Journal of the Korean Institute
of Industrial Engineers
Vol.21, No.1, March, 1995

Simulated Annealing의 효과적 변형 및 HLS에의 적용*

Effective Variations of Simulated Annealing and Their Implementation for High Level Synthesis

윤복식**, 송낙운***

B. S. Yoon**, N. U. Song***

Abstract

Simulated annealing(SA) has been admitted as a general purpose optimization technique which can be utilized for almost all kinds of combinatorial optimization problems without much difficulty. But there are still some weak points to be resolved, one of which is the slow speed of convergence. In this study, we carefully review various previous efforts to improve SA and propose some variations of SA which can enhance the speed of convergence to the optimum solution. Then, we apply the revised SA algorithms to the scheduling and hardware allocation problems occurring in high-level synthesis(HLS) of VLSI design. We confirm the efficiency of the proposed methods through several HLS examples.

1. 서 론

조합적 최적화 문제(combinatorial optimization problem)의 한 해법으로서 Kirkpatrik et al.(1983)등에 의해 제안된 SA(simulated annealing)은 기존의 반복적인 개선(iterative improvement)에 근거한 발견적 기법(heuristic methods)들이 국부적 최소점(local minimum point)에 빠져버리는 단점을 개선한 범용의 최적화 기법으로 현재까지 CAD를 비롯한 많

* 이 논문은 부분적으로 1993년 한국학술진흥재단의 지원에 의해 이루어졌음

** 홍익대학교 기초과학과

*** 홍익대학교 전자공학과

은 분야에서 응용되고 있다(van Laarhoven and Arts(1987), Wong et al.(1988), Aarts and Korst(1989)). 기본 개념의 단순성과 범용성이 두드러지는 SA는 전체 최소점에로의 수렴성이 이론적인 증명(Gidas(1985), Mitra et al.(1986), Hajek(1988))과 많은 실험 결과들(Johnson(1989,1991), van Laarhoven and Aarts(1987) 참조)을 통해 타당성이 확인되어 특별한 대안이 없을 때 편리하게 사용할 수 있는 최적화 알고리듬으로서 일반적으로 받아들여 지고 있다. 그러나 아직까지 SA의 수렴속도와 수렴양태 등에 관해 밝혀지지 못한 특성들이 있어서 충분히 효율적인 SA의 적용이 이루어지지 못하고 있다. 본 논문의 일차적인 목적은 SA의 효율성을 향상시키기 위한 효과적인 변형방법에 초점을 맞추어 새로운 변형 알고리듬을 만들어 내는 것이고 이와 동시에 최근 많은 관심을 끌고 있는 VLSI 디자인의 상위 단계 합성(HLS : high-level synthesis) 문제에 적용하여 효율성을 확인하고 실제적인 응용을 시도하는 것이다. 이를 위해 우선 최근까지의 SA의 응용사례 및 이론적 연구를 통해 밝혀진 SA에 기반을 둔 기존의 알고리듬과의 비교를 행하여 효율성을 확인하고, 동시에 HLS에서 제기되는 다양한 형태의 문제에 SA가 매우 융통성 있게 적용될 수 있음을 보인다.

디지탈 시스템의 디자인을 자동화하려는 많은 노력 덕분에 트랜지스터들의 연결을 정하여주는 circuit-level이나 gate나 flip-flop들의 network를 설정하여 주는 logic-level과 같은 하부 혹은 중간 단계에서의 디자인은 이미 실용화의 단계에 들어서게 되었다[Michel (1992) 참조]. 보다 높은 단계의 디자인, 즉 algorithm-level에서 주어진 behavior description에 따라 register, ALU, multiplexer, bus들과 그것들이 콘트롤을 적절히 설정하여 RT-level(register-transfer level)의 구조를 마련하여 주는 과정의 자동화에 관한 연구가 활발히 진행되고 있다[Mcfarland et al.(1990), Michel et al.(1992) 참조]. 현재까지 충분히 효율적이고 실용적인 범용의 디자인 시스템이 개발되지는 못하였지만 프로세서의 구조를 정하는 단계에서의 자동화는 architecture 디자인 과정에 소요되는 시간을 단축해 주고, 그 과정 자체를 표준화해 줄 수 있으며 또한 동일한 임무를 수행하는 다양한 디자인들을 손쉽게 만들어 비교해 볼 수 있다는 명백한 장점때문에 최근 많은 관심을 끌고 있다. 이러한 상위단계에서의 디자인 자동화 과정이 HLS(High-Level Synthesis)의 대상이 된다. HLS에서 관건이 되는 것은 연산들을 시간슬롯에 배정하여 연산의 수행시작 시간을 설정해 주는 문제(스케쥴링)와 ALU와 같은 하드웨어를 연산에 배정해 주는 하드웨어 할당 문제인데 이것들은 전형적인 NP-하드 조합적 최적화 문제가 된다. 실제 HLS 문제의 경우 규모도 커지고 설계의 주안점에 따라 다양한 형태의 조합적 문제가 발생하기 때문에 융통성과 간편성이 두드러지는 SA가 매우 편리하게 이용될 수 있을 것이다.

본 서론에 이어 2장에서 SA의 기본 형태를 소개하고 실제 적용에서 대두되는 문제점들을 지적한 후 3장에서 SA의 효율성을 향상시키기 위한 기존의 이론적 실험적 연구와 효과적인 변형 방법중 하나인 SE(Stochastic Evolution) 알고리듬을 소개한다. SE는 속도를 증진시키는 장점이 있으나 수렴성을 보장

하지는 못하는 단점이 있는데 4장에서는 SA의 수렴성을 유지하면서 수렴속도를 증진시키는 변형 방법(SA1, SA2)를 제시한다. 5장에서는 HLS에서의 다양한 스케쥴링/할당 문제에 SA를 적용하는 방법을 제시하고 구체적인 사례를 통해 SA, SA1, SA2의 성능을 비교하여 변형의 효율성을 검증하고 마지막으로 6장에서 결론과 향후 연구방향을 언급한다.

2. 기본 알고리듬과 적용상의 문제점

현재 실용적으로 애용되고 있는 SA 알고리듬은 아래와 같은 형태이다.

```
ALGORITHM SA
INITIALIZE(X,T,L);
repeat
   for i:=1 to L do
      Y = PERTURB(X);
      if (C(Y) ≤ C(X)) or (exp( (C(X)-C(Y)) / T ) > random(0,1))
        then
            X = Y; {accept the movement}
      endif;
   endfor;
   UPDATE(T,L);
until (Stop_Criterion);
```

그림 1. 알고리듬 SA

비용함수(C)를 최소화하는 조합적 문제에서 현재의 해(X)로부터 적절한 방법으로 X에서 약간 변형된 새로운 해(Y)를 만들어서 새로운 해의 비용함수의 값이 현재 해의 그것보다 작으면 새로운 해를 최적해 후보로 받아들이고, 그렇지 않을 경우에는 기각하는 방법이 기존의 국부적 탐색(local search)에 의한 반복적인 방법이라면 나쁜 해가 나왔을 경우에도 무조건 기각하지 않고 적절한 확률(exp((C(X)-C(Y)) / T)로 새로운 나쁜 해를 받아들여서 국부최소점에서 빠져나올 수 있도록 하여 전체 최소점을 찾아가게 하는 것이 SA의 요점이다. 위의 알고리듬에서 확정을 시켜주어야 하는 과정들을 구체적으로 살펴보며 문제점들을 찾아보면 :

(1) INITIALIZE에서는 알고리듬이 시작하는데 필요한 초기해 X와 콘트롤 파라미터 T, 안쪽 루프의 반복수 L을 결정해 주어야 한다. T는 통계역학에서 결정상태를 얻기 위한 annealing 과정에서의 온도와 같은 역할을 하는 콘트롤 파라미터인데 초기값이 충분히 높을 필요가 있는데 현재까지 적절한 초기치를 설정하는 편리한 방법이 확립되지 못하고 있다. L은 주어진 온도에서 평형상태에 이를 때까지 무작위적인 입자들의 이동을 계속하게 하는 annealing의 과정에 해당하는 안쪽 루프에서의 반복수이다. 역시 L의 적정한 값을 설정하는 규칙이 확립되지 못하고 있다. 초기해 X는 보통 무작위적으로 선택이 되나 이론적으로는 알고리듬의 성능에 별 영향을 주지 않아야 한다.

(2) PERTURB는 현재의 해로부터 새로운 해를 만들어 주는 과정이다. 이것은 구체적인 문제의 특성에 따라 정해지는데 해의 수렴까지 매우 많은 반복이 계속되므로 되도록 단순한 과정으로 만드는 것이 좋다. 이 과정은 알고리듬의 성능에 직접적인 영향을 미치므로 문제의 구조에 따라 효율적인 해 변동 방법을 찾아내는 노력이 요구된다.

(3) 해 변동 결과 목적함수 값이 커졌을 경우에 나쁜 해를 받아들이는 기준으로 「exp((C(X)-C(Y)) / T) > random(0,1)」를 사용하고 있는데 이는 Metropolis 기준(criterion)에 따른 것이다. 알고리즘의 수렴성을 위해서는 꼭 이 기준을 따를 필요는 없는 것이 경험적으로(Saab and Rao(1991)) 또는 이론적으로

(Anily and Federgruen(1987) 관찰되고 있다.

(4) UPDATE(T, L)의 과정은 쿨링스케줄이라고 부르는 과정으로 알고리듬의 전체 최소점으로의 수렴성에 직접적인 영향을 미치는 과정이나 아직까지 충분히 효율적이며 편리한 쿨링스케줄이 얻어지지 못하고 있다. 이론적인 수렴을 보장하기 위해서는 $T_k=\frac{d}{\log k}$와 같은 로그적 스케줄을 사용하여야 하나 수렴속도가 너무 느리기 때문에 실제로는 $T_k=aT_{k-1}$, $0<a<1$와 같은 기하적 스케줄을 많이 사용한다.

(5) Stop—Criterion은 이론적으로는 온도(T)가 충분히 낮을 때에 멈추는 방식으로 되어야 하나 실제로는 목적함수의 감소에 근거하여 바깥 루프가 특정한 횟수만큼 반복할 동안 계속 해의 변동이 없으면 종료를 하는 방식을 많이 쓰고 있다.

이상에서 알 수 있듯이 SA는 현재까지의 수많은 적용 사례에도 불구하고 아직 충분히 성숙된 알고리듬이 아니다. 많은 적용 결과에서 관측되고 있는 거의 일치된 결론은 충분한 시간이 주어진다면 SA가 최적해에 만족할 정도로 가까이 간다는 사실인데 걸리는 시간이 상당히 많다는 것이 단점으로 지적되고 있다. 그러나 다른 효과적인 알고리듬이 없는 많은 실제적인 문제에서 큰 어려움 없이 SA가 적용될 수 있으므로 SA와 기본적으로 유사하면서 수렴속도가 빠른 변형을 생각해 보는 것이 가치가 있을 것이다.

3. SA의 성능 및 변형

3.1. SA의 성능에 관한 연구

현재까지 SA에 대한 연구는 알고리즘의 성능에 관한 연구와 효과적인 적용에 관한 연구로 대별해 볼 수 있다. 알고리즘의 성능에 관한 연구는 수렴성에 관한 이론적인 연구와 특정한 표준적인 문제에서 기존의 알고리즘과의 성능을 비교하는 실험적인 연구로 나누어 볼 수 있다.

수렴성에 관한 이론적인 연구는 주로 전체 최소점으로의 수렴을 보장하기 위한 효과적인 쿨링스케줄(콘트롤 파라미터 T)을 발견하는데에 주안점을 두어 행해졌다(Mitra et al. (1986), Gidas(1985), Connors and Kumar (1988, 1989), Tsitsiklis(1989), Rose et al. (1990), Stern(1992) 등), 그러나 수렴성을 위한 조건들이 너무 엄격하여 실용적인 쿨링스케줄을 개발하는 데에는 아직까지 크게 기여하지 못하고 있다.

알고리듬의 성능에 관한 실험적인 연구는 매우 많이 발표되었다. 전형적인 NP-hard 문제인 TSP(traveling salesman problem)의 경우 대체적으로 기존의 방법[Lin and Kernighan (1973)] 보다 SA가 우수한 성능을 보인다고 보고되었다(Van Laarhoven and Arts(1987)). GPP(graph partitioning problem)의 경우는 Johnson et al.(1989)에서 상세히 실험되었는데 SA가 시간은 많이 소비하지만 기존의 알고리즘(Kernighan and Lin(1970))을 통해 얻은 최종해보다 최대 5% 향상된 최종해를 주었고 특히 sparse graph의 경우는 70% 향상된 최종해를 주었으나 특별한 구조가 있는 그래프에서는 기존의 알고리즘이 더 나은 결과를 주고 있는 것으로 밝혀졌다. $o(n^3)$의 정확한 알고리즘이 알려져 있는 MWMP(minimum weighted matching problem)의 경우에는 기존의 근사적 알고리듬보다 훨씬 빠르게 (o

(n)) 최적해에 접근하며 30-50% 정도 향상된 최종해를 주었다(van Laarhoven and Arts (1987)). 이 경우들은 SA가 문제의 특성에 근거한 특수한 발견적 알고리듬보다도 오히려 빠르고 좋은 결과를 주는 고무적인 예가 되지만 GCP(graph coloring problem)(Johnson et al.(1991))를 비롯한 많은 경우(QAP (quadratic assignment problem), 스케쥴 문제 등)에 기존의 알고리듬보다 좋은 해를 주는 대신에 시간이 많이 걸리는 것을 관찰할 수 있다(Aarts and Korst(1989). 또한 NPP (number partitioning problem)의 경우는 기존의 알고리듬 보다 나쁜 해를 주고 있으며 더우기 초기치를 달리하여 여러번 국부탐색(local search)를 하는 것보다도 좋지 못한 결과를 보이고 있다(Johnson et al.(1991)). Nahar et al.(1986)의 LAP(linear arrangement problem)에 대한 실험 결과도 시간이나 오차 측면에서 모두 SA가 기존의 근사적 알고리듬을 능가하지 못함을 보이고 있다.

위의 결과에서 보면 SA의 효율성이 경우에 따라 상반되기도 하여 아직까지 SA의 효율성에 대해 뚜렷한 결론을 내리기는 이르다는 것을 알 수 있다. 우선 충분한 정도의 평균적인 분석이 이루어지지 못해서 실험 결과의 타당성에 의심을 갖게 하는 것도 이유의 하나이지만 무엇보다도 대부분의 실험에서 너무 단순한 쿨링스케쥴을 사용하여 SA의 잠재적인 능력을 충분히 살리지 못하여 불공정한 비교가 된 것이 결점으로 지적되고 있다 (van Laarhoven and Aarts(1987)). 하지만 이러한 불충분한 증거에도 불구하고 SA는 다양한 공학문제에 이미 적용되어 편리하게 사용되고 있다. 그중에 특히 효과적인 알고리듬이 개발되어 잇지 못한 VLSI 디자인, 이미지 처리(image processing) (Geman and Geman(1984), Boltzman machine(Arts and Korst (1989)) 등의 분야에서 성공적으로 적용되고 있고 다른 분야에서도 적용 사례가 계속 발표되고 있다. VLSI 디자인의 경우 placement, routing 등의 문제를 해결하는 SA를 기반으로 하는 Timber-wolf 패키지가 개발되어 (Sechen and Sangiovanni-Vincentelli(1985)) 표준적인 방법으로 인정을 받고 있고 그외 매우 많은 적용사례가 발표되고 있다(Wong et al.(1988) 참조). 예를 들면 HLS의 경우에 Devadas and Newton(1989), 통계학에 적용한 경우는 Lundy(1985), LAN에의 적용은 Fetterolf and Anandaligam(1991) 등이 있다.

실제적인 문제에 SA를 적용한 사례에서 보면 기존의 방법보다 만족스러운 해를 주지만 대개의 경우 상당히 오랜 시간을 필요로 함을 알 수 있다. 그러나 초기점을 달리하여 여러번의 국부탐색을 반복하는 경우보다는 같은 시간에 훨씬 좋은 해를 주게 된다. 실제 문제에 SA를 적용할 때는 알고리듬의 효과적인 구현(implementation)이 특히 중요한데, 해의 이동 방법, 쿨링스케쥴의 데이타 구조 등에 따라 알고리듬의 성능이 크게 변하기 때문이다.

일반적인 관점에서 SA의 개선 방향은 대략 세가지로 나누어 행해지고 있는데 SA의 변형을 통해(Nahar et al.(1986), Saab and Rao (1991), Kling and Banerjee(1989)), 병렬 처리를 통한 방법(Aarts and Korst(1989)), 효과적인 쿨링스케쥴을 찾는(Tsitsiklis(1989)) 시도가 그것이다. 특히 첫번째 방법은 SA의 간단한 변형을 통한 상당한 효과를 이미 경

험적으로 관찰할 수 있고(Saab and Rao (1991)), 이것을 상세하게 이론적으로 조사하면 SA의 동적인 변동의 본성을 파악하여 결국은 즉 효과적인 쿨링스케줄을 발견하는 방법과 같은 효과를 기대할 수 있을 것이다.

3.2. 알고리듬 SE

Saab and Rao(1991)는 SE(Stochastic Evolution)이라고 하는 SA의 변형된 형태를 제안하였는데(그림 2 참조) graph partitioning, TSP, standard cell placement 문제에 적용하여 상당한 속도의 개선이 있었다고 보고하고 있다. SE에서는 너무 큰 상승이동(uphill movement)은 최적화로의 진행과정을 혼란시키고 계산시간만 증가시킨다고 파악하여 때문에 작은 규모의 uphill만 허용한다. 이 알고리즘에서는 p라고 하는 초기값을 0에 가까운 값으로 잡고 하강이동(downhill movement)는 무조건 받아들이고 상승이동은 비용함수의 값의 차이가 [-p, 0] 사이의 난수보다 크면 받아들인다. 그리고 연속되는 반복에서 목적함수의 변화가 없으면 p의 값을 올려서 국부최소점에서 빠져나올 기회를 준다.

종료기준은 R이라는 값을 미리 설정해 놓은 다음 iteration을 할 때 마다 카운터를 1씩 증가시켜 나가다가 목적함수의 값이 지금까지의 가장 좋은 해보다 더 좋으면 카운터에서 R값을 뺀 값을 카운터에 유지시켜 나가다가 카운터가 R보다 커질때 끝내는 방법을 사용하고 있다.

4. SA의 효과적인 개선

본절에서는 SA의 개선의 폭표를 최적에 가까운 해에 빨리 도달하게 하고, 사용자가 미리 설정해 주어야 하는 파라미터의 수를 되도록 줄여 간편성을 높이고, 되도록 SA의 기본틀을 유지하여 이론적인 수렴성을 손상시키지 않는 것에 두고 다음과 같은 변형을 행한다.

<Stochastic Evolution>

```
ALGORITHM SE:
INITIALIZE(X,T):
XBest = X;
Counter = 0;
repeat
   CostOld = C(X);
   for (some possible move) do
      Y = PERTURB(X);
      if ( C(X) - C(Y) > RANINT(-T,0) ) then
         X = Y;
      endif;
   endfor;
   CostNew = C(X);
   UPDATE(T, CostOld, CostNew)
   if (C(X) < C(XBest) ) then
       XBest = X;
       Counter = Counter - L;
   else
       Counter = Counter + 1;
   endif;
until (Counter > L);


procedure UPDATE(T, CostOld, CostNew):
if (CostOld = CostNew) then
   T = T+1;
else
   T = TInitial;
endif;
```

그림 2. 알고리듬 SE

(1) 안쪽 for 루프가 한차례 끝날 때마다 그때까지의 최소의 목적함수 값을 준 해를 기억하여 최종해로 준비한다. 이것은 보다 최적에 가까운 해를 구하는 명백한 방법이다. 물론 내부 루프내에서도 매 해의 이동마다 비용함수값을 비교하여 현재까지의 최소해를 추적해 갈 수도 있으나 계산의 증가분에 대한 해의 질의 향상 정도를 비교해 보아야 할 것이다.

(2) acceptance criteria : 기본 알고리듬에서는 해 변동 결과 목적함수 값이 커졌을 경우에 Metropolis 기준을 적용하여 확률 exp(-D/T), D = 변동후의 목적함수 값 - 변동전의 목적 함수 값 으로 변동된 나쁜 해를 받아들이고 있는데 알고리즘의 수렴성을 위해서는 꼭 이 기준을 따를 필요는 없는 것이 경험적으로(Saab and Rao(1991)) 또는 이론적으로(Anily and Federgruen(1987) 판찰되고 있다. 이 기준을 T가 클 때 exp(-D/T) ≅ 1-D/T 임을 이용하여 확률 (1-D/T)로 받아들이도록 수정하여 계산의 효율성을 높일 수 있다. 본 연구에서는 이러한 기준을 따르는 경우(알고리듬 SA2)와 Metropolos 기준을 따르는 경우(SA, SA1)을 모두 고려하여 성능을 비교한다.

(3) 냉각스케줄 : 수렴의 촉진을 위하여 기본적으로 기하적인 스케줄을 사용하되 내부 for 루프의 반복 횟수 L을 줄이기 위하여 매회 내부 루프를 끝낸 후 만약 비용 감소가 없으면 T를 그대로 두고 있으면 a*T (a=0.9 또는 0.95로) 감소 시킨다. 이렇게 하는 이유는 이론적으로 고정된 T에 대해 내부 루프의 전이를 L번 일으킨 마코프체인이 안정상태에 도달해야 하는데 안정상태에서는 목점함수의 평균값이 T값이 낮아짐에 따라 낮아져야 한다. 따라서 내부 루프를 돌고 난 후에도 목적함수 값에 변화가 없으면 아직도 온도 T에서의 안정상태에 도달하지 못했다고 간주할 수 있으므로 T를 고정시킨 채 내부 루프를 계속 돌린다. 이렇게 함으로써 L을 최소로 줄일 수 있어서 내부루프를 필요 이상 많이 돌리는 데 따른 시간의 낭비를 방지할 수 있고 개별 문제에 따라 적절한 L을 설정해야 하는 번거러움을 예방할 수 있다. 실제로 이러한 방법을 통해 많은 계산상의 이득을 볼 수 있었다(4.2절 참조).

(4) 초기해 : 경험적으로 초기해가 최적에 가까우면 수렴이 빠른 것을 관찰할 수 있는데 초기해는 되도록 최적에 가까운 해를 손쉽게 이용할 수 있는 간단한 방법에 의해 구하여 시작점으로 사용한다(5.2절 참조).

(5) 종료조건 : 내부루프를 M(=5)번 반복하는 동안 비용의 변화가 없거나 내부루프 안에서 최소점을 체크하여 최소점이 발견되면 Counter2=0으로 놓고 아니면 Counter2를 증가시켜 Counter2가 N = 내부루프수 x 상수(예를 들면 4xM) 보다 커지면 종료한다. 이것은 SA가 낮은 온도에서 실제적인 최소값의 감소없이 계속 목적함수값을 변동시키는 해의 이동을 반복하여 시간을 소모하는 현상을 방지하기 위한 장치이다.

이와 같은 고려에서 다음 그림 3과 같은 변형된 알고리듬 SA2를 얻을 수 있다. SA2에서 acceptance 기준을 원래의 Metropolis 기준으로 놓은 알고리듬을 SA1이라고 부르기로 한다. SA1, 2는 SE와 달리 기본적으로 SA의 기본 구조를 변화시키지 않았고 오히려 내부 루프에서의 마코프 체인의 평형성을 더욱 강화하였기 때문에 해의 이론적인 수렴성은 자동적으로 유지되고 5장의 실험에서 보듯이 수렴속도가 매우 빨라짐을 확인할 수 있다.

5. HLS에의 적용

5.1. HLS에서의 조합적 문제

HLS의 효율적인 수행에 가장 핵심이 되는

```
ALGORITHM SA2:
INITIALIZE(X,T,L);
XBest = X;
Counter1 = 0;
Counter2 = 0;
repeat

   CostOld = C(X);
   for i = 1 to L  do
      Y = PERTURB(X);
      if ( C(X) - C(Y) > RANINT(-T,0) ) then
          X = Y; {accept the movement}
      endif;
      if (C(X) < C(Best) ) then
         XBest = X; {update the best state}
         Counter2=0;
      else
         Counter2=Counter2 + 1;
      endif
   endfor;
   CostNew = C(X);
   UPDATE(T, CostOld, CostNew)
   if (CostNew = CostOld)) then
       Counter1 = Counter1 + 1;
   else
       Counter1 = 0;
   endif;
until (Counter1 > M or Counter2 > N);


procedure UPDATE(T, CostOld, CostNew)
if (CostOld > CostNew) then
   T = a*T
endif;
```

그림 3. 알고리듬 SA2

것은 데이타 경로(data path)를 최적의 콘트롤 스텝에 배정을 하여 주는 스케쥴링과 이에 따른 하드웨어(ALU, Register, MUX, BUS 등)의 할당(allocation)의 문제인데 이것은 전형적인 NP-hard 문제이다. 시스팀의 복잡성을 최소화하면서 주어진 임무의 처리 속도를 최대화하는 최적의 디자인을 구현하기 위해서는 스케쥴링과 하드웨어 할당의 문제를 상호 연관시켜 고려하여야 하는데 이에 대한 첫번째 접근 방법은 먼저 스케쥴링을 적절히 임의로 설정하여 그 상황하에서의 할당 문제의 최적해를 구해 내고, 이번에는 이 최적해로 할당을 하여 놓고 그안에서 최적 스케쥴을 구하는 방식을 반복적으로 수행하여 개선해가는 방법으로 Marwedel(1986)의 MIMOLA, Park and Parker(1988)의 Sehwa등에서 대표적으로 채택되고 있다. 두번째 접근 방법은 스케쥴링과 할당을 동시에 고려하여 전체적인 비용 함수를 만들어 이를 최소화 시키는 스케쥴과 하드웨어 할당을 찾아내는 보다 직접적인 방법으로 최근에 많이 발표되어 그 효율성을 인정받고 있다. 물론 후자의 방법이 최적화의 관점에서 바람직하나 일반적으로 계산의 복잡성 때문에 적용가능한 문제의 규모에 제한이 있을 수 있다. 대표적인 예로는 HAL(Paulin and Knight(1989)), MAHA (Parker et al.(1986))등의 발견적인 방법, 정수 계획법을 이용한 방법(Hwang et al. (1991)), SA에 근거한 방법(Devadas and Newton(1989))등이 있다. 이중에서 정수계획법을 이용한 접근 방법은 적용할 수 있는 문제의 규모가 매우 제한되기 때문에 실용성이 문제가 된다. 이에 비해 발견적인 기법은 계산상의 신속성이 큰 잇점이나 근본적으로 최적해를 보장해 주지 않기 때문에 적용에 유의해야 한다. SA는 속도가 느리고, 해가 비용함수의 설정에 크게 의존하게 되는 단점이 있으나 최적해에 가까운 해를 얻을 수 있고 본래적인 범용성 때문에 다양한 관점에서의 HLS 설계 문제에 융통성 있게 적용될 수 있는 장점이 있다. 즉, 일반적으로 FDS는 발견적인 기법에 의한 방법이므로 최종 결과가 항상 의도한 최적성을 갖기는 기대하기 힘드나, 이에 비해 SA기법은 본래적인 최적해에의 수렴성과 범용성때문에 보다 실용적인 큰 규모의 문제에 적합한 기법이라고 볼 수 있다. Devadas and Newton(1988)에 의한 HLS에의 적용에서는 만족할 만한 효율성 및 실

험이 추구되지 못하였다. 즉, 문제의 모형화 방식과 알고리듬 적용방법에 개선할 여지가 많은데 본 연구에서는 우선 효과적으로 변형된 SA를 제시하고 실제 적용에 있어서 파이프라인 구조 등 다양한 설계 방식에 적용될 수 있도록 해공간의 구조를 이차원적 기본구조의 내부에 배후구조를 두는 방식으로 개선한다.

5.2. 문제의 형태

일단 입력 프로그램이 CDFG(control data flow graph) 형태로 변환되어 있다고 가정하자. HLS에서의 스케쥴링 및 하드웨어 할당 문제는 세로 방향을 콘트롤 단계의 시간슬롯 축으로 하고 가로 방향을 하드웨어 축으로 하는 2차원 격자형을 만들어 CDGF에서 주어진 오퍼레이션을 배치하는 문제로 정립할 수 있다. 즉 주어진 문제의 기본 연산을 0_i, $i=1,\cdots,n$ 이라고 하고 해집합을 S 라고 하면 하나의 해, $X \in S$ 는,

$$X=[(\mathrm{TimeSlot}_i,\ \mathrm{ALU}_i)_{i=1,\cdots,n}]$$
subject to : CDFG, 시간제약 (1)

과 같이 표현될 것이다. 이때 S는 기본적으로 2차원 격자구조의 형태지만 배후에 CDFG와 파이프라인 설계에서의 다양한 Latency들, 하드웨어 그룹의 모듈화등에서 발생하는 배후구조를 가지고 있는 해공간이다.

해(스케쥴링과 하드웨어 할당이 이루어진 상태) X의 시간과 하드웨어를 고려한 비용함수는

$$C(X)=p1*\mathrm{Number\_Alu}(X)+p2*\mathrm{Number\_Register}(X)+p3*\mathrm{Number\_Bus}(X)+p4*\mathrm{Number\_Mux}(X)+p5*\mathrm{Latency} \quad (2)$$

로 놓을 수 있다. 여기서 p5는 Latency의 중요도에 따라 적절히 값을 설정하고 p1, p2, p3, p4는 각각 단위 하드웨어들의 필요면적을 추정하여 값을 주면 될것이다. 이들 값은 설계자의 의도에 따라 적절히 설정하여 주어 다양한 관점에서의 스케쥴링/할당을 할 수 있을 것이다. 본 연구에서는 연산의 특성에 따라 걸리는 다르게 줄 수 있도록 허용하여 보다 일반적인 상황에서의 스케쥴링을 행할 수 있도록 한다. 예를 들어 덧셈은 1 시간슬롯, 곱셈은 2 시간슬롯 등으로 ALU에서의 지연시간을 다르게 줄 수 있다. (2)에서 Number_Alu(X), Number_Register(X), Number_Bus(X)를 구할때는 Devadas and Newton(1989) 등에서 사용하고 있는, 각 시간슬롯에서의 이들의 값을 구하여 그중 최대값을 선택하는 방법을 개선시켜서 파이프라인 방식과 module-binding이 고려될 수 있도록 하여 보다 정확한 하드웨어 비용을 반영될 수 있도록 한다.

(1), (2)에서 볼 수 있듯이 본 연구에서 고려되는 최적화 문제는 매우 융통성이 있다. 즉, SA에서 해의 이동범위를 (1)의 제약조건에 따른 feasible solution set 범위내부로 제한함으로써, 설계자가 의도하는대로 하드웨어 제약적인 문제, 또는 시간 제약적인 문제를 풀어서 최선의 디자인을 얻어 낼 수 있다.

5.3 구체적인 구현 방법

본 문제의 특성은 일단 CDFG에서 주어진 데이타 경로의 조건을 벗어나면 조건을 만족하는 해로의 이동이 매우 시간소모적인 과정이 되므로 해의 이동을 feasible 영역안에서만 수행한다. 이렇게 함으로써 근방(neighborhood)의 규모를 상당히 줄일 수 있어서 해의 이동이 보다 손쉽게 행해질 수 있다. 식(1), (2)의 해공간과, 제약조건, 목적함수에서 보듯이 시간에 대해서는 항상 feasibility를 유지하고 하드웨어는 목적함수화 하여 스케쥴링 및 할당을 시도한다.

HLS의 첫단계에서는 ASAP 스케쥴[3]과 주어진 시간제약 조건에 따른 ALAP 스케쥴[3]을 수행하고 주어진 L을 고려하여 각 연산별로 빠른 시작시간과 늦은 시작시간을 설정하여 SA1의 INITIALIZE(X, T, M)에 필요한 정보를 제공한다. 즉 초기해(X)를 ASAP 스케쥴과 ALAP 스케쥴중에서 비용함수 값이 적은 스케쥴링으로 둔다. 이때 ASAP과 ALAP 스케쥴은 주어진 CDFG에서 최단 수행시간을 결정하고 해의 feasibility 여부를 결정해 주는 필수적인 과정이므로 초기해를 위한 여분의 시간을 필요로 하지 않는다. 시간슬롯의 수를 고정(최단시간)시키지 않고 최단시간 이외의 스케쥴링을 고려하기 위해서는 ALAP의 방법을 원하는 최종시점을 기준으로 행하면 된다. M(SAO의 경우는 대략 연산 갯수의 5-10배 정도로 설정하였으나, SA1의 경우에는 SAO 경우의 1/3 - 1/4 정도의 값으로 충분함)은 적절히 설정한다.

둘째 단계에서는 본격적으로 해를 이동시키며 SA 스텝들을 수행하는데 PERTURB(X)에서는 다음과 같은 2가지 기본 이동을 수행한다.

a. 1개의 연산의 위치를 이동

b. 2개의 연산의 위치를 상호 교환

이중에 a는 원래해와 유사한 근방의 해로의 이동이라고 볼 수 있으며, b는 현재의 해로부터의 비교적 급격한 이동이라고 볼 수 있다. 이동을 무작위적으로 만들어 내기 위해서 PERTURB(X)에서는 우선 난수를 연산 갯수의 범위내에서 1개 발생시켜 기준연산의 번호를 정한 후에, 전체 연산 갯수의 5배 정도의 범위에서 새로운 난수를 발생시켜 난수가 총 연산의 갯수보다 클 경우는 이동 a를, 그밖의 경우는 b를 선택하는 방식으로 하여 실제로 a에 의한 이동을 더 많이 일어나게 조정한다. 이때 a의 경우 이동 위치는 가능한(feasible) 범위내에서 무작위적으로 정하고 b의 이동의 경우에는 두번째 난수가 기준연산과 상호교환될 번호가 된다. 본 연구에서는 연산의 지연시간을 임의로 줄 수 있게 하였으므로 b에서 선택된 두개의 연산들이 차지하는 시간슬롯의 수가 다를 경우가 발생하는데 이때에는 이동하고자 하는 연산의 시간슬롯의 크기를 고려하고 동시에 그 연산의 전후 관계를 살펴 가능한(feasible) 이동의 범위를 설정한다.

UPDATE(T, M)에서 온도(T)를 매 반복마다 약간 내려주어야 하는데(cooling schedule), 본 연구에서는 0.9를 곱해 주고 M은 고정을 시키는 방식을 택하였다. SA1에서는 내부루프의 수렴성과 길이가 자동 조정되므로 비교적 급격한 냉각스케쥴을 따라도 최종해의 질에 큰 해가 되지 않을 것이다.

5.4. 간단한 미분방정식의 예

본절에서는 Paulin and Knight(1989)에서 예시된 있는 미분방정식 $y''+3xy'+3y=0$ 을 푸는 다음의 알로리듬을 예로 들어 SA1에 의한 스케쥴링 결과를 SA, SE와 비교하여 효율성을 확인한다. 알고리듬

```
while (x,a) repeat
        x1=x + dx ;
        u1=u - (3*x*u*dx) - (3*y*dx) ;
        y1=y + (u*dx) ;
        x=x1 ; u=u1 ; y=y1 ;
end ;
```

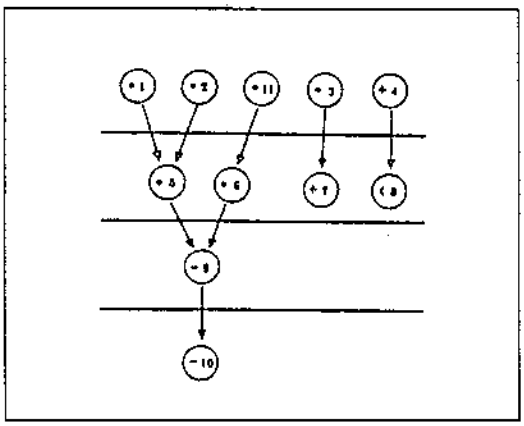

그림 4. 미분방정식 예제의 ASAP 스케쥴

C-STEP	ADD.	SUB.	MULTI	DIV.	ETC.
1	4		1 2		
2			5 11		8
3		9	3 6		
4	7	10			

그림 5. SA계열 알고리듬에 의한 전형적인 스케쥴링

을 CDFG로 나타내어 가능한 한 빠른 시점에 연산을 시작하게 하는 ASAP 스케쥴을 하면 그림 4와 같다. 이것을 각 연산에 번호를 붙여서 세로축을 시간슬롯 가로축을 ALU 하드웨어로 놓은 격자틀에 스케쥴을 SA, SA1, SA2, 또는 SE를 통해서 시행했을 때 얻어지는 전형적인 결과를 그림 5에서 볼 수 있다. 그림 5에서는 모든 연산을 1 단위슬롯씩 걸린다고 가정하였다.

표 1에는 이들을 5회 정도 돌려서 얻은 평균 비용함수(비용)값과 평균 소요시간이 나와 있는데 SA1, SA2, SE들이 SA 보다 우월한 결과를 보임을 확인할 수 있다. 거의 모든 예제에서 비용함수는 p1=10, p2=1, p3=5, p4=0, p5=10으로 두었는데 설계자의 의도와 판단에 따라 가장 합리적인 값을 주면 될 것이다. 표 2는 시간슬롯의 수를 7개로 늘인 경우의 결과를 보여주고 있는데 시간비용을 고려하지 않을 때는 시간슬롯의 수의 증가에 따라 하드웨어 비용이 감소하는 것을 관찰할 수 있다. 이 경우에는 SE가 매우 열등한 성능을 보이는데 이것으로부터 SE가 문제에 따라 최소값에 수렴하기 이전에 수행을 끝내버리는 단점이 있음을 알 수 있다. 참고로 SE의 내부루프수와 종료조건을 강화하여 133초간 수행된 실험에서는 비용이 다른 알고리듬과 같은 151이 얻어짐을 관찰할 수 있었다. 표 3에서는 파이프라인 구조로 L=2로 스케쥴링을 할 때 시간슬롯수를 7개로 늘린 경우의 결과를 보여주고 있는데 역시 SA1의 성능이 우월함을 관찰할 수 있다. SA2는 대개의 경우 SA1에 비해 시간의 단축 효과가 없는 것으로 관찰되고 있다.

표 1, 표 2에서는 비용에 가장 중요한 영

향을 미치는 ALU 갯수를 볼 수 있는데 이는 여러번의 실험의 결과에서 나타나는 갯수들이므로 항상 같을 수는 없다. 그러나 시간슬롯수가 늘어남에 따라 필요한 ALU의 갯수가 줄어드는 것을 확인할 수 있다.

표 1. 미분방정식 예제(non-pipelined, 시간슬롯수 4)

	SA	SA1	SA2	SE
소요시간(초)	51.23	11.17	11.88	11.44
비용함수값	174	174	174	174
ALU 갯수	4	4	4	4

표 2. 미분방정식 예제(non-pipelined, 시간슬롯수 7)

	SA	SA1	SA2	SE
소요시간(초)	72.19	18.07	21.84	23.96
비용함수값	151	151	151	161
ALU 갯수	2 또는 3	2 또는 3	2 또는 3	2 또는 3

표 3. 미분방정식 예제(piplined:L=2, 시간슬롯수 7)

	SA	SA1	SA2	SE
소요시간(초)	141.12	42.54	44.28	71.63
비용함수값	157	157	160	158

5.5. 파이프라인 FIR 디지탈 필터에서의 다양한 스케쥴링

본절에서는 변형된 SA를 적용하여 Park and Parker(1988)에서 소개된 그림 6의 16-point 디지탈 FIR 필터(파이프라인이 고려된 구조)의 스케쥴링/할당을 시도하여 변형된 SA의 효율성을 확인한다. 앞의 예에서 SA2

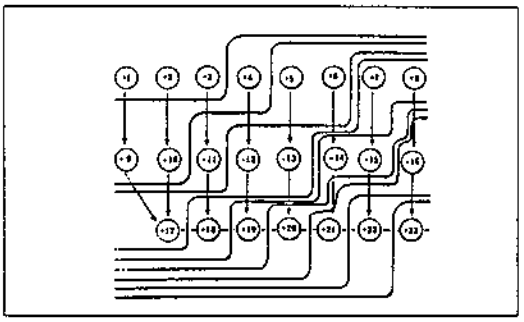

그림 6. FIR DF 예제의 CDFG

표 4. FIR 디지탈 필터 예제(n=23, non-pipelined, 시간슬롯수:10)

	SA	SA1	SE
소요시간	302	25.76	32.68
비용함수값	392	347	347
ALU 갯수	2	2	2

표 5. FIR 디지탈 필터 예제(n=23, pipelined, 시간슬롯수:18)

	SA	SA1	SE
소요시간	289	162.75	198.23
비용함수값	366	346	346
ALU 갯수	2	2	2

는 대개의 경우 SA1에 비해 시간의 단축 효과가 없는 것으로 관찰되고 있으므로 본절의 보고에서는 이 방법의 결과를 생략한다. 시간의 단축이 되지 않는 이유는 매번 계산은 빨라지지만 Metropolis 기준과의 차이 때문에 해의 이동 양태가 달라져 결과적으로 수렴에 필요한 반복수가 늘어나게 되기 때문이리라고 추측된다.

우선 파이프라인 구조가 아닐 경우 이 문제에 각 알고리듬을 적용했을 경우에 얻어지

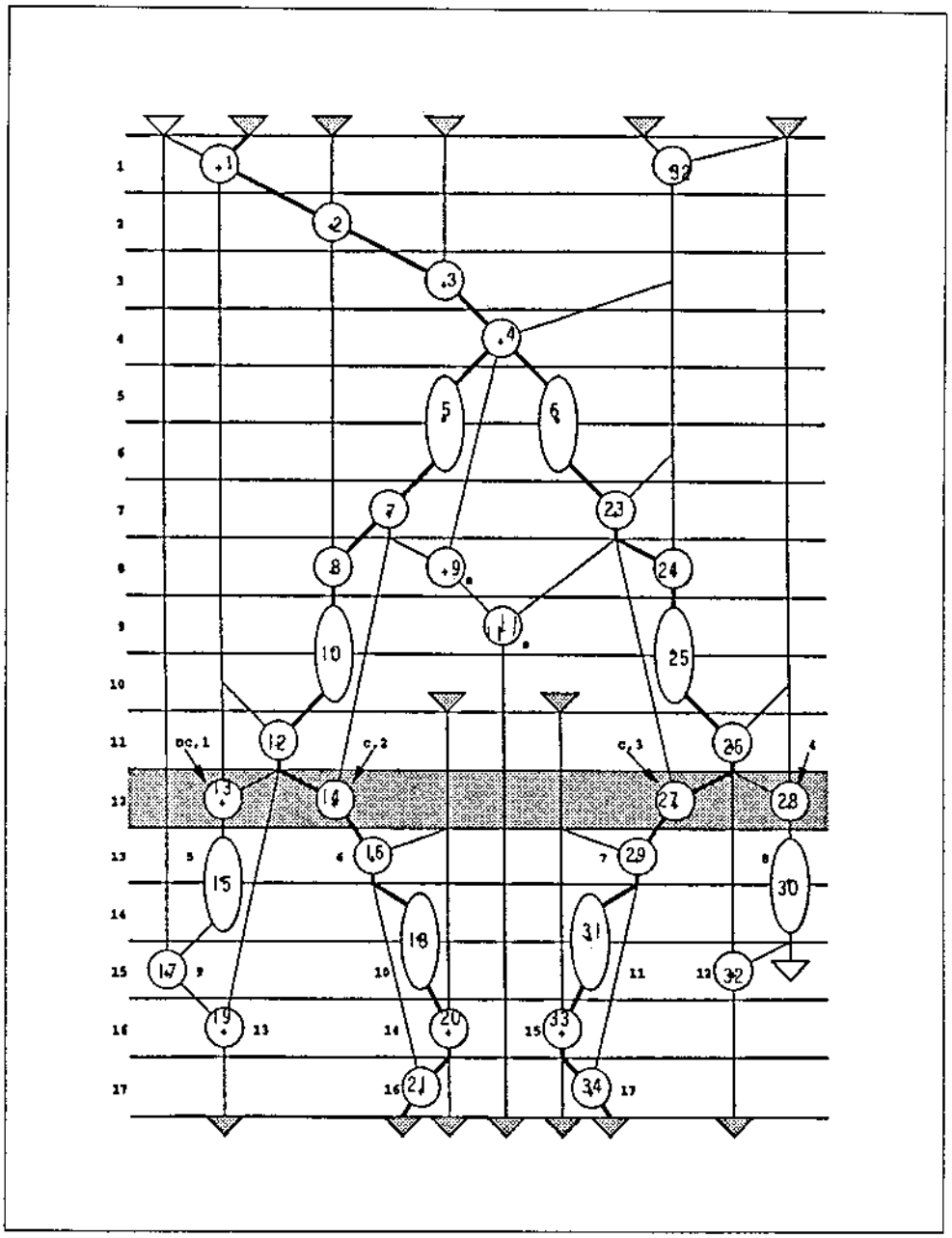

그림 7. 5th order DWF의 ASAP 스케줄

표 6. 5th 필터 예제(non-pipelined, 시간슬롯수:17)

	SA	SA1	SE
소요시간	282.41	172.08	803.92
비용함수값	193	185	185
ALU 갯수	(3,2)	(3,1)	(4,2)

표 7. 5th order 필터 예제(non-pipelined, 시간슬롯수:19)

	SA	SA1	SE
소요시간	277.88	149.2	458.69
비용함수값	205	203	209
ALU 갯수	(3,2)	(3,2)	(3,2)

는 최종비용 함수값과 걸린시간의 평균을 표 4에서 볼 수 있는데 SA1가 SA의 1/12의 시간에 11%가 향상된 해를 주는 것을 확인할 수 있고, SE의 79%의 시간에 같은 수준의 최종해를 주는 것을 볼 수 있다. 표 5는 L=3의 파이프라인 구조로 스케쥴/할당을 한 경우의 결과인데 SA1이 SA의 56%, SE의 82%의 시간에 5% 향상되거나 같은 수준의 해를 주는 것을 볼 수 있다.

5.6. 5th order digital wave filter

그림 7에서는 HLS의 방법들의 성능비교시에 표준적인 문제로 취급되고 있는 5차(5th order) 디지탈 파동 필터(digital wave filter : DWF)의 ASAP 스케쥴된 CDFG를 볼 수 있는데 총연산의 수가 34개이고 * 연산이 + 연산의 2배의 시간이 걸린다고 가정할 때 필요한 최저 시간슬롯의 수가 17개 임을 알 수 있다. 이 문제를 시간슬롯 수 17개인 비파인 프라인 구조로 스케쥴/할당 했을 경우의 결과를 표 6에서 볼 수 있는데 SA1가 SA에 비해 61%의 시간에 4%정도 향상된 해를 주고 있음을 알 수 있다. SE의 경우 1070초를 돌려서 이 값을 얻을 수 있었다. 표 7에서는 시간슬롯수를 19개까지 허용한 경우의 결과를 볼 수 있는데 역시 SA1의 우월성을 확인할 수 있다. 이 경에는 비용함수를 p1=10, p2=2, p3=2, p4=2로 두고 계산하여 표 6의 비용함수 값과는 차이가 있다.

6. 결론

SA는 범용성과 적용의 용이성 때문에 접근이 난해한 조합적 최적화 문제에 다양하게 적용되어 왔으나 앞에서 살펴본 바와 같이 아직까지 개선할 여지가 많이 있다. 본 논문에서는 SA에 내재된 문제점들을 살펴보고 SA의 개선에 관한 이론적 경험적인 연구들을 소개하고 SA의 변형을 통한 성능의 제고 방법을 제시하고 HLS에의 적용을 통한 성능의 비교 연구 결과를 간략히 소개하였다. SA의 수렴성을 유지하면서 변형된 SA1, SA2가 SA에서 얻을 수 있는 품질의 해를 보다 빠른 시간에 얻을 수 있음을 확인하였고 더우기 기존의 매우 효과적인 변형 방법중 하나인 SE를 능가하는 성능을 검증하였다. SA의 잠재적인 적용 가능성이 매우 큰 만큼 향후 보다 깊은 이론적인 연구와 광범위한 실험을 통한 알고리듬의 개선에 관한 연구에 매우 가치가 있다고 판단된다.

또한 HLS 문제에의 적용에서 보다 효과적인 자료구조와 해의 이동방법에 관한 연구가 계속되어야 하면 이를 바탕으로 최근에 대두되는 다양한 디자인의 관점들을 수용하여 SA

의 융통성을 활용한 포괄적이고 종합적인 도구를 만드는 연구가 매우 가치가 있을 것이다.

참 고 문 헌

[1] E. Aarts and J. Korest, *Simulated annealing and Boltzmann machines,* John Wiley & Sons, New York, 1989.

[2] S. Anily and F. Federgruen, "Simulated annealing methods with general acceptance probabilities," *J. Appl. Prob.,* v.24, pp. 657-667, 1987.

[3] R. Camposano and W. Wolf, *High-Level VLSI Synthesis*, Kluwer, Boston, 1991.

[4] D.P. Connors and P.R. Kumar, "Balance of recurrence order in time-inhomogeneous Markov chains with application to simulated annealing," *Probability in the Engineering and Informational Sciences,* " v.2, pp.157-184, 1988.

[5] D.P. Connors and P.R. Kumar, "Simulated annealing type Markov chins and their order balance equations," *SIAM J. Control and Optimization,* v.27, pp.1140-1461, 1989.

[6] S. Devadas and R. Newton, "Algorithms for hardware allocation in data path synthesis," An approach using simulated annealing," *ORSA Journal on Computing,* v.3, pp.275-287, 1991.

[7] P.C. Fetterolf and G. Anandalingam, "Optimizing Interconnection of local area network : An approach using simulated annealing," *ORSA Journal on Computing,* v.3, pp. 275-287, 1991.

[8] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. Pattern Recognition and Machine Intelligence,* v. PAMI-6, pp.721-741, 1984.

[9] B. Gidas, "Non-stationary Markov chains and convergence of the annealing algorithms," *J. Statistical Phys.,* v.39, pp. 73-131, 1985.

[10] C.T. Hwang, J.H. Lee, and Y.C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans, CAD,* v.10, pp.464-475, 1991.

[11] D.S. Johnson, C.R, Aragon, L.A. Mcgeoch, and C. Schevon, "Optimization by simulated annealing : an experimental evaluation ; Part Ⅰ. graph partitioning", *Operations Research,* v.37, pp.865-892, 1989.

[12] D.S. Johson, C.R. Aragon, L.A. Mcgeoch, and C. Schevon, "Optimization by simulated annealing : An experimental evaluation ; Part Ⅱ. graph partitioning", *Operations Research,* v.37, pp.865-892, 1989.

[13] B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Systems Technical Journ.,* v.49, pp.291-307, 1970.

[14] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing," *Science,* v.220, pp.671-680, May 1983.

[15] R.M. Kling and P. Banerjee, "ESP : Placement by simulated evolution," *IEEE Trans. Computer-Aided Design*, v.8, pp.245-256, 1989

[16] P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing : Theory and Applications*, Reidel, Boston, 1987.

[17] S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the travelling salesman problem," *Operations Research*, v.21, pp.498-516, 1973.

[18] M. Lundy, "Applications of the annealing algorithm to combinatorial problems in statistics," *Biometrika*, v.72, pp.191-8, 1985.

[19] P. Marwedel, "A new synthesis algorithm for MIMOLA software design," Proc. of the 23rd Design Automation Conf., New York, NY : ACM/IEEE, 1986, pp.271-277.

[20] M.C. Mcfarland, A.C. Parker, and R. Camposano, "The High-Level Synthesis of Digital Systems," Proceedings of the IEEE, v.78, n.2, Feb. 1990, pp.301-318.

[21] P. Michel, U. Lauther, and P. Duzy, *The Synthesis Approach to Digital System Design*, Kluwer, Norwell, 1992

[22] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and finite-time behavior of simulated annealing," *Advances in Applied Probability*, v.18, pp.747-771, 1986.

[23] S. Nahar, S. Sahni, and E. Shragowitz, "Simulated annealing and combianatorial optimization," in *Pro. 23rd Design Automation Conf.*, pp.293-299, 1986.

[24] N. Park and A.C. Parker, "Sehwa : A Software Package for Synthesis of Pipelines from Behavioral Specifications," IEEE Trans. on CAD, v.7, n.3, March 1988, pp.356-370.

[25] A.C. Parker, J. Pizarro, and M. Mlinar, "MAHA A Program for Datapath Synthesis," Proc. of the 23rd Design Automation Conf., New York, NY : ACM/IEEE, 1986, pp.461-466.

[26] P.G. Paulin and J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs," IEEE Trans. on CAD, v.8, n.6, Jun. 1989, pp.661-678.

[27] J. Rose and W. Klebsch, and J. Wolf, "Temperature measurement and equilibrium dynamics of simulated annealing placements," *IEEE Trans. Computer-Aided Design*, v.9, pp.253-259, 1990.

[28] Y.G. Saab and V.B. Rao, "Combinatorial optimization by stochastic evolution," *IEEE Trans. Computer-Aided Design*, v.10, pp.525-535, 1989.

[29] C. Sechen and A. Sangiovanni-Vincentelli, "The Timberwolf placement and routing packages," *IEEE J. Solid-State Circuits*, v.SC-20, pp.510-522, 1985.

[30] J.M. Stern, "Simulated annealing with a temperature dependent penalty function," *ORSA Journal on Computing*, v.4, pp. 311-319, 1992.

[31] J.N. Tsitsiklis, "Markov chain with rare transitions and simulated annealing,"

Mathematics of Operations Research, v. 14, pp.70-90, 1989.

[32] D.F. Wong, H.W. Leong, and C.L. Liu, *Simulated annealing for VLSI design*, Kluwer, Boston, 1988.