch76, Ios83, Nah86].

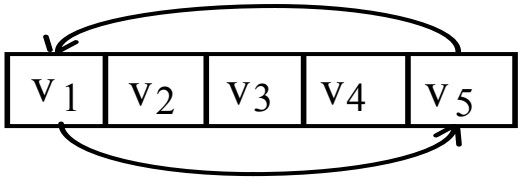

Fig. 7.1 Pairwise interchange.

Neighborhood interchange is similar to pairwise interchange; however, the primary component is interchanged only with components in its vicinity. The vicinity (distance or the number if components included) in a parameter which can be set by the user. Placement systems using neighborhood interchange are discussed in [Han76, Ios83].

In the **cycle of length λ method**, components are selected and changed in cyclic. Fig. 7.2 shown an example of λ = 3. Obviously, this is a nature extension of pairwise interchange to the case of λ components. This method is discussed in [Han76, Nah86]

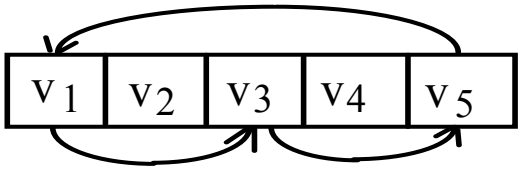

Fig. 7.2 Cycle of length λ = 3.

More complicated form of interchange were reported in [Coh86] and [Kli89]. Cohoon and Paris use a genetic paradigm to derive a selection mechanism based on random mutation in biological systems. In their method, each element in a new solution is derived by combining the features of two or more elemints in the previous solution. Kling and Banerjee's method [Kli89] is quite similar to Cohoon and Paris's. However, they derive a new solution from parts of the previous solution. Both two algorithms will be introduced more detail in the later.

7.1.2.2 Force-Directed Methods:

Force-directed interchange uses a force analog to select components as well as determine the positions to which the components should be moved [Han76]. The selected component is trial interchanged with the adjacent components in the direction of the desired location. Placement techniques using force-directed interchange are described in [Han76, Ios83]. Experimental results are given in [Han76].

Force-directed relaxation is similar to force-directed interchange in the calculation of force vectors for each component. In this technique the primary component is positioned in the compatible slot nearest to its zero force point [Han76]. The component that was occupying that slot is chosen as the next primary component. This results in a series of components to be relaxed that terminates when the primary component is moved to an empty slot. When a series terminates, the new score for the placement is compared to the previous score. If the placement improved, the series is accepted and all components remain in their current positions; otherwise, the series is rejected and all components are returned to their previous position.

Force-directed pairwise relaxation (FDPR) method also uses force vectors to find the zero-force target locations for each component. In this method, however, the primary component does not initiate a series of moves. Instead the primary component A is trial interchanged with a component B in the vicinity of the target location only if the target location of B is in the vicinity of component A [Han76].

Generalized-force-directed-relaxation (GFDR) method is a generalization of the FDPR. It is different from the FDPR method which tries to interchange only a pair of components, whereas the GFDR method interchanges more than two components at the same time. The retails of this method can be seen in [GOT81].

All force-directed techniques share the characteristic that the force is a restricted selection method. Iteration converges quickly, but the process tends to stop even when many productive exchanges still exist.

7.1.3 ACCEPT(New S, S)

Classical iterative techniques accept trial solutions only if the objection function does not increase. This characteristic may cause an algorithm to get stuck in a local minimum rather than finding the global optimum. **Stochastic techniques** use a probabilistic hill-climbing method to avoid the local optima.

7.1.4 ADAPT(parameters)

This procedure adapts the heuristic parameters. These parameters may include such things as the perturbation function, the acceptance function, and terminating criterion.

7.2 Simulated annealing algorithm (SA)

The first presented stochastic approach is the **Simulated annealing.** This algorithm, proposed by Kirkpatrick et al., is a general purpose combinatorial optimization technique that is analogous to the process of metallurgical annealing in which a system is heated and then cooled gradually until the material achieves certain desired metallurgical properties. This method is an extension of a Monte Carlo method developed by Metroplois et al., to determine the equilibrium state of a collection of atoms at any given temperature T. The combinatorial optimization function (cost) is analogous to the energy **E(S)** of a system in state S which must be minimized to achieve a stable system. The probability that a system is in state S is given by **exp(-E(s)/kT)**, where T is the temperature of the system and k is the Boltzmann constant. Starting from an initial configuration, different configurations of the system (states) are generated at random. A perturbation of a system state consists of reconfiguring the system from its current state to a next state within a neighborhood of the solution space. The change in energy (cost) between the two configurations is determined and used to compute the probability p of the system moving from the present state to the next. If the change in cost, dE, is negative, then the change in state is always accepted. If not, then a random number r between 0 and 1 is generated and the new state of the system is accepted if $\mathbf{r} \leq \mathbf{p}$, else the system is returned to its original state. Initially, the temperature is 'high' meaning that a large number of perturbation are accepted. The temperature is reduced gradually according to a cooling schedule, while allowing the system to reach equilibrium at each temperature through the cooling process. Obviously, when T is very high, the algorithm resembles a random walk wherein moves with both positive as well as negative gains are almost always accepted. On the other hand, when T is close to 0, the SA resembles a decent method since only moves with positive gain are accepted.

Table 7.1 Analogy between physical systems and optimization problems

Physical Systems	Optimization Problems
State	Configuration
Energy	Cost Function
Ground State	Optimal Solution
Rapid Quenching	Iterative Improvement
Careful Annealing	Simulated Annealing

SA can be outlined as follows:

```
procedure SA( )
(S, T) ← (S0, T0)
iterations ← i0   // initial number of iterations of inner loop //
repeat
    repeat
        New_S ← PERTURB(S)
        dE ← E(New_S) - E(S)
        if dE < 0 or RANDOM(0,1) > e^-dE/T then S ← New_S endif
    until "inner-loop has been repeated iterations times"
    T ← UPDATE(T)
    iterations ← β×iterations
until "out of time" or "T < Tf"
return(S)
end SA
```

Theoretical study [Rom85] has shown that simulated annealing can climb out of local minima to find a global optimum if the proper conditions on the annealing schedule are satisfied. In practical problems this search for the global optimum implies an infinite number of iterations at each temperature. Since the SA is very time consuming, several approaches have been proposed to speed it by improving the implementations of move selection, cost computation, and temperature scheduling.

Sechen describes a move set design based on range limiting. This discards moves involving components which are move than some specified distance apart; the distance decreases as temperature decrease [Sec84]. Mallela et al. presented a clustering-based technique to reduce the size of the problem that SA has to handle [Ma188]. Annealing schedule improvements are another approach to improving simulated annealing performance. A widely used temperature decrement is a geometric progression, $\mathbf{T} = \alpha \times \mathbf{T}$, where typical α is 0.95~0.98. To improve performance, Huang [Hua86] derives the temperature decrement $\mathbf{T} = \mathbf{T} \times \alpha^{-0.7\mathbf{T}}$ based on the condition required to maintain quasi-equilibrium. The equilibrium condition at each temperature may be a fixed Markov chain length [Aar85], a minimum acceptance [Kir83], or a dynamic Markov chain length [Hua86]. A typical stopping criterion is that the average score is unchanged for a few consecutive temperatures. Another way is to use a two-stage approach: begin with a good reasonably successful heuristic and then follow it with a simulated annealing-based approach for more fine optimization [Hon89, Ros90]. Rose, Klebsch and Wolf presented a method for measuring the temperature of an existing placement [Ros90]. A practical implementation of SA is in the TimerWolf package [Sec85]. It is an integrated set of placement and routing optimization algorithms, all of which use simulated annealing as optimization technique.

Experiments reported in [Nah85, Nah86] compared SA to other heuristics. The conclusion is that several other heuristics performed as well as or better than simulated annealing for a fixed amount of computation time.

7.3 Sequence Heuristic:

Outline of the sequence heuristic algorithm is listed as follows:

```
procedure Sequence_Heuristic( )
S⟵ S0
L ⟵ L0          // initial sequence length //
length ⟵ 0
repeat
        repeat
                New_S ⟵ PERTURB(S)
                 if E(New_S) < E(S) then [S ⟵ New_S; length ⟵ 0]
                                                    else length ⟵ length + 1
                 endif
        until length > L
        UPDATE(L)    // increase L to (say) L+β or βL //
        S ⟵ New_S
until "terminating criterion"
end Sequence_Heuristic
```

This algorithm accepts a new solution New_S with E(New_S) ≥ E(S) if the last L permutations on failed to generate a New_S with E(New_S) < E(S). So, bad permutations are accepted only if one has been unable (over a sequence of attempts) to find a good permutation. The sequence heuristic is slightly easier to use than simulated annealing as one fewer parameter needs to be selected. The results reported by Nahar, Sahni, and Shragowitz [Nah85, Nah86] indicated it is superior to SA.

7.4 Stochastic Evolution

Basically, stochastic evolution improves on the SA approach by using application specific heuristics to select among alternate moves when generating new solutions. The stochastic evolution can be outlined as follows:

```
procedure Stochastic_Evolution( )
S_best ← S ← S_0;
p ← p_0        // p is a control parameter, typical p_0 is 0 //
c ← 0          // c is a counter //
repeat
        C_pre ← E(S)
        S' ← S
        for each m ∈ M do                  // M is a linear set of movable element sorted by a
//
                                                        // priori ordering //
                New_S ← MOVE(S', m)
                dE ← E(New_S) - E(S')
                if (dE < RANDOM(0, p)) then S' ← New_S endif
        repeat
        if   S' satisfies the state constraint then S ← S' endif
        C_cur ← E(S)
        if C_pre = C_cur then p = f(p) else p = p_0 endif
        if E(S) > E(S_best) then [S_best ← S; c ← c-R]
                                        else c ← c+1
        endif
until c > R    // Typical R is between 10 and 20 //
return(S_best)
end Stochastic_Evolution( )
```

The input to stochastic evolution is an initial solution S_0, an initial value of the control parameter p_0, and a parameter R used in the stopping criterion. The stochastic evolution algorithm retains the state of lowest cost among those produced by a series of vertex movement. Each time a state is found which has a lower cost than the best state so far, stochastic evolution algorithm decrements the counter by R, thereby rewarding itself by increasing the number of its iterations. Experiments reported in [Saa91] compared this algorithm to TimberWolf 5.4. The conclusion is that stochastic evolution algorithm is better than the TimerWolf 5.4 in both results and running time.

7.5 Genetic Algorithm

Genetic algorithm (GA) is a general methodology for searching a solution space based on an analogy to biological evolution. As an optimization technique, such a methodology produces good solutions by examining and manipulating simultaneously a set of possible solutions. Let Π be the solution space. Each solution in Π is represented by a string of symbols called **allele**. A genetic algorithm procedure is a sequence of identical sizes sets P_0, P_1, P_2, P_3, ..., where each P_i is a subset of Π. The set of solutions P_i is referred to as the **population** in the i-th generation. The population **evolves** from one generation to the next by means of the **crossover** and **mutation** operations. The **mutation operator** μ is a function from Π to Π**.** For every pair of solutions x and y, a new solution

$\sigma(x, y)$ is created by combining x and y in such a way that they both contribute to the information carried by the solution $\sigma(x, y)$. (The solutions x and y are the parents and $\sigma(x, y)$ is the offspring.) At each generation, a fraction of the population is chosen as parents. The crossover operator σ is then applied to these parents to create a set of offspring. A set of solutions of the same size as the original population is then selected from the combined set of the original population and the offsprings as **survivors** in the next generation. Each such surviving solution may be subjected to mutation by the **mutation operator** μ. Mutation prevents loss of diversity within a population by introducing new solutions. As the evolutionary process proceeds from one generation to the next, better solutions tend to survive and less fit solutions tend to die off. The quality of the solutions in general improves from one generation to the next. Eventually, one or more optimal or near-optimal solutions emerge. Outline of this algorithm is listed as follows:

```
procedure GA( )
P ← P0          // P is the set of population //
p ← | P |       // p is the number of population //
for i  ← 1 to N do
       offspring   ← ∅
       for k ← 1 to p·Kφ do
            (x, y) ← φ(P)
            offspring ← offspring ∪ {ψ(x, y)}
       repeat
       P ← ρ(P, offspring)
       for each string x ∈ P do
              with probability Kμ mutate x with μ
        repeat
repeat
return highest scoring string in P
end GA
```

One of the problems of the GA is its convergence behavior. Initially, the cost of the solutions improves rapidly. But then it becomes very difficult to obtain further improvement. The majority of the runtime is spent in the later phase of the process in which only small improvements are obtained very slowly. Experiments reported in [Won88] compared GA to SA. The conclusion is that both algorithms are comparable in results and running time. Applications of genetic algorithm to the placement problem can be found in [Coh87, Coh91].

7.6 Simulated Evolution (SE)

SE is a general optimization strategy based on evolutionary processes. Outline of this algorithm is listed as follows:

procedure SE( )
$S_{best} \leftarrow S_0$
$S_p \leftarrow$ nil //S_p is a partial solution//
repeat
    $S \leftarrow$ GENERATE(S_p)
    if (E(S) < E(S_{best})) **then** $S_{best} \leftarrow S$ **endif**
    $S_p \leftarrow$ SELECT(S)
until "terminating criterion"
return(S_{best})
end SE

Basically, this algorithm repeatedly alternates between two steps, generate and select. The generate step implements application specific heuristics to derive a new and complete solution from a partial solution; and the select step applies cost functions to evaluate each new solution on a global basis, and then discards the "unfit" parts of the solution. The remaining partial solution is then fed to the generate step in the next iteration to derive a new solution. Throughout these iterations, the best solution is recorded and eventually returned as the final solution. The basic SE algorithm is a greedy search strategy which constructs new solutions out of the "fit" parts of previous solutions. Again, in order to escape from local optima in the solution space, capabilities for uphill moves must be added to this basic algorithm. This may be implemented in the select step by probabilistically discarding the unfit parts of the solutions. Moreover, a catastrophe step may be invoked every now and then to perturb the search by randomly discarding all parts of a solution.

SE is different from other stochastic approaches primarily in the way new solutions are derived from the old solution. Genetic algorithm is quite similar to SE. However, in genetic algorithm, new solutions are generated by combining the features of two or more elements in the previous solution. This contrasts with SE, in which arbitrary heuristics and application specific algorithms are used to derive a new solution from parts of the previous solution.

Experiments reported in [Won88] compared SE to TimerWolf 3.2 with five examples. They concluded that the SE is superior to TimmerWolf 3.2 in both results (in terms of the estimated total wiring length) and running time. Another experimental comparison in [Kli89] found that SE achieves results comparable to SA algorithms (TimerWolf 3.2 and 4.1) with less CPU time. However, recently experimental results that compare SE to TimerWolf 5.4 showed that SE is slightly worse than TimerWolf 5.4 [Kli90, Kli91].

7.7 Hybrid Heuristic (HA)

Outline of this algorithm is shown as follows:

procedure HA( )
$S_{best} \leftarrow S \leftarrow S_0$
$L \leftarrow 0$
repeat
    $S \leftarrow$ PERTURB(S)
    if $E(S) < E(S_{best})$ **then** [$S_{best} \leftarrow S$; $L \leftarrow 0$]
                                else $L \leftarrow L + 1$
    endif
until $L = L_{max}$
end HA

As a classical iterative technique, new solution is accepted if its cost is less than the old one. However, contrary to the classical iterative techniques, this algorithm does not stop after a set of iterations. Rather, it probes deeper into solution space even if it encounters a bad solution. The idea behind searching ahead from a bad solution is to climb up the local optima. Experimental results showed that this algorithm is very fast when compared to SA; however, the obtained results were worse than SA [Bag94].

7.8 Summary

The goal of iterative placement is to transform a complete placement into an improved, complete placement. This process is iterated until some stopping criterion is met. The stopping criterion might be relative or absolute improvement in the placement metric, or perhaps the time expended in the iterative process. In this chapter, a brief review of iterative algorithms is given.