An Efficient Method for the Open-Shop Scheduling Problem Using Simulated Annealing

Haidar M. Harmanani and Steve Bou Ghosn

Abstract This paper presents a simulated annealing algorithm in order to solve the nonpreemptive *open-shop scheduling problem* with the objective of minimizing the *makespan*. The method is based on a simulated annealing algorithm that efficiently explores the solution space. The method was implemented and tested on various benchmark problems in the literature. Experimental results show that the algorithm performs well on the benchmarks. The algorithm was able to find an optimum solution in many cases.

1 Introduction

Scheduling is the allocation of shared resources over time to competing activities [16]. The $n \times m$ minimum makespan job shop scheduling problem consists of n jobs that are to be processed on m machines. A job is composed of a sequence of operations that are to be processed on a particular machine in a fixed order specified by this sequence. Each machine can perform a specific operation during a fixed processing time without interruptions. A schedule for a given machine specifies the exact time interval, start time and finish time, during which each job in the shop is to be processed on that particular machine. A schedule for the shop is a set of machine schedules, one for each machine and each schedule satisfies resources constraints.

The open shop scheduling problem (OSSP) is a variation of the general job scheduling problem. The open-shop scheduling problem consists of a set of n jobs, $\mathcal{J} = \{J_1, J_2, ..., J_n\}$, that are to be processed on a set of m machines, $\mathcal{M} = \{M_1, M_2, ..., M_m\}$. Each job $J_j \in \mathcal{J}$ consists of m operations $o_{j,i}$ that have

H.M. Harmanani(✉)
Department of Computer Science and Mathematics, Lebanese American University, Byblos 1401 2010, Lebanon
e-mail: haidar@lau.edu.lb

S.B. Ghosn
Department of Computer Science, Westfield State University, Westfield, MA 01085, USA

S. Latifi (ed.), *Information Technology New Generations*,
Advances in Intelligent Systems and Computing 448,
DOI: 10.1007/978-3-319-32467-8_102

Table 1 A 4x4 *Taillard Benchmark* instance for the OSSP

Jobs	*(Processing Time, Machine)*			
Job 1	(54, M_3)	(34, M_1)	(61, M_4)	(2, M_2)
Job 2	(9, M_4)	(15, M_1)	(89, M_2)	(70, M_3)
Job 3	(38, M_1)	(19, M_2)	(28, M_3)	(87, M_4)
Job 4	(95, M_1)	(34, M_3)	(7, M_2)	(29, M_4)

processing time $p_{j,i}$ and must be processed on M_i. At any given time, each machine can process a single operation while two operations that belong to the same job cannot be processed on different machines simultaneously. The problem is to find a schedule for the operations on the machines that minimizes the makespan, C_{max}, that is, the time from the beginning of the first operation until the end of the last operation [8]. An optimal finish time schedule is one that has the least finish time among all schedules. A schedule is said to be *non-preemptive* if for each individual machine there is at most one tuple $<i, s(i), f(i)>$ for each job i to be scheduled. A preemptive schedule is a schedule in which no restrictions are placed on the number of tuples per job per machine. The *non-preemptive* open-shop scheduling problem has been shown to be $\mathcal{NP}$-hard through a transformation to the *partition* problem [4]. It has been shown that one can establish the following lower bound for the optimal finish schedule [13]:

$$\mathcal{L} = \max \left\{ \max_i \sum_{j=1}^{m} p_{ij}, \max_j \sum_{i=1}^{n} p_{ij} \right\} \tag{1}$$

Where $1 \leq i \leq n$ *and* $1 \leq j \leq m$. The first part in equation 1 deals with the maximum completion time among all jobs that are to be processed, and simply states that the processing time for every job must necessarily take at least the sum of the processing times required to perform each of its tasks. The second part refers to the maximum completion time for the jobs allocated to a given machine. Thus, every machine must necessarily take a processing time which is at least equal to the sum of the times required to perform each of the jobs allocated to it. The optimal makespan for the open-shop scheduling problem can never be less than $\mathcal{L}$, but it is not necessarily equal to $\mathcal{L}$.

We illustrate the OSSP using the 4×4 *Taillard Benchmark* shown in Table 1. The benchmark instance consists of 4 jobs and 4 machines. Figure 1 presents a possible schedule with an optimal makespan of 193.

2 Related Work

One of the earliest and most significant efforts related to the open-shop scheduling problem was reported by Gonzalez et al. [4] who showed that the problem is

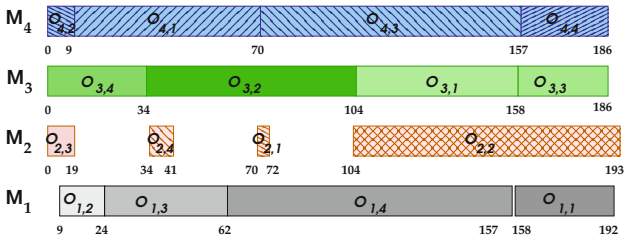

Fig. 1 Optimal schedule for problem in Table 1 with makespan = 193

$\mathcal{NP}$-complete by reducing it to the partition problem. The authors proposed a linear-time algorithm for the open-shop scheduling problem with $m = 2$, and a polynomial time algorithm to find the optimal schedule for the preemptive open-shop scheduling problem. Some attempts were made at solving the open-shop scheduling problem using branch and bound techniques [5]. Approximate methods were also reported. For example, Davis [3] proposed a genetic algorithm method that uses a memory intensive chromosome representation and very simple genetic operators. Fang et al. [6] proposed a genetic algorithm with an ordinal chromosomal representation, and adapted the approach to solve the open shop scheduling problem [15]. The algorithm used a representation that has the advantage of only producing valid schedules when altered by the genetic operators; thus allowing more accurate solutions in considerably less time. Prins [14] proposed a competitive genetic algorithm for solving the open-shop scheduling problem. Other researchers proposed hybrid genetic algorithms techniques that involve combining genetic algorithms with other optimization techniques. Khuri et al. [7] presented three different genetic algorithms approaches. One of these approaches is based on a genetic algorithm algorithm that uses a selfish gene. Another implementation was based on a hybrid GA implementation that proved to be better than the first two proposed approaches. Liaw [9] proposed an iterative improvement method for the open shop scheduling problem. The author, proposed later other efficient solution approaches based on a hybrid GA implementation [12], a tabu search technique [11], and a simulated annealing algorithm [10]. Blum [1] proposed a method based on a hybrid approach that combines ant colony optimization with beam search. Colak et al. [2] formulated the problem using an augmented neural network approach.

3 Simulated Annealing

Simulated Annealing is a global stochastic method that is used to generate approximate solutions to very large combinatorial problems. The annealing algorithm begins with an initial feasible configuration and proceeds to generate a neighboring solution by perturbing the current solution. If the cost of the neighboring solution is less than that of the current solution, the neighboring solution is accepted; otherwise,

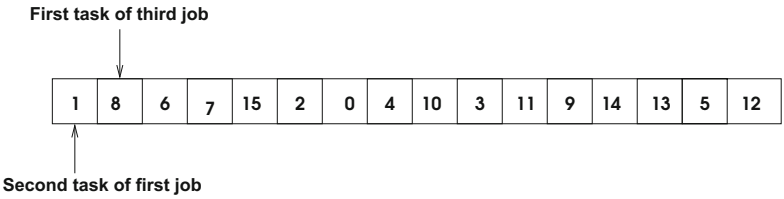

Fig. 2 Configuration representation and interpretation

it is accepted or rejected with probability $p = e^{-\frac{\Delta_C}{T}}$. The probability of accepting inferior solutions is a function of the *temperature*, T, and the change in cost between the neighboring solution and the current solution, Δ_C. The temperature is decreased during the optimization process and thus the probability of accepting a worse solution decreases as well. The set of parameters controlling the initial temperature, stopping criterion, temperature decrement between successive stages, and the number of iterations for each temperature is called the *cooling schedule*. Typically, at the beginning of the algorithm, the temperature T is large and an inferior solution has a high probability of being accepted. During this period, the algorithm acts as a random search to find a promising region in the solution space. As the optimization process progresses, the temperature decreases and there is a lower probability of accepting an inferior solution. The algorithm behaves like a down hill algorithm for finding the local optimum of the current region.

4 Simulated Annealing OSSP

The proposed algorithm starts with a number of machines, and a set of jobs and corresponding tasks. The algorithm generates, through a sequence of transformations, a set of compact schedules. The key elements in implementing the annealing test scheduling algorithm are: 1) the definition of the initial configuration, 2) the definition of a neighborhood on the configuration space and a perturbation operator exploring it; 3) the choice of the cost function; and 4) a cooling schedule. In what follows, we describe the simulated annealing algorithm with reference to the benchmark in Table 1.

4.1 Configuration Representation

In order to solve the OSSP, we propose the configuration shown in Figure 2. The representation is based on a vector where every cell corresponds to an operation in the schedule. The values are a permutation of integer values between 0 and $(m \times n) - 1$,

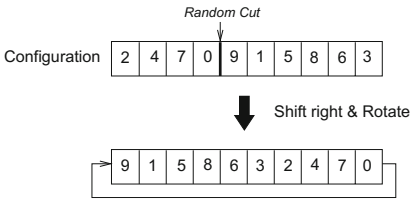

Fig. 3 Shift and Rotate Transformation

and where m is the number of machines and n the number of jobs. A value V is interpreted as the $(V \ mod \ m)$ operation of the $(V \ div \ m)$ job.

4.2 Initial Configuration

The initial configuration is created randomly by selecting the values to be unique permutations between 0 and $(n \times m) - 1$, where n is the number of jobs and m is the number of machines. The algorithm ensures that all $n \times m$ elements in the initial configuration have different values.

4.3 Objective Function

The objective is to find a schedule for the operations on the machines that minimizes the makespan, C_{max}. That is, the time from the beginning of the first operation until the end of the last operation. The objective function is given by the following:

$$F = \frac{1}{C_{max}} \tag{2}$$

4.4 Neighborhood Transformation

The algorithm uses two neighborhood transformations in order to explore the solution space. However, it should be noted that when scheduling operations, the algorithm applies an as soon as possible scheduling strategy in order to minimize idle gaps in the middle of the schedule. Thus, the algorithm iterates over the neighborhood solution and finds the earliest idle time slot that is large enough to fit the operation to

Algorithm 1. Annealing Algorithm

Input: (S_0, T_0, α, β, M, MaxTime)

```
1: T = T0                                   ▷ T0 is the initial temperature
2: CurSol = S0
3: CurCost = Cost(CurSol)
4: BestCost = CurCost
5: Time = 0
6: do
7:     Metropolis(CurSol, CurCost, BestSol, BestCost, T, M)
8:     Time = Time + M
9:     T = α × T
10:    M = β × M
11: while (Time ≥ maxTime || T > 0.001)
```

Algorithm 2. Metropolis

Input: (CurSol, CurCost, BestSol, BestCost, T, M)

```
1: do
2:     NewSol = Neighborhood(CurSol)
3:     NewCost = Cost(NewSol)
4:     Δ_Cost = NewCost - CurCost
5:     if Δ_Cost < 0 then
6:         CurSol = NewSol
7:         if NewCost < BestCost then BestSol = NewSol
8:         end if
9:     else
10:        if Random < e^(-Δ_Cost / T) then CurSol = NewSol
11:        end if
12:    end if
13:    M = M - 1
14: while M ≠ 0
```

be scheduled. If no available time slot is large enough then the operation is scheduled after the last operation that is scheduled on that machine.

4.4.1 Swap

The swap operator selects two *random* positions between 0 and $(m \times n) - 1$, and swaps their positions. Since a solution is valid to the extent that it represents a feasible schedule, swapping two operations will alter the order of the execution of operations in the schedule resulting in a new feasible schedule that is 'similar' to the original schedule. The number of swaps we perform in each call to the neighborhood function is controlled by a tuning parameter that is kept very small in order not to drastically alter the solution in a single annealing iteration.

4.4.2 Shift and Rotate Transformation

The shift and rotate transformation is used in order to perturb the solution while maintaining some of the attributes of the original solution while guaranteeing that the resulting solution is feasible. The transformation selects a random position between 0 and $(m \times n) - 1$. The neighborhood solution is next shifted and rotated either right or left as determined by a random variable. Figure 3 illustrates a shift right and rotate by seven.

4.5 Cooling Schedule

The cooling schedule is the set of parameters that control the initial temperature, the stopping criterion, the temperature decrement between successive stages, and the number of iterations for each temperature. The idea is to initially set the initial temperature, T_0, to a high number, which causes the algorithm to perform a random walk in the solution space by moving to every newly-created solution regardless of how good it is. This implies that the probability of accepting a solution should be 1 or close to 1:

$$P(T_0) = \frac{\text{Number of moves accepted}}{\text{Total Number of Moves Attempted}} \tag{3}$$

We determine the initial temperature T_0 by initially setting it to a small value and computing $P(T_0)$, if the value isn't close to 1 then we keep gradually incrementing the temperature by multiplying it by a constant K $(K > 1)$ and repeating this procedure until we reach a proportion which is 1 or very close to 1. The temperature we use to obtain that proportion becomes T_0. This procedure models the process of heating the material until all its atoms are completely free. After performing this process we empirically determined the optimal initial temperature (T_0) to be 400.

Another very important parameter we need to tune for our cooling schedule is the rate at which we decrease the temperature. The cooling rate is determined by α which is the multiplier that is used to decrease the temperature. Ideally, we would like to decrease the temperature as slowly as possible until to eventually reach 0, at which point the algorithm is doing nothing more than plain Hill-Climbing. It follows that we should select a value of α that is very close to 1. In our empirical testing we have determined that the optimal value range is $0.99 \leq \alpha \leq 0.999$, and for most samples setting α to 0.999 seems to give us the best results. Since the approach we have followed for our annealing implementation is a metropolis approach, the algorithm will stop either when the temperature reaches 0.001 or until the *MaxTime* is reached.

The *MaxTime* parameter is problem dependent as bigger problems require more running time in order to achieve good results. We have also empirically determined the metropolis parameters, M and β to be 5 and 1.05, respectively.

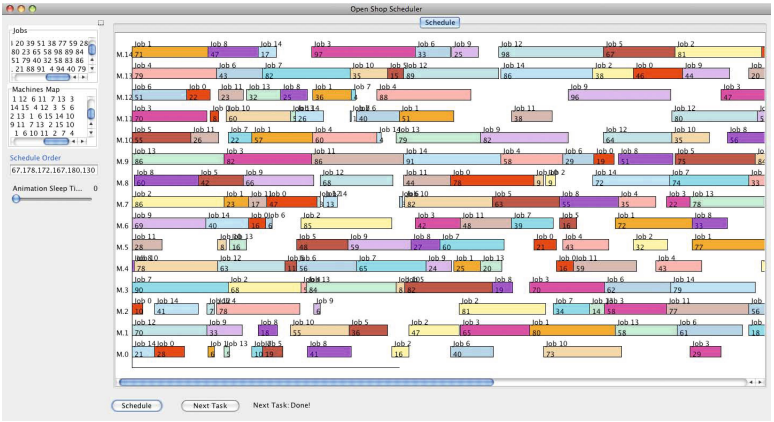

Fig. 4 Schedule for an instance of problem 20 × 20 with a makespan of 1292

5 Results

The algorithm was implemented using C++ and tested on the *Taillard Benchmark* problem instances. Figures 5(a) and 5(b) show plots representing the results we obtained for a small 4 × 4 problem instance and a large 15 × 15 problem instance. Tables 2, and 3 show the results obtained after running each of the *Taillard Benchmark* instances using the determined parameters. It can be observed that the algorithm was able to find the optimum solutions in many benchmark instances. The execution time was less than five minutes for all benchmark instances. The neighborhood

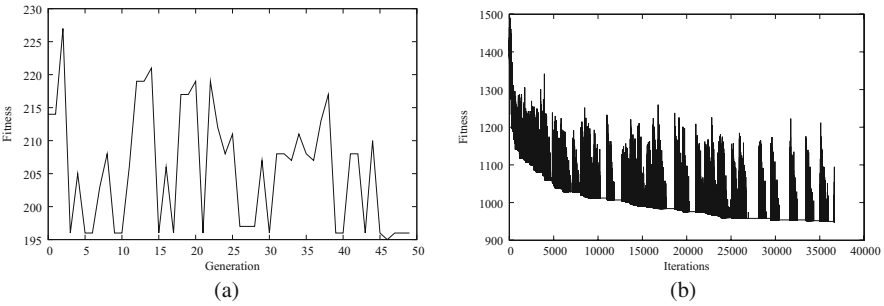

Fig. 5 (a) Plot for Taillard 4 x 4 - 0; b) Plot for Taillard 15 x 15 - 0

Table 2 Results Comparison for Taillard's instances 4x4, 5 x5, 7x7, and 10x10

Problem Instance	Optimal	Ours	Khoury [7]	Hybrid GA [7]	Fang [15]	Prins [14]
4 x 4 - 1	193	193	193	213	193	193
4 x 4 - 2	236	236	236	240	236	239
4 x 4 - 3	271	271	271	293	271	271
4 x 4 - 4	250	250	250	253	250	250
4 x 4 - 5	295	295	295	303	295	295
4 x 4 - 6	189	189	189	209	189	189
4 x 4 - 7	201	201	201	203	201	201
4 x 4 - 8	217	217	217	224	217	217
4 x 4 - 9	261	261	261	281	261	261
4 x 4 - 10	217	217	217	230	217	221
5 x 5 - 1	300	300	301	323	300	301
5 x 5 - 2	262	262	262	269	262	263
5 x 5 - 3	323	323	331	353	323	335
5 x 5 - 4	310	310	N/A	N/A	310	316
5 x 5 - 5	326	326	N/A	N/A	326	330
5 x 5 - 6	312	312	312	327	312	312
5 x 5 - 7	303	303	N/A	N/A	303	308
5 x 5 - 8	300	300	N/A	N/A	300	304
5 x 5 - 9	353	353	353	373	353	358
5 x 5 - 10	326	326	326	341	326	328
7 x 7 - 1	435	435	438	447	435	436
7 x 7 - 2	443	443	455	454	443	447
7 x 7 - 3	468	468	N/A	N/A	468	472
7 x 7 - 4	463	463	N/A	N/A	463	463
7 x 7 - 5	416	416	N/A	N/A	416	417
7 x 7 - 6	451	451	N/A	N/A	451	455
7 x 7 - 7	422	422	443	450	422	426
7 x 7 - 8	424	424	N/A	N/A	424	424
7 x 7 - 9	458	458	465	467	458	458
7 x 7 - 10	398	398	405	406	398	398
10 x 10 - 1	637	637	667	655	637	637
10 x 10 - 2	588	588	N/A	N/A	588	588
10 x 10 - 3	598	598	N/A	N/A	598	598
10 x 10 - 4	577	577	586	581	577	577
10 x 10 - 5	640	640	N/A	N/A	640	640
10 x 10 - 6	538	538	555	541	538	538
10 x 10 - 7	616	616	N/A	N/A	616	616
10 x 10 - 8	595	595	N/A	N/A	595	595
10 x 10 - 9	595	595	627	598	595	595
10 x 10 - 10	596	596	623	605	596	596

transformations were applied in every iteration with a probability 0.75 for the swap operator and 0.25 for the shift and rotate transformation. We show the solution for an instance of the 20×20 benchmark in Figure 4 using our GUI interface.

Table 3 Results Comparison for Taillard's instances 15x15 and 20x20

Problem Instance	Optimal	Ours	Khoury [7]	Hybrid GA [7]	Fang [15]	Colak [2]	Prins [14]
15 x 15 - 1	937	937	967	937	937	937	937
15 x 15 - 2	918	918	N/A	N/A	918	918	918
15 x 15 - 3	871	871	904	871	871	871	871
15 x 15 - 4	934	934	969	934	934	934	934
15 x 15 - 5	946	946	N/A	N/A	946	946	946
15 x 15 - 6	933	933	N/A	N/A	933	933	933
15 x 15 - 7	891	891	N/A	N/A	891	891	891
15 x 15 - 8	893	893	928	893	893	893	893
15 x 15 - 9	899	899	N/A	N/A	899	901	899
15 x 15 - 10	902	902	N/A	N/A	902	902	902
20 x 20 - 1	1155	1155	1230	1165	1155	1115	1115
20 x 20 - 2	1241	1241	N/A	N/A	1241	1242	1241
20 x 20 - 3	1257	1282	1292	1257	1257	1173	1257
20 x 20 - 4	1248	1274	N/A	N/A	1248	1248	1248
20 x 20 - 5	1256	1289	1315	1256	1256	1256	1256
20 x 20 - 6	1204	1204	1266	1207	1204	1204	1204
20 x 20 - 7	1294	1294	N/A	N/A	1294	1294	1294
20 x 20 - 8	1169	1169	N/A	N/A	1169	1173	1169
20 x 20 - 9	1289	1307	1339	1289	1289	1289	1289

6 Conclusion

In this paper, we presented a simulated annealing algorithm for minimizing the makespan in the nonpreemptive open shop scheduling problem. The method was implemented and tested on various benchmark problems in the literature. The algorithm was able to find an optimum solution in many cases. The computing times did not exceed five minutes for the largest benchmark example.

References

1. Blum, C.: Beam-aco-hybridizing ant colony optimization with beam search: an application to open shop scheduling. Computers and Operations Research **32**(6), 1565–1591 (2005)
2. Colak, S., Agarwal, A.: Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. Naval Research Logistics **52**(7), 631–644 (2005)
3. Davis, L.: Job shop scheduling with genetic algorithms. In: Proceedings of the International Conference on Genetic Algorithms and their Applications, pp. 136–140 (1985)
4. Gonzalez, T., Sahni, S.: Open shop scheduling to minimize finish time. Journal of the Association for Computing Machinery **23**(4), 665–679 (1976)
5. Gueret, C., Prins, C.: Classical and new heuristics for the open-shop problem: a computational evaluation. European Journal of Operational Research **107**, 306–314 (1998)
6. Fang, H.-L., Ross, P., Corne, D.: A promising genetic algorithm approach to job shop scheduling, rescheduling and open-shop scheduling problems. In: Proceedings of the Fifth International Conference in Genetic Algorithms, pp. 375–382 (1993)

7. Khuri, S., Miryala, S.R.: Genetic algorithms for solving open shop scheduling problems. In: Proceedings of the 9th Portuguese Conference on Artificial Intelligence. Lecture Notes in Computer Science, pp. 357–368 (1999)
8. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: Sequencing and scheduling: Algorithms and complexity. Handbooks in Operations Research and Management Science: Logistics of Production and Recovery **4**, 445–522 (1993)
9. Liaw, C.-F.: An iterative improvement approach for the nonpreemptive open shop scheduling problem. European Journal of Operational Research **111**, 509–517 (1998)
10. Liaw, C.-F.: Applying simulated annealing to the open shop scheduling problem. IIE Transactions, Scheduling and Logistics **31**(5), 457–465 (1999)
11. Liaw, C.-F.: A tabu search algorithm for the open shop scheduling problem. Computers and Operations Research **26**(2), 109–126 (1999)
12. Liaw, C.-F.: A hybrid genetic algorithm for the open shop scheduling problem. European Journal of Operational Research **124**, 28–42 (2000)
13. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice Hall (1995)
14. Prins, C.: Competitive Genetic Algorithms for the Open-Shop Scheduling Problem. Mathematical Methods of Operations Research **52**(3), 389–411 (2000)
15. Fang, H.-L., Ross, P., Corne, D.: A promising hybrid GA/heuristic approach for open-shop scheduling problems. In: Proceedings of the 11th European Conference on Artificial Intelligence, pp. 590–594 (1994)
16. Yamada, T., Nakano, R.: Job-Shop Scheduling. Genetic algorithms in Engineering Systems, chap. 7, pp. 134–160. IEE (1997)