OMEGA Int J of Mgmt Sci, Vol 17, No 6, pp 551–557, 1989
Printed in Great Britain All rights reserved

Simulated Annealing for Permutation Flow-Shop Scheduling

IH OSMAN

Imperial College, London, UK

CN POTTS

Faculty of Mathematical Studies, University of Southampton, UK

(Received February 1989, in revised form June 1989)

The problem of scheduling jobs in a flow-shop is considered. The job processing order must be the same on each machine and the objective is to minimize the maximum completion time. Simulated annealing is proposed as a heuristic to obtain approximate solutions. Extensive computational tests with problems having up to 20 machines and 100 jobs show simulated annealing to compare favourably with known constructive heuristics and with descent methods.

Key words—scheduling, flow-shop, heuristics, simulated annealing

1. INTRODUCTION

THE PERMUTATION FLOW-SHOP problem may be stated as follows. Each of n jobs is to be processed on machines $1, \dots, m$ in that order. The processing time p_{ij} of job i ($i = 1, \dots, n$) on machine j ($j = 1, \dots, m$) is given. At any time, each machine can process at most one job and each job can be processed on at most one machine. Preemption is not allowed. The sequence in which the jobs are to be processed is the same for each machine. The objective is to find a sequence of jobs which minimizes the maximum completion time $C_{\max}$.

For the case of two machines, the efficient algorithm of Johnson [6] solves the problem. However, for three or more machines it is generally conjectured that a branch-and-bound algorithm is needed to obtain an optimal solution since the problem is NP–hard [4, 8]. The most effective of the branch-and-bound algorithms is that of Potts [13]. Although this algorithm usually solves three-machine problems using only moderate computer resources, large search trees are commonly generated when there are four or more machines and over 20 jobs. For practical purposes, therefore, it is often more appropriate to apply a heuristic method which generates an approximate solution at relatively minor computational expense.

Currently available heuristics for the permutation flow-shop problem may be classified as either constructive or improvement methods. A constructive heuristic builds a sequence of jobs so that once a decision is taken it cannot be reversed. On the other hand, an improvement heuristic starts with any sequence of jobs and then attempts to decrease the value of the objective by amending the sequence. In a descent method, only sequences which decrease the value of the objective function are accepted for further consideration. Clearly, an improvement method can be applied to the sequence obtained from a constructive heuristic.

Several constructive heuristics for the permutation flow-shop problem are proposed in the literature [2, 3, 11, 12]. Dannenbring [3] also proposes a descent method in which adjacent job interchanges are attempted. Computational results [3, 11, 14] show that the constructive method of Nawaz *et al.* [11] is currently the best available.

A new technique called simulated annealing has recently received much attention. Essentially, it is a randomized improvement method which allows solutions which increase the value of the objective to be accepted with a certain probability. In this paper, we investigate the application of simulated annealing to permutation flow-shop scheduling. Section 2 reviews some of the currently available constructive heuristics and discusses possible descent methods. Section 3 describes how simulated annealing heuristics are obtained from descent methods. Computational results comparing simulated annealing with constructive heuristics and descent methods are given in Section 4. Section 5 contains some concluding remarks.

2. HEURISTIC METHODS

2.1 Constructive heuristics

The heuristic of Palmer [12] computes the slope index s_i of each job i using

$$s_i = \sum_{j=1}^{m} (2j - m - 1)p_{ij}/2$$

A schedule is obtained by sequencing the jobs in non-increasing order of s_i. The time requirement for the heuristic is $O(n(m + \log n))$.

Campbell *et al.* [2] and Dannenbring use Johnson's two-machine algorithm to generate sequences. Campbell *et al.* construct problem k $(k = 1, \ldots, m-1)$ so that job i $(i = 1, \ldots, n)$ has processing times

$$\sum_{j=1}^{k} p_{ij} \quad \text{and} \quad \sum_{j=m-k+1}^{m} p_{ij}$$

on the two machines respectively. Each of the $m - 1$ sequences that are generated by applying Johnson's algorithm are evaluated with respect to the original m-machine problem and the one which yields the smallest maximum completion time is selected. Dannenbring similarly solves a two-machine problem in which the processing times of job i $(i = 1, \ldots, n)$ on the two machines are

$$\sum_{j=1}^{m} (m - j + 1)p_{ij} \quad \text{and} \quad \sum_{j=1}^{m} jp_{ij}$$

respectively. The time requirement for the heuristic of Campbell *et al.* is $O(mn(m + \log n))$, whereas Dannenbring's heuristic requires $O(n(m + \log n))$ time.

In the heuristic of Nawaz *et al.*, the jobs are first listed in non-increasing order of their total processing requirements $\Sigma_{j=1}^{m} p_{ij}$. A sequence is constructed by introducing one job at a time from the list into the current partial sequence. For the general step of the method in which the current partial sequence is $(\sigma(1), \ldots, \sigma(s))$, each of the $s + 1$ possibilities $(i, \sigma(1), \ldots, \sigma(s))$, $(\sigma(1), i, \sigma(2), \ldots, \sigma(s)), \ldots, (\sigma(1), \ldots, \sigma(s), i)$ for introducing the first unscheduled job i of the list into the partial sequence is considered and one which yields the smallest maximum completion time for this $(s + 1)$-job problem is chosen to be the new current partial sequence. The procedure is repeated until the current partial sequence contains n jobs. This heuristic requires $O(mn^3)$ time.

2.2 Descent methods

As pointed out above, a descent method repeatedly attempts to construct an improved sequence from a current sequence. Included in a specification of the method should be a precise statement of how the current sequence is amended to give a new sequence. We define a neighbourhood to be the set of sequences which can be generated from the current sequence. Our research concentrates on the two following neighbourhoods. Firstly, if $(\sigma(1), \ldots, \sigma(n))$ is the current sequence, then a new sequence is obtained by selecting the jobs in positions h and i, where $h < i$, and interchanging them in position to give the sequence $(\sigma(1), \ldots, \sigma(h-1)$, $\sigma(i), \sigma(h+1), \ldots, \sigma(i-1), \sigma(h), \sigma(i+1), \ldots$, $\sigma(n))$. We refer to this as the interchange neighbourhood. For this neighbourhood, each sequence has $n(n-1)/2$ neighbours given by the various possibilities for h and i. Dannenbring proposes a descent method based upon an adjacent job interchange neighbourhood in which $h = i - 1$. Since the results of Turner and Booth [14] indicate that this method is inferior to the heuristic of Nawaz *et al.*, we prefer our interchange neighbourhood which allows non-adjacent interchanges. Secondly, a shift neighbourhood is considered, which may be subdivided into forward and backward shift. A forward shift neighbour is obtained by selecting positions h and i, where $h < i$, removing job $\sigma(h)$ from its current position and reinserting it in position i, with jobs $\sigma(h+1), \ldots, \sigma(i)$ moving forward in the sequence by one position, to give $(\sigma(1), \ldots, \sigma(h-1), \sigma(h+1), \ldots, \sigma(i)$,

$\sigma(h), \sigma(i+1), \ldots, \sigma(n))$. A backward shift neighbour is similarly defined for $h > i$ as the sequence $(\sigma(1), \ldots, \sigma(i-1), \sigma(h), \sigma(i), \ldots, \sigma(h-1), \sigma(h+1), \ldots, \sigma(n))$. The shift neighbourhood consists of all $n(n-1)$ forward and backward shift neighbours.

In addition to defining the neighbourhood, it is necessary to specify the order in which neighbours are searched. An ordered search is usually adopted in which all possible values of h and i are considered after which the same cycle of values is repeated. Thus, an ordered search of the interchange neighbourhood would typically examine (h, i) values in the order $(1, 2)$, $(1, 3), \ldots, (1, n), (2, 3), \ldots, (n-1, n)$, whereas for the shift neighbourhood, (h, i) values would be examined in the order $(1, 2)$, $(1, 3)$, $\ldots$, $(1, n), (2, 1), (2, 3), \ldots, (n-1, n)$. It would also be possible to perform a random search of the neighbourhood by selecting a neighbour at random. For the interchange neighbourhood, a random search would select h and i with $h, i \in \{1, \ldots, n\}$ and $h < i$, at random and for the shift neighbourhood it would randomly select h and i with $h, i \in \{1, \ldots, n\}$ and $h \neq i$.

The starting sequence can either be chosen arbitrarily (for example the sequence $(1, \ldots, n)$) or it can be found by the application of a constructive heuristic. The latter approach is sometimes adopted to reduce the computation time spent on the descent method.

The descent method selects its starting sequence σ, generates a neighbour σ', evaluates the objective values $C_{\max}(\sigma)$ and $C_{\max}(\sigma')$ for the two sequences and computes

$$\Delta = C_{\max}(\sigma') - C_{\max}(\sigma) \qquad (1)$$

If $\Delta < 0$, then σ' is accepted as the current sequence. Alternatively, when $\Delta \geqslant 0$, σ is retained as the current sequence. In either case, a neighbour of the current sequence is again generated and the process is repeated. The search usually continues until a local minimum is found. This is detected when all neighbours of the current sequence are searched without improving upon the objective value.

Unfortunately, there is no guarantee that a descent method will find a local minimum in polynomial time.

3. SIMULATED ANNEALING

Simulated annealing is a heuristic which attempts to overcome the disadvantage inherent in descent methods that no further search for a global minimum is performed after a local minimum is detected (unless the method is repeated using a different starting sequence). The heuristic follows the same basic steps which are used in descent with one exception: sometimes a new sequence σ' is accepted as the current sequence, even though its objective value exceeds that of the old sequence σ, i.e., (1) yields $\Delta > 0$. More precisely, when $\Delta \leqslant 0$, sequence σ' is accepted. Alternatively, when $\Delta > 0$, σ' is accepted with probability $e^{-\Delta/T}$, where T is a parameter known as the temperature. To test whether σ' is accepted when $\Delta > 0$, a random number R is generated from the uniform distribution [0, 1]. If $R \leqslant e^{-\Delta/T}$, then σ' is accepted; otherwise σ is retained as the current sequence. Typically, the temperature is high in the initial stages of the search so that many increases in the objective function are accepted and then decreases until it is close to zero in the final stages at which point the procedure resembles a descent method.

Simulated annealing has its origins in statistical physics where the process of cooling solids slowly until they reach a low energy state is called annealing. Metropolis *et al.* [10] simulate energy levels in cooling solids by generating a sequence of states as follows. Suppose that Δ is the difference in energy levels between the current state and a new slightly perturbed one. If $\Delta \leqslant 0$, the process moves to the new state, whereas if $\Delta > 0$, it moves to the new state with probability $e^{-\Delta/T}$, where T is the temperature of the solid. Kirkpatrick *et al.* [7] are the first to point out the relevance of this process for combinatorial optimization problems. Their pioneering work has stimulated research into a wide variety of applications of simulated annealing. A review of various applications is given by van Laarhoven and Aarts [15]. Hajek [5] derives conditions under which the method generates an optimal solution with probability one. However, these conditions are of mainly theoretical interest since the usual aim is to generate an approximate solution using moderate computer resources.

As in a descent method, it is necessary to specify which neighbourhood is used, how the neighbourhood is searched and the method by which a starting solution is selected. A simulated annealing heuristic should additionally specify a sequence of temperatures to be used in the acceptance probability and indicate how many iterations are to be performed at each temperature.

Our implementation of simulated annealing for the permutation flow-shop problem is described now. We adopt the system of performing a single iteration at each temperature. This has the advantage of reducing the number of parameters to be set. Intuitively, there is likely to be little difference between performing several iterations at the same temperature and performing these iterations at temperatures which do not vary significantly. Our temperatures $T_1, \ldots, T_K$, where K is the total number of iterations (K sequences are generated and evaluated), follow the pattern

$$T_{k+1} = T_k/(1 + \beta T_k)$$

which is proposed by Lundy and Mees [9]. Using the results of preliminary experiments, we propose

$$T_1 = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij}/(5mn)$$

as an initial temperature and

$$T_K = 1$$

as a final temperature. Also, through observation of the number of iterations during which a significant improvement in the objective value occurs for various values of m and n (where $m \leqslant 20$ and $n \leqslant 100$) and fitting a regression equation, we propose that the number of iterations is given by

$$K = \max\{3300 \ln n + 7500 \ln m - 18250, 2000\} \quad (2)$$

Having fixed K, T_1 and T_K, the value of β is given by

$$\beta = (T_1 - T_K)/((K-1)T_1 T_K)$$

When the number of iterations is determined by equation (2), we have for large m and n that $K < 7500(\ln m + \ln n) < 15{,}000 \ln(m+n)$. Since $O(mn)$ computations per iteration are necessary to evaluate the new sequence, the time requirement for simulated annealing is $O(mn \log(m+n))$.

We evaluate four simulated annealing heuristics, each of which has $(1, \ldots, n)$ as an arbitrary starting sequence. The heuristics are denoted by SA(N, S), where N denotes the neighbourhood that is used and S describes how the neighbourhood is searched. We restrict our attention to $N \in \{I, S\}$, where I is the interchange neighbourhood and S is the shift neighbourhood. Also, $S \in \{O, R\}$, where O is an ordered search and R is a random search. In each heuristic, the sequence which yields the smallest maximum completion time is selected (although it may not be the current sequence when the procedure terminates).

4. COMPUTATIONAL EXPERIENCE

4.1 Test problems

The heuristics were compared using test problems with $m \in \{4, 7, 10, 20\}$ and $n \in \{20, 50, 100\}$. For each problem, an integer processing time p_{ij} from the uniform distribution [1,100] was generated for each job i and each machine j. Ten test problems were generated for each of the 12 pairs of values of m and n.

For each test problem, an initial estimate of the optimal objective value was made by performing a large number (100,000) of simulated annealing iterations. In cases where improved objective values were found during the computational tests described in the following subsections, the estimate was updated accordingly. In all the computational tests that are described, heuristics are coded in FORTRAN 77 and run on a CDC 7600 computer. Also, all maximum completion time evaluations in the improvement methods were computed directly: computation times can be reduced if completion times of each job on each machine for the current sequence are stored and used in the evaluation of the new sequence.

4.2 Simulated annealing heuristics

This subsection compares the four simulated annealing heuristics SA(I, O), SA(I, R), SA(S, O) and SA(S, R). Results are evaluated using the average relative percentage deviation (ARPD) of the heuristic solution from the best known solution, i.e. using·

$$\text{ARPD} = 100(C_{\max}(\sigma^H) - C_{\max}(\sigma^B))\, C_{\max}(\sigma^B),$$

where σ^H is the heuristic sequence and σ^B is the sequence which yields the best known solution. Average relative percentage deviations are listed in Table 1.

Since computation times for each of the simulated annealing heuristics are similar, the most suitable neighbourhood and method of search can be assessed using the figures given in Table 1. We first observe that the two heuristics which use a random search yield better results on

Table 1 Average relative percentage deviations for simulated annealing heuristics

m	*n*	SA(*I*, *O*)[a]	SA(*I*, *R*)[b]	SA(*S*,*O*)[c]	SA(*S*, *R*)[d]
4	20	0.10	0.18	0.49	0.10
7	20	0.54	0.70	1.59	0.56
10	20	1.62	1.43	1.89	0.85
20	20	1.36	1.35	0.42	0.25
4	50	0.60	0.25	1.45	0.10
7	50	0.49	0.19	1.09	0.17
10	50	1.37	1.11	2.78	0.43
20	50	1.54	1.69	2.60	0.85
4	100	0.60	0.14	1.88	0.08
7	100	1.11	0.49	2.00	0.33
10	100	2.29	0.88	3.50	0.54
20	100	3.31	2.00	5.41	1.59

[a] SA(*I*, *O*)—simulated annealing with interchange neighbourhood and ordered search

[b] SA(*I*, *R*)—simulated annealing with interchange neighbourhood and random search

[c] SA(*S*, *O*)—simulated annealing with shift neighbourhood and ordered search

[d] SA(*S*, *R*)—simulated annealing with shift neighbourhood and random search

average than their counterparts in which an ordered search is adopted. This is, perhaps, slightly surprising, because a random search may require many iterations to detect a neighbour having a better solution, whereas the ordered search is guaranteed to find it within a complete cycle of iterations. Of the methods SA(*I*, *R*) and SA(*S*, *R*), the latter which uses the shift neighbourhood appears to be far superior. We are unable to offer a satisfactory explanation of this result. Possibly, the best choice of neighbourhood in a sequencing problem is strongly dependent on the particular objective function.

Having found SA(*S*, *R*) to be the best of our simulated annealing heuristics, we next compare it with constructive heuristics and descent methods.

4.3 Comparison of heuristics

We now compare SA(*S*, *R*) with the constructive heuristic NEH of Nawaz, Enscore and Ham. Average relative percentage deviations and average computation times in seconds are listed in the first columns of Table 2. We observe that, on average, simulated annealing generates better quality solutions than heuristic NEH for all values of *m* and *n* considered. However, NEH does have the advantage of requiring much smaller computation times. There is some evidence that the quality of solutions generated by NEH is satisfactory for small *m*, but average relative percentage deviations increase as *m* increases. The results for $m = 10$ and $n = 20$, where the value of each solution generated by NEH deviates by more than three per cent from the optimum, best demonstrate that NEH should only be used when computation time is a limiting resource.

Our final tests compare simulated annealing with two descent methods. The first descent method uses NEH as a starting sequence, while the second uses PCDSD which is the best of the $m + 1$ sequences generated from the constructive methods of Palmer, Campbell, Dudek and Smith, and Dannenbring. Both descent methods use two phases. After computing NEH or PCDSD as a starting sequence, the first phase uses descent based on an ordered search of the interchange neighbourhood to generate a local minimum. The second phase uses the sequence generated by the first phase as a starting sequence and then applies descent based on an ordered search of the shift neighbourhood

Table 2 Average relative percentage deviations and computation times

		SA(*S*, *R*)[a]		NEH[b]		NEH + DES[c]		PCDSD + DES[d]	
m	*n*	ARPD[e]	ACT[f]	ARPD	ACT	ARPD	ACT	ARPD	ACT
4	20	0.10	0.26	0.61	0.02	0.61	0.02	0.33	0.09
7	20	0.56	1.19	3.78	0.03	2.75	0.08	2.56	0.21
10	20	0.85	2.23	4.59	0.03	2.78	0.12	3.01	0.31
20	20	0.25	6.40	3.41	0.06	1.89	0.21	1.78	0.73
4	50	0.10	1.30	0.12	0.21	0.06	0.49	0.15	1.00
7	50	0.17	3.75	0.92	0.33	0.37	0.91	0.75	2.14
10	50	0.43	6.61	1.96	0.46	0.94	2.17	1.66	5.36
20	50	0.85	17.77	4.04	0.88	2.03	4.61	2.06	13.63
4	100	0.08	3.51	0.13	1.53	0.00	4.60	0.00	8.91
7	100	0.33	8.91	0.62	2.50	0.20	8.61	0.25	20.60
10	100	0.54	15.07	1.59	3.48	0.77	14.50	0.65	47.19
20	100	1.59	39.08	3.10	6.72	1.41	43.13	1.44	154.04

[a] SA(*S*, *R*)—simulated annealing with shift neighbourhood and random search

[b] NEH—heuristic of Nawaz *et al*

[c] NEH + DES—heuristic of Nawaz *et al.* followed by two descent phases

[d] PCDSD + DES—best of Palmer, Campbell *et al* and Dannenbring sequences followed by two descent phases

[e] ARPD—average relative percentage deviation from the best known solution

[f] ACT—average computation time in seconds

to obtain another local minimum. Results for the descent methods NEH + DES and PCDSD + DES are shown in the final columns of Table 2.

Commenting first on the relative merits of the two descent heuristics, we note that PCDSD + DES requires much more computation time because it uses an inferior starting sequence. In terms of solution quality, however, both methods are comparable. Thus, NEH + DES is preferred. It remains to discuss the relatives merits of the methods SA(S, R) and NEH + DES. The general trend in our results is that the method which generates the better quality solution also requires more computation time. Over all test problems, average relative percentage deviations are 0.49 and 1 15, while average computation times are 8.84 seconds and 6.62 seconds for SA(S, R) and NEH + DES respectively. On this evidence, the superior solutions generated by SA(S, R) compensate for the larger computation times relative to the descent heuristic NEH + DES. Thus, our simulated annealing heuristic SA(S, R) is preferred to the other methods.

One further test of simulated annealing on the real problem with $m = 6$ and $n = 44$, given by Bestwick and Hastings [1], was performed. The processing times of the missing operations in this problem were taken to be zero. A sequence with a maximum completion time of 1640.6 was obtained in 2.43 seconds of computer time using heuristic SA(S, R). The optimal solution claimed in [1] has value 1640.7. The most likely explanation of this anomaly is that the data are incorrectly listed.

5. CONCLUDING REMARKS

With recent rapid developments in computer technology, fairly cheap micro-computers can now perform tasks that a few years ago required a mainframe. This increase in computer power means that one is no longer restricted to fairly simple constructive heuristics. For 'hard' problems, such as the permutation flow-shop problem in which there few available results about the structure of an optimal solution, improvement methods are strongly recommended. Our computational results show that the known constructive methods for permutation flow-show scheduling sometimes generate solutions which deviate significantly from the optimum

Descent methods and simulated annealing both deserve serious consideration when selecting a heuristic, although our results indicate that descent is slightly erratic unless a good starting solution is used. Simulated annealing has the advantage that its computational requirements are predictable; the number of iterations can be fixed according to the available computer resources and the urgency with which the solution is required. A further advantage of simulated annealing is that it is extremely easy to code.

During the writing of this paper, we became aware of the work of Widmer and Hertz[1]. They use an improvement heuristic called taboo search. In this method, all eligible neighbours of the current sequence are searched and the best of these is selected to be the new current sequence, even if it causes an increase in the objective value. All neighbours are eligible unless they correspond to entries in a dynamically changing list of forbidden job positions The list of forbidden job positions stops the method cycling over a short period. Computational results based on the interchange neighbourhood are given by Widmer and Hertz for problems with up to 20 jobs. No meaningful comparison of our results with those of taboo search can be given because more iterations are used in our improvement methods. It can be observed for our 20-job test problems, however, that SA(S, R) produces better solutions than heuristic NEH for 82.5% of problems, with identically valued solutions generated for the other 17.5%. Corresponding results of Widmer and Hertz show that taboo search yields better solutions than NEH for 58% of problems, identically valued solutions for 14% of problems and worse solutions for 28% of problems.

[1] A heuristic method for the flow-shop sequencing problem, presented at a workshop on Production Planning and Scheduling, Paris, 1988

REFERENCES

1 Bestwick PF and Hastings NAJ (1976) A new bound for machine scheduling. *Opl Res. Q.* **27**(2), 479–487
2 Campbell HG, Dudek RA and Smith ML (1970) A heuristic algorithm for the *n* job, *m* machine sequencing problem *Mgmt Sci* **16**(10), B630–637
3 Dannenbring DG (1977) An evaluation of flowshop sequencing heuristics *Mgmt Sci* **23**(11), 1174–1182
4 Garey MR, Johnson DS and Sethi R (1976) The complexity of flowshop and jobshop scheduling *Math Ops Res* **1**(2), 117–129
5 Hajek B (1988) Cooling schedules for optimal annealing *Math. Ops Res* **13**(2), 311–329

6 Johnson SM (1954) Optimal two- and three-stage production schedules with setup times included *Naval Res Logist Q* **1**(1), 61–68
7 Kirkpatrick S, Gelatt CD Jr and Vecchi MP (1983) Optimization by simulated annealing *Science* **220**, 671–680
8 Lenstra JK, Rinnooy Kan AHG and Brucker P (1977) Complexity of machine scheduling problems *Ann Discrete Math* **1**, 343–362
9 Lundy M and Mees A (1986) Covergence of an annealing algorithm *Math Prog.* **34**(1), 111–124
10 Metropolis N, Rosenbluth A, Rosenbluth M, Teller A and Teller E (1953) Equation of state calculations by fast computing machines *J. Chem Phys.* **21**, 1087–1092
11 Nawaz M, Enscore EE Jr and Ham I (1983) A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem *Omega* **11**(1), 91–95.
12. Palmer DS (1965) Sequencing jobs through a multistage process in the minimum total time—a quick method of obtaining a near optimum. *Opl Res Q* **16**(1), 101–107
13 Potts CN (1980) An adaptive branching rule for the permutation flow-shop problem *Eur J Opl Res* **5**(1), 19–25
14 Turner S and Booth D (1987) Comparison of heuristics for flow shop sequencing *Omega* **15**(1), 75–78
15 Van Laarhoven PJM and Aarts EHL (1987) *Simulated Annealing Theory and Applications* Reidel, Dordrecht, The Netherlands

ADDRESS FOR CORRESPONDENCE: *Dr CN Potts, Faculty of Mathematical Studies, University of Southampton, Southampton SO9 5NH, UK*