An Efficient Heuristic Algorithm for m-Machine No-Wait Flow Shops

Dipak Laha and Sagar U. Sapkal

Abstract—**We propose a constructive heuristic for the well known NP-hard problem of no-wait flow shop scheduling. It is based on the assumption that the priority of a job in the sequence is given by the sum of its processing times on the bottleneck machine(s) for selecting the initial sequence of jobs. The computational experimentations show that there is a significant improvement in solution quality over the existing heuristic, especially for large problem sizes while not affecting its time complexity. Statistical tests are used to substantiate the significance of the results by the proposed method.**

Index Terms— **No-wait flow shop scheduling, heuristics, total flow time, combinatorial optimization**

I. INTRODUCTION

A no-wait flow shop is a manufacturing system where each job is processed until completion without interruption either on or between any two consecutive machines; that is, once a job is started on the first machine, it has to be continuously processed through machines without interruption. In addition, we assume that each machine can handle no more than one job at a time and each job has to visit each machine exactly once without preemption. Therefore, when needed, the start of a job on the first machine must be delayed in order to meet the no-wait requirement. A detailed survey of the methods and applications of these scheduling problems is given by Hall and Sriskandarajah [1].

The no-wait flow shop scheduling problems with more than two machines belong to the class of NP-hard [2], [3]. Various researchers have developed the constructive heuristics as well as metaheuristics for solving these problems for different criterion such as makespan and total flow time. For no-wait flow shop scheduling problems, noteworthy constructive heuristics with the total flow time objective have been studied by Rajendran and Chaudhuri [4], Bertolissi [5], Aldowaisan and Allahverdi [6], and Framinan, Nagano, and Moccellin [7].

Rajendran and Chaudhuri [4] proposed two constructive heuristics considering two heuristic preference relations as the basis for selecting the seed sequence of jobs. The seed sequence of jobs thus generated is then improved further by using the job insertion method of Nawaz-Enscore-Ham (NEH) [8] in the remaining parts of the heuristics. Their heuristics, especially heuristic 1 perform significantly better than those of Bonney and Gundry [9], and King and Spachis [10].

Bertolissi [5] presented a heuristic based on calculating the minimum apparent flow time of each pair of jobs and then finding the number of times (here, marks) of the starting jobs of the pairs as a basis for selecting the seed sequence of jobs. The seed sequence thus generated is then improved further using the job insertion algorithm in the same manner as done by Rajendran and Chaudhuri [4]. The computational results reveal that the heuristic of Bertolissi performs better than those given by the heuristic of Rajendran and Chaudhuri [4], and Bonney and Gundry [9].

The heuristics of Aldowaisan and Allahverdi [6] and Framinan, Nagano, and Moccellin [7] have shown superior performance compared to the existing heuristics. However, these heuristics take much larger computational times compared to those given by the existing heuristics. Therefore, these heuristics are not considered here for comparison of the heuristics for the total flow time minimization problem.

In this paper, we present a constructive heuristic for minimizing the total flow time which is based on the assumption that the priority of a job in the sequence is given by the sum of its processing times on the bottleneck machine(s). We show, through computational experimentation that the proposed heuristic performs significantly well compared to the Bertolissi heuristic [5] which has shown better than the heuristics of Rajendran and Chaudhuri [4], and Bonney and Gundry [9]. Also, the mean CPU time used by the proposed method is found to be less compared to the Bertolissi heuristic.

II. THE PROPOSED HEURISTIC ALGORITHM

Given the processing time $p(i, j)$ of job i on machine j in the no-wait flow shop scheduling, each of n jobs is processed on m machines in the same technological order without preemption and interruption on or between any two consecutive machines. The problem is to determine a sequence of n jobs that minimizes the total flow time criterion. Let $\sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ represent the sequence of n jobs to be processed on m machines, and $d(i, k)$ the minimum delay on the first machine between the start of job i and the start of job k (required because of the no-wait restriction). Also, let $p(\sigma_i, j)$ represent the processing time on machine j of the job in the ith position of a given sequence, and let $d(\sigma_{i-1}, \sigma_i)$ denote the minimum delay on the first machine between the start of two consecutive jobs found in

Manuscript received January 03, 2011.

Dipak Laha, Reader, Mechanical Engineering Department, Jadavpur University, Kolkata 700032, India. (e-mail: dipaklaha_jume@yahoo.com, dlaha@mech.jdvu.ac.in).

Sagar U. Sapkal, Research Scholar, Mechanical Engineering Department, Jadavpur University, Kolkata 700032, India. (corresponding author, phone: +9133-2414-6890; fax: +9133-2414-6890; e-mail: sagar_us@indiatimes.com).

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the (*i*-1)th and *i*th position of the sequence. The total flow time (*TFT*) of the sequence of *n* jobs in the no-wait flow shop scheduling is given by,

$$TFT = \sum_{i=2}^{n} (n+1-i)d(\sigma_{i-1}, \sigma_i) + \sum_{i=1}^{n}\sum_{j=1}^{m} p(i, j)$$

where, the delay matrix D, of the $d(i, k)$ values are calculated as in Fink and Voß [11]. It may be noted that since the release times of the jobs are all zeros, the total completion time criterion is equivalent to the *TFT* criterion.

The shortest processing time (*SPT*) rule optimizes the mean flow time of a set of jobs processing on single machine [12], [13] and also it has been shown to be effective in m-machine flow shop scheduling [14]. Based on this idea, we make a similar attempt to obtain a sequence of jobs considering bottleneck machine(s) [15], [16], which is used as a starting sequence of the proposed method. The proposed method is based on the assumption that the priority of a job in the sequence is given by the sum of its processing times on the bottleneck machines. The seed sequence of jobs thus obtained is then applied to the remaining parts of the heuristic following a job insertion algorithm as described in the work of Rajendran and Chaudhuri [4].

The proposed heuristic algorithm is given as follows:

- Step 1: Set z = 1. Consider single bottleneck machine among *m* machines with the largest sum of the processing times of *n* jobs and determine the sequence of jobs by arranging them in ascending order of their processing times on it.
- Step 2: For z = 2 to *m*, consider z adjacent machines as bottlenecks with largest sum of the processing times of jobs on these machines and obtain the sequence of jobs in ascending order of the sum of the processing times of the individual jobs on these machines.
- Step 3: A total of *m* sequences of jobs are generated from steps 1 – 2 and the best one is selected as the initial sequence of jobs for the rest part of the heuristic.
- Step 4: Set $k = 1$. Select the first job in the initial sequence and insert it in the first position of the partial sequence σ and call this partial sequence as the current partial sequence σ.
- Step 5: Update $k = k+1$. Select the *k*-th job from the initial sequence and insert it in the *r*-th possible position of the current partial sequence σ where *r* is the integer varying $k/2 \le r \le k$ to produce *r* sequences. Among these *r* sequences, select the one as the new partial sequence σ with the minimum value of the expression: $\sum_{i=2}^{k}(k+1-i)d(\sigma_{i-1}, \sigma_i)$ and make it the current partial sequence σ.
- Step 6: If $k = n$, go to Step 7, else go to step 5.
- Step 7: The sequence σ is the final sequence.

The algorithmic complexity of the proposed heuristic is as follows: Step 1 finds a sum of *n* terms for each of *m* machines with complexity O(*nm*), and then gives sorting of *n* items using the bubble sort technique [17] with an average – case complexity of $O(n^2)$. The complexity of a schedule of *n* jobs on *m* machines is O(*nm*). Thus, the total complexity of step 1 is $O(2nm + n^2)$ or $O(n^2)$. Step 2 actually dictates the complexity of the proposed method. In the first part of step 2, the total number of comparisons (varying the number of adjacent machines from 2 to *m*) is equal to $m(m-1)/2$ and its complexity is $O(m^2)$. In the next part of step 2, $(m-1)$ schedules of *n* jobs on *m* machines are generated, thereby resulting in total complexity of $O(m^2 + nm)$. Hence, the overall complexity of the proposed method is $O(2nm + 2n + m^2 + nm)$ or $O(n^2 + 3nm)$.

III. Bertolissi Heuristic [5]

The main idea of the Bertolissi heuristic [5] is to obtain an initial schedule as a starting point which is used for generating the job seed and defining the order in which jobs will be selected for the job insertion as described by Rajendran and Chaudhuri [4]. The principle for the selection of the seed job and the order of insertion of jobs in the available partial schedule is based on the work of Chan and Bedworth [18].

The heuristic consists of the following steps:

- Step 1: Compute the flow times for each pair of jobs i, k by using the equation $F(i,k) = 2p(i,1) + \sum_{j=2}^{m} p(i,j) + R_{m(ik)}$, where $R_{m(ik)}$ is recursively computed as $R_{m(ik)} = p(k,m) + \max(R_{m-1(ik)}, \sum_{r=2}^{m} p(i,r))$, and $R_{1(ik)} = p(k,1)$.
- Step 2: Compare each pair of flow times ($F(i,k)$ and $F(k,i)$) and select the smallest one, and mark the starting job of the pair.
- Step 3: perform step 2 for all the pairs of flow times.
- Step 4: Count the number of marks of each job and order the jobs in decreasing number of marks and use this ordering as the initial sequence of jobs.
- Step 5: Set $k = 1$. Select first job from this initial sequence and insert it in the first position of the partial sequence σ. Call it as current sequence σ.
- Step 6: Increment *k*, $k = k+1$. Select the *k*-th job from the sorted array of step 1 and insert it in the *r* possible positions of the current sequence σ, where $k/2 \le r \le k$. Select the best one among *r* sequences with the minimum value of $\sum_{i=2}^{k}(k+1-i)d(\sigma_{i-1}, \sigma_i)$ and set it as current sequence σ.
- Step 7: If $k = n$, go to step 8, else go to step 6.
- Step 8: The sequence σ is the final solution of the heuristic.

The algorithmic complexity of Bertolissi heuristic is as follows: Step 1 involves the generation of $n(n-1)/2$ schedules of 2 jobs. Also, the complexity of each flow time calculation for each schedule of a pair of jobs on *m* machines is O(2*m*). Thus, the complexity of step 1 is $O(n(n-1)2m)$ or $O(n^2m)$. In step 2, the number of comparison between two flow times is $n(n-1)/2$ and its complexity is $O(n(n-1)/2)$ or $O(n^2)$. Step 4 gives sorting *n* items and using bubble sort technique [17], an average – case complexity is $O(n^2)$. Hence, the overall complexity of steps 1 – 4 of the Bertolissi heuristic is $O(n^2m)$, which is more than that of the proposed method with the complexity of $O(n^2 + 3nm)$.

It has been shown, through the exhaustive computational experimentation, that the Bertolissi heuristic produces near – optimal solutions, which generally gives better results than

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the ones provided by Rajendran and Chaudhuri [4] heuristic and Bonney and Gundry [9] for both small and large problem sizes. However, Bertolissi heuristic requires more computational times compared to those given by the existing heuristics. Based on this evaluation, the method proposed by Bertolissi is considered as the best existing heuristic for the minimization of total flow time in no-wait flow shop scheduling.

IV. Computational Experience

The proposed heuristic, and the heuristic of Bertolissi [5] were coded in C and run on an Intel Core 2 Duo, 2 GB RAM, 2.93 GHz PC. To compare the proposed heuristic with the existing heuristic, we carried out the experimentation in two phases. In the first phase, we considered small problem sizes with number of jobs (n) = 5, 6, 7, 8, and 9 and number of machines (m) = 5, 10, 15, 20, and 25. The second phase was formed taking the large problem sizes with n = 10, 20, 30, 40, 50, 60, and 70 and m = 5, 10, 15, 20, and 25. Thirty independent problem instances were considered for each problem size. Each problem instance corresponds to a new processing time matrix where each processing time was generated from a uniform random discrete u(1, 99) distribution, commonly used by researchers [19], [20].

The following two performance measures, popular in the scheduling literature [4], [6], [7] are used in the present experimentation: average relative percentage deviation (*ARPD*), and percent of optimal solutions (for small problem sizes) or percent of best heuristic solutions (for large problem sizes).

ARPD for small number of jobs problems is given by,

$$ARPD = \frac{100}{k}\sum_{i=1}^{k}\left(\frac{Heuristic_i - Optimal_i}{Optimal_i}\right)$$

ARPD for large number of jobs problems is given by,

$$ARPD = \frac{100}{k}\sum_{i=1}^{k}\left(\frac{Heuristic_i - Best_i}{Best_i}\right)$$

where, $Heuristic_i$ denotes the objective function value obtained for i-th instance by a heuristic, $Optimal_i$ is the optimal solution value obtained for that instance, $Best_i$ is the best solution value obtained for that instance, and k is the number of problem instances for a problem size.

Table I displays comparative evaluation of the proposed method, and the heuristic of Bertolissi [5] based on *ARPD*, and the percent of optimal solutions for the small problem sizes (n = 5, 6, 7, 8, and 9). The results of Table I show that the overall performance of the proposed heuristic with respect to *ARPD*, and percent optimal solution is comparable to that of Bertolissi heuristic for small job size problems. The proposed method performs better than the Bertolissi heuristic for 15 cases with respect to the *ARPD,* and 15 cases with respect to the percent times optimal solution found out of each 25 cases for small problem sizes.

Table II presents comparative results based on *ARPD*, and the percent of best heuristic solutions for the large problem sizes (n = 10, 20, 30, 40, 50, 60, and 70). Table II indicates that the proposed method also performs significantly better than the Bertolissi heuristic for large job sizes. The proposed method performs better than the Bertolissi heuristic for 26 cases with respect to the *ARPD,* and 23 cases with respect to

TABLE I
ARPD, and Percent Times Optimal Solution Obtained for Small Problem Sizes

n	m	No. of problem instances	*ARPD*		Percent optimal	
			Bertolissi Heuristic	Proposed Heuristic	Bertolissi Heuristic	Proposed Heuristic
5	5	30	0.2928	0.1409	73	73
	10	30	0.4158	0.3560	60	73
	15	30	0.0723	0.0485	90	93
	20	30	0.2428	0.1466	77	83
	25	30	0.3326	0.1079	83	90
6	5	30	0.6458	0.7016	60	63
	10	30	0.7408	0.6968	47	43
	15	30	0.1921	0.2568	77	77
	20	30	0.3801	0.3108	53	60
	25	30	0.3490	0.2417	67	73
7	5	30	1.0754	1.3380	57	50
	10	30	1.0386	1.0061	33	40
	15	30	0.3247	0.5015	63	53
	20	30	0.4922	0.5889	47	50
	25	30	0.5762	0.6903	40	43
8	5	30	1.3554	1.5647	43	40
	10	30	1.1104	1.0262	20	27
	15	30	0.4167	0.5670	50	43
	20	30	0.7958	0.5904	30	40
	25	30	0.6239	0.6552	33	33
9	5	30	1.7903	1.5634	13	17
	10	30	1.5373	1.3318	17	20
	15	30	0.5359	1.1005	40	30
	20	30	0.8441	0.7762	17	30
	25	30	0.9923	0.8494	27	27
Average			0.6869	0.6863	49	51

TABLE II
ARPD, and Percent Times Best Solution Obtained for Large Problem Sizes

n	m	No. of problem instances	*ARPD*		Percent best	
			Bertolissi Heuristic	Proposed Heuristic	Bertolissi Heuristic	Proposed Heuristic
10	5	30	0.63	0.41	60	60
	10	30	0.39	0.33	67	63
	15	30	0.12	0.83	83	50
	20	30	0.45	0.38	63	63
	25	30	0.35	0.40	67	63
20	5	30	0.58	0.42	43	57
	10	30	0.66	1.17	60	40
	15	30	0.50	0.75	53	50
	20	30	0.81	0.53	37	63
	25	30	0.41	0.63	53	47
30	5	30	0.79	0.56	40	60
	10	30	0.61	0.65	57	43
	15	30	1.08	0.44	37	63
	20	30	1.01	0.81	40	60
	25	30	0.49	0.58	43	57
40	5	30	0.93	0.67	50	50
	10	30	0.74	0.43	47	53
	15	30	0.87	0.26	40	60
	20	30	0.56	0.73	50	50
	25	30	0.58	0.68	50	50
50	5	30	1.21	0.19	27	73
	10	30	0.64	0.55	33	67
	15	30	0.97	0.44	40	60
	20	30	1.07	0.21	27	73
	25	30	0.80	0.28	27	73
60	5	30	1.12	0.29	27	73
	10	30	0.78	0.65	33	67
	15	30	0.80	0.29	33	67
	20	30	0.93	0.34	30	70
	25	30	0.93	0.13	20	80
70	5	30	1.31	0.41	27	73
	10	30	0.83	0.33	37	63
	15	30	0.96	0.22	33	67
	20	30	0.85	0.29	40	60
	25	30	0.78	0.24	27	73
Average			0.76	0.47	43	61

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the percent times best solution found out of each 35 cases for large problem sizes.

Next, we show the statistical significance [21] of the results obtained by the proposed method over those produced by the Bertolissi heuristic. The number of problem instances for each problem size is taken as 30. We test the null hypothesis, $H_0 : \mu = 0$ against alternative hypothesis, $H_1 : \mu > 0$; i.e., if H_0 holds true, then, statistically, the difference between the two methods is not significant. At 5 % level of significance, the critical value, $t_{0.05,v}$ is obtained from the relation Probability $(t \geq t_{0.05,v}) = \alpha = 0.05$. Using the standard tables of *t*-distribution, we obtain, $t_{0.05,v} = 1.699$ for $v = N\text{-}1 = 29$ degrees of freedom. We also compare the level of significance with the *p*-value. The results are presented in Table III.

TABLE III
RESULTS OF STATISTICAL TEST

n	*m*	No. of problem instances	Bertolissi heuristic versus proposed heuristic			
			TFT difference		*t*	*p*-value
			Mean	Std. dev.		
40	5	30	144	1106	0.71	0.241
	10	30	254	1226	1.13	0.133
	15	30	612	1409	2.38	0.012
	20	30	-203	1797	-0.62	0.729
	25	30	-124	2173	-0.31	0.622
50	5	30	892	1360	3.59	0.001
	10	30	119	1974	0.33	0.372
	15	30	778	2328	1.83	0.039
	20	30	1502	2503	3.29	0.001
	25	30	1044	2460	2.32	0.014
60	5	30	999	1934	2.83	0.004
	10	30	220	3331	0.36	0.360
	15	30	1045	2523	2.27	0.015
	20	30	1420	3683	2.11	0.022
	25	30	2202	3105	3.88	0.000
70	5	30	1449	3256	2.44	0.011
	10	30	1131	3109	1.99	0.028
	15	30	2007	3487	3.15	0.002
	20	30	1823	4536	2.20	0.018
	25	30	1950	4533	2.36	0.013

The average computational times (in seconds) required for solving each problem instance by the heuristics are given in Table IV. The results show that the proposed method takes less computational time than that of the Bertolissi heuristic. Overall, the results of the proposed method are better by 38% in *ARPD* and 42% in percent times best found especially for large problem size instances while taking about 15% less CPU time than required by the Bertolissi heuristic.

V. CONCLUSION

In this paper, we have presented a constructive heuristic for the no-wait flow shop scheduling with the objective of minimizing total flow time criterion. The method is based on the principle of the sum of processing times of individual jobs on the bottleneck machines to determine the initial sequence of jobs. Based on the computational experimentation, the proposed method gives comparable performance as that of the Bertolissi heuristic for small problem sizes, whereas, there is significant improvement in solution quality for large problem sizes. Also, it has been shown that the CPU time required by the proposed method is less.

TABLE IV
MEAN CPU TIME (IN SECONDS) REQUIRED BY THE BERTOLISSI HEURISTIC AND THE PROPOSED HEURISTIC

n	*m*	Bertolissi Heuristic	Proposed Heuristic
10	5	0.000	0.000
	10	0.000	0.000
	15	0.000	0.000
	20	0.001	0.000
	25	0.002	0.001
20	5	0.001	0.000
	10	0.001	0.001
	15	0.001	0.001
	20	0.002	0.002
	25	0.004	0.003
30	5	0.001	0.000
	10	0.003	0.002
	15	0.003	0.002
	20	0.004	0.004
	25	0.007	0.007
40	5	0.002	0.001
	10	0.003	0.002
	15	0.005	0.004
	20	0.007	0.007
	25	0.012	0.013
50	5	0.003	0.001
	10	0.005	0.003
	15	0.007	0.007
	20	0.014	0.011
	25	0.020	0.020
60	5	0.003	0.002
	10	0.007	0.005
	15	0.010	0.009
	20	0.019	0.016
	25	0.029	0.030
70	5	0.005	0.003
	10	0.012	0.007
	15	0.019	0.015
	20	0.026	0.022
	25	0.040	0.040
Average		0.0079	0.0069

REFERENCES

[1] N. G. Hall and C. Sriskandarajah, "A survey of machine scheduling problems with blocking and no-wait in process," *Opeations Research*, vol. 44, pp. 510-525, 1996.

[2] C. H. Papadimitriou and P. C. Kanellakis, "Flowshop scheduling with limited temporary storage," *Journal of Associate Computer Machinery*, vol. 31, pp. 343-357, 1980.

[3] H. R¨ock, "The three-machine no-wait flow shop is NP-complete," *Journal of Associate Computer Machinery*, vol. 31, pp. 336–345, 1984.

[4] C. Rajendran and D. Chaudhuri, "Heuristic algorithms for continuous flow-shop problem," *Naval Research Logistic Quarterly*, vol. 37, pp. 695-705, 1990.

[5] E. Bertolissi, "Heuristic algorithm for scheduling in the no-wait flow-shop," *Journal of Materials Processing Technology*, vol. 107, pp.459–465, 2000.

[6] T. Aldowiasan and A. Allahverdi, "New heuristics for m-machine no-wait flowshop to minimize total completion time," *Omega*, vol. 32, pp. 345–352, 2004.

[7] J. M. Framinan, M. S. Nagano, and J. V. Moccellin, "An efficient heuristic for total flowtime minimization in no-wait flowshops," *International Journal of Advanced Manufacturing Technology*, vol. 46, pp. 1049-1057, 2010.

[8] M. Nawaz, E. E. Jr. Enscore, and I. Ham, "A heuristic algorithm for the *m* machine, *n* job flowshop sequencing problem," *Omega*, vol. 11, pp. 91–95, 1983.

[9] M. C. Bonney and S. W. Gundry, "Solutions to the constrainted flowshop sequencing problem," *Operational Research Quarterly*, vol. 27, pp. 869-883, 1976.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[10] J. R. King and A. S. Spachis, "Heuristics for flowshop scheduling," *International Journal of Production Research*, vol. 18, pp. 343-357, 1980.

[11] A. Fink and S. Voß, "Solving the continuous flow-shop scheduling problem by metaheuristics," *European Journal of Operational Research*, vol. 151, pp. 400–414, 2003.

[12] R. Haupt, "A survey of priority rule based scheduling," *OR Spektrum*, vol. 11, pp. 3-16, 1989.

[13] R. Ramasesh, "Dynamic jobshop scheduling: a survey of simulation research," *Omega*, vol. 18, pp. 43-57, 1990.

[14] C. Rajendran and O. Holthaus, "A comparative study of dispatching rules in dynamic flowshops and jobshops," *European Journal of Operational Research*, vol. 115, pp. 156-170, 1999.

[15] A. A. Kalir and S. C. Sarin, "A near-optimal heuristic for the sequencing problem in multiple-batch flow-shops with small equal sublots," *Omega*, vol. 29, pp. 577-584, 2001.

[16] C. Rajendran, K. Alicke, "Dispatching in flowshops with bottleneck machines," *Computers and Industrial Engineering*, vol. 52, pp. 89-106, 2007.

[17] T. H. Cormen, C. E. Leiserson, and R.L. Rivert, Introduction to algorithms, Cambridge MA, MIT press, 1990.

[18] D. Chan and D. D. Bedworth, "Design of a scheduling system for flexible manufacturing cells," *International Journal of Production Research*, vol. 28, pp. 2037-2049, 1990.

[19] D. Laha and U. K. Chakraborty, "A constructive heuristic for minimizing makespan in no-wait flow shop scheduling," *International Journal of Advanced Manufacturing Technology*, vol. 41, pp. 97-109, 2009.

[20] D. Laha and S. C. Sarin, "A heuristic to minimize total flow time in permutation flow shop," *Omega*, vol. 37, pp. 734-739, 2009.

[21] E. Kreyszig, Advanced engineering mathematics, Wiley, New York 1972.

ISBN: 978-988-18210-3-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)