Real-time Scheduling for Multi Headed Placement Machine

Masri Ayob And Graham Kendall

Automated Scheduling, Optimisation and Planning (ASAP) Research Group,
University of Nottingham, School of Computer Science and IT
Nottingham NG8 1BB, UK
E-mail: mxa|gxk@cs.nott.ac.uk

Abstract

This paper proposes a methodology for real-time scheduling to sequence the pickup and placement of component on multi headed placement machines in printed circuit board (PCB) assembly. The latest technology of surface mount device (SMD) placement machines have a smart feeder carrier that automatically identifies the exact location of each component type on the feeder slot, detects component's missing from the component feeders (and continues working with other component types) and allows a component to be reloaded during a pick and place operation without stopping the operation. Assuming that the components on the feeder carrier may be misplaced or some of the required components are missing, we generate an initial schedule using a greedy constructive heuristic by only considering the available placement points. The initial solution can immediately be used to assemble components for the first PCB. While the placement machine is assembling components, we employ the CPU free time (whilst the robot arm is moving) to improve the initial schedule by using a randomised hill climbing search technique. Thus, the subsequent PCB's will use the improved schedule. Our experimental result on two data sets show that we gain 58.79% and 76.69% (respectively) improvement on assembly cycle time over the initial schedule.

1 Introduction

The placement rate of an SMD (surface mount device) placement machine can be optimised by considering significant factors such as the placement head travel, the placement sequence and the position of components in the feeder bank. A lot of work has been done on improving the efficiency of SMD placement machines such as [1,2,3,4]. However, most previous work involves an offline scheduler that is ineffective if there is a change in the resources such as a component being missing from the feeder carrier. Hence, in this work, we propose a real-time scheduling approach that can continuously generate a new schedule and can improve upon the current schedule using spare CPU time during pick and place operations. That is, we use CPU time not required by the robot arm as it is in the process of carrying out an operation (such as moving, placing or picking up component).

2 SMD Placement Machine

The SMD placement machine is used to assemble components onto a PCB (printed circuit board). SMD placement machines are categorised into five categories, based on their specification and operational methods. These are dual delivery, multi-station, turret style, multi-head and a sequential SMD placement machine [5]. In general, each SMD placement machine has a feeder carrier (sometimes called a feeder magazine), a PCB table (or worktable), head(s), nozzle(s) (or gripper(s)) and a tool magazine. The PCB table, the feeder carrier and the head(s) can either be moveable or fixed. A typical feeder carrier consists of either several tape reels or vibratory ski slope feeders, or both [1]. The feeder reels, that is tapes holding electronic components; or vibratory ski slope feeders are positioned in the feeder slots according to the arrangement given by the feeder setup [5]. Usually, larger components are supplied by tray feeders [6]. Some machines allow a tray to be placed into the machine feeding area whilst others have an automatic tray-handling unit. Normally, the components are transported from the feeders to the placement position on the PCB using vacuum nozzle(s) that are placed at the end of the head(s) [6]. The PCB table is required to support the PCB, in a locked position and locating the required placement point for the placement operation. Some machines may have more than one PCB table. The PCB table(s) can be a stationary, use a conveyor system, or use an X-Y motion table [5]. Figure 1 is an example of multi head placement machine.

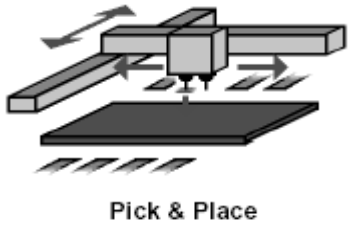

Figure 1 An example of pick place multi head placement machine (taken from [6]).

3 Real-time Scheduling

A good off-line scheduling algorithm can produce a very good solution. However, they are ineffective if there is a change in the resources after the schedule is generated. For example, if the component feeders are misplaced by the machine's operator or if some components are missing from the feeder carrier, then the solution given by an off-line scheduling is not feasible. Hence, the placement machine will be idle, waiting for a new schedule. The duration of the idle state will be dependent on the time taken by the off-line scheduler. In this work, we propose a real time scheduling approach that incorporates an interrupt feature.

Since some of the latest technology in placement machines such as MY15 and MY19 [7] have a smart feeder carrier that can automatically detect the exact location of each component type on the feeder slot and the availability of components on the feeder slots, we propose using this feature to enhance the scheduling technique. The information about the missing and reloaded component types will be managed by two interrupt service routines (ISR). There are two interrupt events in this problem, the missing component's interrupt and the reloaded component's interrupt. Generally, the interrupts can be triggered by a sensor using a positive and a negative edge triggering signals. When the sensor detects a missing component type it sends an interrupt signal to the central processing unit (CPU) of the placement machine to call the missing component's ISR (named as ISR1). Similarly, when the sensor detects a newly reloaded component, it will send an interrupt signal to the CPU of the placement machine to execute the reloaded component's ISR (named as ISR2). ISR1 only needs to store the name of the missing component type in a missing component file such that the scheduler program can use this file to identify the missing component type. Similarly, ISR2 will store the name of a new reloaded component type into the component's reloaded file. The ISR2 will also remove the name of component type that has just been reloaded from the missing component file. The scheduler will automatically remove the contents of the component's reloaded file after these component types have been inserted into the schedule.

Generally, a surface mount placement machine is a microprocessor based electronic device. The microprocessor, or CPU, is the machine's controller. In a placement machine, the CPU will control the movement of the robot arm to pick and place components and carry out calculations such as component recognition, alignment etc. To perform the pick and place operations, the current schedule will be transferred to the robot arm to control its movement. While the robot arm is moving to pickup a component (or perhaps some components), the CPU is in an idle state. After picking up the component(s), the robot arm will interrupt the CPU to acknowledge job completion. Then the CPU will send other control signal to the robot arm. When the robot arm is moving to place a component (for example), the CPU may do work for component recognition and alignment then may be in an idle state again. Generally we can say that the CPU is always in an idle state while the robot arm is moving or picking and placing components. Since the robot arm is normally an interrupt driven I/O device, we can make use of the CPU free time to carry out other tasks. When the robot arm completes its current task, it will send an interrupt request to the CPU. In responding to this interrupt request, the CPU will suspend the current running task (the task we will run during CPU free time). The appropriate interrupt service routine (ISR) will be called to service the request (which may include sending other appropriate control data). When the ISR completes the task, the CPU will automatically continue the suspended task. Information about interrupts and how they work can be found in many operating system's textbooks such as [8,9].

In this work we introduce a real-time scheduler that utilises the CPU free time. We assume the components are randomly assigned to the feeder slots (or the operator may have misplaced the components). We generate a greedy constructive heuristic for a pick and place on multi headed placement machine that randomly assigns components to feeder slots, a placement point to a sub tour, a nozzle for pickup and placement, a pickup sequence, a placement sequence and sub tour sequences. The initial schedule (generated by the greedy constructive heuristic) can immediately be used to perform the pick and place operations. In our proposed framework, we will start the pick and place operation immediately, once the initial schedule is ready. The first PCB will be processed using the initial schedule. While the robot arm is moving, picking and placing components, we will employ the CPU spare time to improve our initial schedule. We apply a local search, using a randomised hill climbing method, that only accepts an improved solution. There are so many factors involved in determining the quality of the solution such as the grouping of placement points to a sub tour, nozzle assignment, pickup and placement sequencing etc. Usually, optimising one factor will increase the cost of another factor(s). For example, if we optimise the grouping of the placement point to a sub tour, we may have to pay a cost in pickup time. In our approach, we apply swaps between placement/pickup points in a sub tour or among the sub tours. The details of the swapping method is discussed in section 5. While the CPU is running the optimisation software (our local search) to find a better schedule, the robot arm may interrupt the CPU whenever it needs attention. Therefore, our optimisation software actually runs without incurring any cost to the assembly cycle time of the PCB currently being processed. Thus, without paying any cost, we can improve the initial solution. By using this concept, the pickup and placement schedule is always updated and we may obtain a near optimal solution if the placement machine produces a large number of PCBs. Moreover, if the component feeder runs out of components, a new schedule can be generated easily without stopping the production. The latest technology placement machine also allows components to be reloaded during production [7].

After components been reloaded, a new schedule must be generated and again our approach can solve this problem.

4 The Scheduling Model

In order to evaluate the quality of a schedule, it is necessary to measure the resulting machine placement time. In this work, a placement time function that is an objective function for a pick place multi headed placement machine is developed to evaluate the proposed approach. We model a pick place multi headed placement machine that has a single head equipped with *G* number of nozzles, fixed PCB table and fixed feeder carrier (refer Figure 1). This type of placement machine was categorised in [5] as a multi-head placement machine that has the arm and head which can moves in X-Y axis simultaneously but the feeder carrier and PCB table are fixed. Hence, our objective function only needs to consider minimising the total assembly cycle time by minimising the traveling distance of the robot to perform pick and place operation. The following notations are used to describe the scheduling model:

CT	:	the cycle time to assemble all components;
N	:	the number of placement points on the PCB;
Q	:	the total number of available placement points to be scheduled, where Q ≤ N.
K	:	the number of component types (each feeder slot holds multiple copies of one component type);
G	:	the number of nozzles per head;
T	:	the total number of sub tours;
M	:	the total number of feeder slots where K ≤ M;
$c(j,k)_{x,y}$	:	the X,Y coordinate on the PCB which will have a component placed there in the k^{th} placement sequence of the j^{th} sub tour;
V_r	:	the robot speed (average);
T_p	:	the time for picking up a component;
T_i	:	the time for placing a component;
I(j,k)	:	the time taken for the robot arm to travel and place a component in the k^{th} placement sequence of the j^{th} sub tour;
P(j,k)	:	the time taken for the robot arm to travel and pick a component in the k^{th} pickup sequence of the j^{th} sub tour;
$h_p(j,k)$	:	the nozzle used to pick a component in the k^{th} pickup sequence of the j^{th} sub tour;
$h_b(j,k)$	:	the nozzle used to place a component in the k^{th} placement sequence of the j^{th} sub tour;
s(i)	:	the i^{th} slot distance referring to the origin of feeder slot, s(0) where s(0)=0 and i<K;
r(j,k)	:	the slot distance for the k^{th} pickup sequence of the j^{th} sub tour;
d(j,k)	:	The $\max\{\|d(j,k)_x\|, \|d(j,k)_y\|\}$ where *x* and *y* is the X,Y robot traveling distance to place a component in the k^{th} placement sequence of the j^{th} sub tour, where the distance is measured as a chebychev distance ;
m(j,k)	:	The $\max\{\|m(j,k)_x\|, \|m(j,k)_y\|\}$ where *x* and *y* is the X,Y robot traveling distance to pick a component in the k^{th} pickup sequence of the j^{th} sub tour, where the distance is measured as a chebychev distance ;
F	:	the gap between each feeder slot;
L	:	the gap between each nozzle.

The objective function is to minimise the assembly cycle time, CT:

$$CT = \min \sum_{j=1}^{T} \left[\sum_{k=1}^{G} P(j,k) + \sum_{k=1}^{G} I(j,k) \right] \quad (1)$$

where:

$$P(j,k) = \left(\frac{m(j,k)}{V_r} + T_p\right) * z(j,k);$$

$$I(j,k) = \left(\frac{d(j,k)}{V_r} + T_i\right) * z(j,k);$$

$$\sum_{k=0}^{G-1} z(j,k) < G ; \qquad T = \begin{cases} Q/G \text{ if } Q\%G=0; \\ 1 + \ Q/G \text{ if } Q\%G\neq 0; \end{cases}$$

$m(j,k)=\max\{|m(j,k)_x|, |m(j,k)_y|\}$; As the robot arm moves concurrently in X-Y axis, we use a chebychev distance.

$d(j,k)=\max\{|d(j,k)_x|, |d(j,k)_y|\}$;

$$z(j,k) = \begin{cases} 1 : \text{if there is a nozzle assigned to pick or place a component in the } k^{th} \text{ sequence of } j^{th} \text{ sub tour.} \\ 0 : \text{otherwise} \end{cases}$$

$$\sum_{j=0}^{K-1} g_{ij} \le 1, \forall i$$

$$s(i) = \sum_{j=0}^{K-1} \left[j * F * g_{ij} \right]$$ assuming that components are assigned to feeder slots adjacent to each other.

$$b_i(j,k) = \begin{cases} 1 : \text{if component type } i \text{ involve in the } k^{th} \text{ pickup sequence of the } j^{th} \text{ sub tour,} \\ 0 : \text{otherwise} \end{cases}$$

$$r(j,k) = \sum_{i=0}^{K-1} \left[s(i) * b_i(j,k) \right] ; \qquad \sum_{i=0}^{K-1} b_i(j,k) \le 1, \forall j, \forall k$$

$$g_{ij} = \begin{cases} 1 : \text{if component type } i \text{ is assigned to feeder slot } j, \\ 0 : \text{otherwise} \end{cases}$$

$h_p(j,k) = \{0,1,2,\ldots(G-1)\}$;

s.t.

$h_p(j,k) \neq h_p(j,m)$ and

$j=\{0,1,2,\ldots,(T-1)\}$, $k = \{0,1,2,\ldots(G-1)\}$;

$h_b(j,k) = \{0,1,2,\ldots(G-1)\}$;

s.t.

$h_b(j,k) \neq h_b(j,m)$ and

$j=\{0,1,2,\ldots,(T-1)\}$, $k = \{0,1,2,\ldots(G-1)\}$;

$$m(j,k)_x = \begin{cases} r(j,k) - h_p(j,k)*L & \text{if } j=k=0; \\ [r(j,k) - h_p(j,k)*L] - [c(j-1,G-1)_x - h_b(j-1,G-1)*L] & \text{if } k=0,\ j>0; \\ [r(j,k) - r(j,k-1)] - [(h_p(j,k) - h_p(j,k-1))*L] & \text{if } k>0; \end{cases}$$

$$m(j,k)_y \begin{cases} 0 - [c(j-1,G-1)_y] & \text{if } k=0,\ j>0; \\ 0 & \text{otherwise} \end{cases}$$

$$d(j,k)_x = \begin{cases} [c(j,k)_x - r(j,G-1)] - [(h_b(j,k) - h_p(j,G-1))*L] & \text{if } k=0; \\ [c(j,k)_x - c(j,k-1)_x] - [(h_b(j,k) - h_b(j,k-1))*L] & \text{if } k>0; \end{cases}$$

$$d(j,k)_y = \begin{cases} c(j,k)_y & \text{if } k=0; \\ [c(j,k)_y - c(j,k-1)_y] & \text{if } k>0; \end{cases}$$

5 Implementation

Initially, the algorithm will read all the PCB data and identify the PCB points that have to be scheduled at the moment (there could be PCB points that cannot be scheduled at the moment because of components being missing from the feeder carrier; these will be marked and inserted into a later schedule, whenever their components are reloaded onto the feeder carrier).

Our random constructive heuristic begins by randomly assigning each PCB point to a sub tour. The size of a sub tour depends on the number of available nozzles per head and the number of available placement points. Each sub tour consists of a set of pickup points and a set of placement points. The nozzles are indexed from 0 (at most left side) to *G-1* (at the most right side). For each sub tour, the pickup and placement nozzles are randomly allocated. However, we must ensure that each placement point is assigned the correct component type. The pickups and placements sequence in each sub tour are also generated randomly and are independent of each other. A sub tour begins by picking up a set of components in a random sequence, then placing these components onto the PCB, also in a random placement sequence (but the correct PCB point will receive a component of the correct type). Finally, the constructive heuristic randomly sequences the sub tours to complete the overall pickup and placement schedule to generate an initial solution.

To improve the initial solution, we apply a local search with a randomised hill climbing method. In the local search we only accept improved solutions. We propose six swapping methods, which are:

1) Swap the pickup sequence in a sub tour. For each sub tour we perform a *G* number of swaps in the component pickup sequence. In a sub tour, each i^{th} component pickup sequence will be swapped with a randomly selected j^{th} component pickup sequence.
2) Swap the placement sequence in a sub tour. For each sub tour we perform a *G* number of swaps in the component placement sequence. In a sub tour, each i^{th} component placement sequence will be swapped with a randomly selected j^{th} component placement sequence.
3) Swap the pickup nozzles in a sub tour. For each sub tour we perform a *G* number of nozzle swaps of pickup operations. In a sub tour, the nozzle's used in the i^{th} component pickup sequence will be swapped with a randomly selected nozzle's used in the j^{th} component pickup sequence. If the swapping operations involve two different component types, then we modify the appropriate placement nozzle in the sub tour such that the PCB points will received the correct component type.
4) Swap the placement nozzles in a sub tour. For each sub tour we perform a *G* number of nozzle swaps in placement operations. In a sub tour, the nozzle's used in the i^{th} component placement sequence will be swapped with a randomly selected nozzle's used in the j^{th} component placement sequence. If the swapping operations involve two different component types, then we modify the appropriate pickup nozzle in the sub tour such that the component will be picked with the right nozzle.
5) Swap the PCB placement points among sub tours. In this swapping method, we perform a *Q* number of swapping operations where *Q* is a total number of available placement points. Each placement point in a sub tour will be swapped with other placement points in other sub tour (chosen at random). If the swapping operations involve two different component types, then we modify the appropriate pickup component in the appropriate sub tour such that the pickup components are valid in both sub tours.
6) Swap the sub tours sequence order. We perform an *S* number of swapping operations where *S* is a total number of complete sub tours.

Since there is no clue or good strategy on selecting the best swapping method at any point, we use a random selection approach that is, the next local search swap is chosen by generating a random number between 1 and 6. A random number, generated between 1 and 6. This process continues until completing the number of iterations which is set at 100 for our experiments. In our formulation, we consider the robot makes a positive traveling distance when it moves in increasing X or Y coordinate and a negative traveling distance otherwise.

6 Testing and Results

In our experiments we assume that the feeder carrier and PCB table are positioned close to each other in order to minimise the robot arm travel distance [3]. We also assume that the gap between the feeder carrier and the PCB board is 10 unit length, the nozzle's gap is equal to the size of feeder slot (chosen as 4 unit length) and all components are the same size (we ignore the nozzle size selection). In the experiment we modeled the multi headed placement machine as a head equipped with 4 nozzles. Since we modeled the placement machine that

has fixed PCB table and fixed feeder carrier, we only apply five of the seven factors (Table 1) of the parameters used in [3]. The other two factors are feeder carrier speed and PCB table speed are not required since we model the fixed feeder carrier's and fixed PCB table's SMD placement machine.

Factors	**Levels (low/high)**
Number of assembly points (N)	20/100
Number of component types (K)	8/20
Length of PCB (BL)	100 (unit distance)
Width of PCB (BW)	50 (unit distance)
Speed of robot (Vr)	10 (unit distance/unit time)

Table 1: Experimental parameters

The pick up, T_p and placement time, T_i are both set as 0.5 unit time. For the purpose of generating the random placement points, we set the length and the width of the PCB as 100 and 50 unit length, respectively, such that the random placement points fall within these limits. The placement points are generated randomly. Since we use a constructive heuristic that randomly groups the PCB points into sub groups (each sub group will be assigned to a sub tour), randomly assigns component feeders to slots, randomly sequences the component pickup and placement, randomly assigns nozzles for pickup and placement and randomly sequence the sub tours, we can use the same data set for different runs (every run will generate different initial solutions). Hence, two data sets that are randomly created are adequate to demonstrate our approach. Data set 1 has 20 assembly points consisting of 8 component types whilst data set 2 has 100 assembly points consisting of 20 component types. We chose these two data sets in order to show that our method can work with a larger problem size as well as a smaller problem. To simulate the assembly cycle time, we set the speed of the robot arm, V_r as 10 unit distance/unit time (as shown in Table 1). We assume that all components use the same nozzle type and the speed of robot arm is constant for all component types. We ran the experiment using an AMD Athlon XP1700+ PC with 1.47GHz speed and 240 MB RAM. The computational results are shown in Table 2 and Table 3 and are obtained from 10 runs for each data set.

Test	Constructive Heuristic		Improvement Heuristic		Improvement
	CT (unit time)	P(seconds)	CT (unit time)	P(seconds)	(%)
1	52.89	0.078	31.89	8.844	65.85
2	**54.23**	**0.062**	**28.20**	**10.093**	**92.30**
3	55.61	0.063	35.16	9.281	58.16
4	48.06	0.063	28.56	12.047	68.28
5	**49.11**	**0.078**	**38.54**	**9.031**	**27.43**
6	46.79	0.078	31.10	12.312	50.45
7	44.48	0.078	29.23	11.015	52.17
8	52.65	0.063	32.98	7.953	59.64
9	54.78	0.062	34.48	10.454	58.87
10	49.21	0.078	31.81	11.203	54.70
Average	**50.78**	**0.070**	**32.20**	**10.220**	**58.79**

Note: CT = Assembly cycle time (unit time)
P = Computation time (seconds)

Table 2: An experimental result of ten runs on data set 1 (N=20, K=8)

Test	Constructive Heuristic		Improvement Heuristic		Improvement (%)
	CT (unit time)	P(seconds)	CT (unit time)	P(seconds)	
1	**419.15**	**0.250**	**212.29**	**75.187**	**97.44**
2	387.52	0.234	209.15	80.266	85.28
3	391.46	0.234	222.40	76.797	76.02
4	413.59	0.219	236.86	79.109	74.61
5	397.95	0.219	217.60	76.312	82.88
6	394.26	0.235	221.05	87.422	78.36
7	426.06	0.250	246.10	71.375	73.12
8	363.50	0.235	218.86	59.359	66.09
9	**367.85**	**0.250**	**236.54**	**70.297**	**55.51**
10	379.00	0.234	213.63	80.907	77.41
Average	**394.03**	**0.236**	**223.45**	**75.703**	**76.67**

Note: CT = Assembly cycle time (unit time)
P = Computation time (seconds)

Table 3: An experimental result of ten runs on data set 2 (N=100, K=20)

The results in Table 2 and Table 3 show that our greedy constructive heuristic can generate an initial solution in a short time of about 0.070 seconds for data set 1 and 0.236 seconds for data set 2. For data set 1, the initial solutions have improved about 58.79% after 10.22 seconds (average result) whilst a more complex data set gained 76.67% after 75.703 seconds (average result). The results show that a good initial solution does not guarantee a good final solution and a bad initial solution does not mean that we cannot obtain a good solution. For example, in data set 1, test 2 started with a CT=54.23 unit time as an initial solution and finished with a CT=28.20 unit time as a final solution whilst test 5 started with CT=49.11 unit time but finished with CT=38.54 unit time. In data set 2, test 1 started with a CT=419.15 unit time as an initial solution and finished with a CT=212.29 unit time as a final solution whilst test 9 started with CT=367.15 unit time but finished with CT=236.54 unit time. Generally, we can see that a quality of a final solution is not dependent on the quality of an initial solution.

7 Conclusions

As the CPU of the SMD placement machine is always in the idle state while the robot arm is moving, picking and placing components and the robot arm is normally an interrupt driven I/O device, we can make use of the CPU free time to improve the initial schedule. Hence, in this work we introduce a real-time scheduling approach that employs the CPU free time to improve the initial schedule. We use a greedy constructive heuristic to generate an initial solution then apply a randomised hill climbing method to improve the initial schedule. Results shows that the CT improved by 58.79% (data set 1) and 76.67% (data set 2) over the initial schedule. Results also indicate that generally the quality of the final schedule is not dependent on the initial schedule.

References

[1] J. Ahmadi, S. Grotzinger and D. Johnson, "Component allocation and partitioning for a dual delivery placement machine", *Operations Research*, vol. 36, pp. 176-191, 1988.

[2] K. P. Ellis, J. E. Kobza, and F. J. Vittes, "Development of placement time estimator function for a turret style surface mount placement machine", *Robotic and Computer Integrated Manufacturing*, vol. 18, pp. 241-254, 2002.

[3] Y. -C. Su, C. Wang, P. J. Egbelu and D. J. Cannon, "A dynamic point specification approach to sequencing robot moves for PCB assembly", *Int. J. Computer integrated Manufacturing*, vol. 8, no. 6, pp. 448-456, 1995.

[4] S. Deo, R. Javadpour and G. M. Knapp, "Multiple setup PCB assembly planning using genetic algorithms", *Computers & Industrial Engineering*, vol. 42, pp. 1-16, 2002.

[5] M. Ayob, P. Cowling and G. Kendall, "Optimisation for surface mount placement machines", *Proc. of the IEEE ICIT'02*, Bangkok, 11-14 Dec. 2002, pp. 498-503.

[6] B. Bentzen, "SMD placement", *the SMT in FOCUS*, Nov. 28, 2000, url: http://www.smtinfocus.com/PDF/SMD_placement.pdf (Sept. 25, 2002).

[7] MYDATA automation worldwide brochure, 2002.

[8] A. Tanenbaum, *Modern operating systems*. Prentice-Hall, New Jersey, USA, 1992.

[9] A. Silberschatz, P. Galvin and G. Gagne, *Applied operating system concepts*. John Wiley & Sons, USA, 2000.