Contents list avaliable at Directory of Open Access Journals (DOAJ)

JUTIN : Jurnal Teknik Industri Terintegrasi

Volume 7 Issue 3 2024, Page 1777-1783

ISSN: 2620-8962 (Online)

Journal Homepage: https://journal.universitaspahlawan.ac.id/index.php/jutin/index

Job Shop Scheduling Problem menggunakan *Ant Colony Optimization* dan Algoritme Genetika

Rizka Aulia[1]✉, Shinta Aprilisa[2]

Teknologi Informasi, Fakultas Sains dan Teknologi, Universitas PGRI Silampari, Lubuklinggau[(1)]

Sistem Informasi, Fakultas Sains dan Teknologi, Universitas PGRI Silampari, Lubuklinggau [(2)]

DOI: 10.31004/jutin.v7i3.31591

✉ Corresponding author:
[rizkaaulia1515@gmail.com]

Article Info

Kata kunci:
Ant Colony Optimization;
Algoritme Genetika;
Job Shop Scheduling Problem

Abstrak

Proses produksi memilikii permasalahan dan tantangan dalam mengelola mesin dan Sumber Daya Manusia berkerja secara optimal. Permasalahan ini muncul karena jadwal produksi tidak jelas dan tidak terartur. Solusinya, dengan melakukan penjadwalan dalam proses produksi. *Job Shop Scheduling Problem* (JSSP) merupakan suatu permasalahan untuk menentukan urutan operasi yang dilakukan dalam mesin yang ada dengan tujuan meminimumkan waktu proses total yang dibutuhkan. Perkembangan metode optimasi untuk mencapai solusi permasalahan urutan operasi mesin mendorong munculnya banyak metode penyelesaian baru. Penelitian ini ingin membandingkan dua metode penyelesaian dengan menggunakan *Ant Colony Optimization* (ACO) dan Algoritme Genetika. Dua metode tersebut dibandingkan untuk mengetahui optimasi yang paling baik digunakan untuk menyelesaikan permasalahan JSSP. Hasil dari penelitian ini menunjukkan algoritme ACO lebih baik rata-rata error rate sebesar 72.99%, dibandingankan dengan Algoritme Genetika sebesar rata-rata error rate 11.71%.

Keywords:
Ant Colony Optimization;
Genetic Algorithm;
Job Shop Scheduling Problem

Abstract

Problem (JSSP) is a problem to determine the sequence of operations carried out on existing machines with the aim of minimizing the total processing time required. The development of optimization methods to achieve solutions to machine operation sequence problems has encouraged the emergence of many new solution methods. This research wants to compare two solution methods using Ant Colony Optimization (ACO) and Genetic Algorithms. The two methods are compared to find out which optimization is best used to solve the JSSP problem. The results of this research show that the ACO algorithm is better with mean squared error of 72.99%, compared to the Genetic Algorithm with mean squared error of 11.71%.

Received 23 June 2024; Received in revised form 28 June 2024 year; Accepted 6 July 2024
Available online 10 July 2024 / © 2024 The Authors. Published by Jurnal Teknik Industri Terintegrasi Universitas Pahlawan Tuanku Tambusai.
This is an open access article under the CC BY-SA license (https://creativecommons.org/licenses/by-sa/4.0)

1. INTRODUCTION

Penjadwalan (scheduling) adalah melakukan sejumlah pekerjaan dengan mengalokasikan sumber daya yang terbatas. Penjadwalan produksi bertujuan untuk meminimasi makespan (total waktu yang dibutuhkan menyelesaikan seluruh pekerjaan (Demir & Erden, 2020). Penjadwalan dibedakan menjadi dua yaitu flowshop dan jobshop. Penjadwalan jobshop merupakan penetuan waktu operasi pada saat mulai dikerjaan dan pengalokasian resource dalam mengerjakan suatu operasi (Baker & Trietsch, 2018). Permasalahan yang sering muncul dalam penjadwalan job shop atau yang lebih dikenal Jobshop Scheduling Problem (JSSP).

JSSP merupakan suatu permasalahan dalam penentuan urutan operasi yang dilakukan dalam mesin yang ada, bertujuan meminimumkan waktu proses total yang dibutuhkan dan termasuk kategori kasus NP-hard. Karena semakin besar ukuran problem, maka waktu komputasi yang diperlukan untuk menyelesaikan problem akan semakin lama (Sayoti et al., 2016).

Banyak penelitian mengenai komputasi menggunakan metode metaheuristik maupun heuristic untuk membandingan beberapa algoritma. Perbandingan algoritma tersebut digunakan untuk mengetahui optimasi dalam pemecahan masalah pada JSSP. Metode metaheuristik yang sering digunakan antara lain, jaringan saraf tiruan, pencarian Tabu, Genetic Algorithm, Ant Colony optimization, dan lain-lain. Alfandianto et al., (2017) mengatakan Algoritme Genetika dapat meneyelesaikan masalah diberbagai bidang seperti teknologi, entertainment, bisnis untuk, model ekonomi, pemrograman otomatis dan optimasi penjadwalan. Menurut (Katoch et al., 2021) Genetic Algotithm dapat memecahkan masalah optimasi dengan melibatkan fungsi tunggal untuk memevahkan masalah multiobjective.

Algoritma lain yang dapat menyelasikan masalah optimasi yaitu algoritma semut atau *Ant Colony Optimization*. *Ant Colony Optimization* algoritma terinspirasi dari prilaku atau karakteristik dari koloni semut, dimana pada saat mencari atau membawa sumber makanan ke sarang mereka, semua akan mengeluarkan feromon pada jalur yang dilalui (Zhang et al., 2010)

Pada penelitian ini, akan dilakukan perbandingan hasil optimasi antara Algoritme Genetika dan Ant Colony Optimization untuk menyelesaikan masalah *Job Shop Scheduling Problem*. Hasil ini akan dibandingkan dengan algoritme lain yang menggunakan data sama yaitu algoritme *Golden Ball Algorithm* (Sayoti et al., 2016), *Multiple Colony Ant Algorithm* (Udomsakdigool & Kachitvichyanukul 2008), dan *Artificial Immune System* (Mahapatra, 2012).

2. METHODS

a. Job Shop Scheduling Problem

Job Shop Scheduling Problem (JSSP) yaitu permasalahan optimasi yang terkenal tidak baik, permasalahan ini termasuk NP- hard. Tujuan dari JSSP yaitu untuk menjadwalkan suatu set job dengan set mesin yang terbatas. Setiap job terdiri dari beberapa operasi. Untuk urutan mesin dari setiap pekerjaan telah ditetapkan dan ditentukan sebulumnya. Semua operasi yang dilakukan diproses selama waktu tertentu (Yu, 2021).

Formula untuk JSSP yaitu *n job* dan *m* mesin, JSSP didefinisikan sebagai suatu set *J* untuk *job* $J = \{J_{1, \ldots,} J_n\}$, yang dilakukan pemrosesan pada set *M* untuk $M = \{M_1, \ldots, M_m\}$. Setiap *job* terdiri dari beberapa oprsi *m,* dilambangkan O_{ik}, *I* didefinisikan untuk *job* dan *k* didefiniskan untuk M_k pada pengoperasian yang diproses.

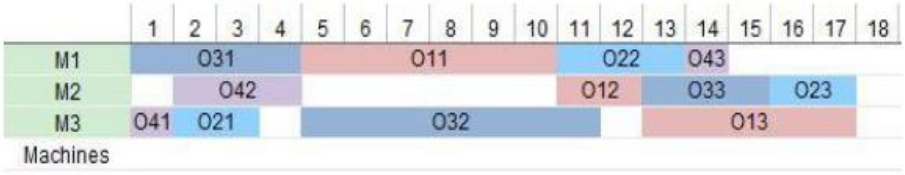
Setiap operasi harus dijalankan mengikuti *order* yang telah ditentukan dan waktu pemprosesan tidak terputus. Hanya satu operasi yang dapat diproses pada mesin pada periode waktu tertentu. Waktu penyelesai untuk setiap *job* (*maskepan* C_{max}) harus dioptimalkandengan menetukan jadwal pada *maskepan* minimum.

Matriks berikut menunjukan *tree* JSSP dengan 4 *job*:

$$\begin{pmatrix} 1\,6 & 2\,2 & 3\,5 \\ 3\,2 & 1\,3 & 2\,2 \\ 1\,4 & 3\,7 & 2\,3 \\ 3\,1 & 2\,3 & 1\,1 \end{pmatrix}$$

Setiap baris berisi nomor mesin dan waktu pemrosesan pada masing-masing operasi. Misalnya, kolom pertama dan kedua dari baris pertama (1 6) berarti operasi O_{11} diproses pada mesin nomor 1 sebanyak 6 kali, kolom ketiga dan keempat dari baris kedua (1 3) memiliki arti untuk proses O_{22} pada mesino nomor 1 sebanyak 3 kali dan seterusnya (Sayoti et al., 2016).

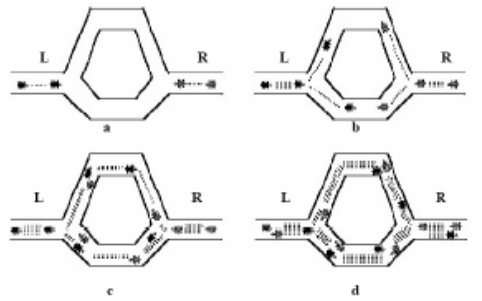
Penjadwalan (*schedule*) diwakili oleh permutasi satu set operasi pada setia mesin, di contohkan pada penjadwalan terbaik diperoleh oleh O_{31}, O_{41}, O_{42}, O_{11}, O_{21}, O_{32}, O_{12}, O_{13}, O_{22}, O_{33}, O_{23}, O_{43} merupakan minimal *maskepan* C_{max} = *17. Makespan* dihitung menggnkan reprensentasi *Gantt chart* yang ditunjukan pada Gambar 1.



Gambar 1 Representasi *Gantt Chart*

2.2 Metode Ant Colony Optimization (ACO)

Ant Colony (koloni semut) terinspirasi dari prilaku atau karakteristik dari koloni semut. dimana pada saat mencari atau membawa sumber makanan ke sarang mereka, semua akan mengeluarkan feromon pada jalur yang dilalui. Semakin banyak semut yang melalui jalur tersebut maka semakin kuat juga konsentrasi feromon pada jalur tersebut Semut dapat menemukan jalur terpendek dari sarangan ke sumber makanan, tanpa menggunakan isyarat visual tetapi dengan mengeksploitasi informasi feromon (Rui *et al.,* 2014).



Gambar 2 Perjalan semut dari sarang ke sumber makanan

Terdapat dua kelompok semut yaitu L dan R yang data dari arah yang berlainan arah. Semut pada jalur L berangkat dari sarang semut sedangkan semut R data dari sumber makanan. Kedua kolompok semut mencari jalur perjalanan terpendek yang dapat mereka lalui. Untuk setiap masingkelompok semut membagi 2 kelompok lagi dalam pencari jalur yang akan mereka lalui. Sebagian semut memilih jalur atas sebagian lagi memilih jalur bawah. Semut – semut dalam perjalananya akan meningalkan *pheromone* pada jalur yang dilalui. *Pheromone* pada jalur atas mengalami penguapan karena jaraknya yang lebih panjang sehingga konsetrasi *pheromone* berkurang. Berkurangnya konsetrasi *pheromen* ini menyebabkan berkurangnya semut yang melalui jalur tersebut.Jalur atas ditinggalkan karena *pheromone* yang ditinggalkan semut menguap dan menghilang. Sehingga banyak semut yang lalui jalur bawah dan meningkatkan jumlah *pheromone* yang ada pada jalur bawah.

Secara umum algoritma ACO menurut (Rui *et al.,* 2014) sebagai berikut:

1. Menginisialisasi parameter awal dan *pheromone trails* pada bentuk graf G (V, E).
2. Mencari kendala yang terjadi ketika mengunjungi n titik dengan titik yang pernah dilalui, titik awal sama dengan titik akhir. Untuk mengetahui jalur terpendek.
3. Pemberian nilai intensitas *pheromone* dan informasi heuristic. Pemberian nilai *pheromone* dilakukan ketika semut mengunjungi titik akhir setelah mengunjungi titik awal. Informasi heuristic adalah informasi yang menjelaskan kualitas *edge*.
4. *Tour contruction* berguna untuk mengaplikasikan prosedur sederhana. Inisialisasi penempatan *m* semut di titik *n* dengan mengunaan *state rule* secara relative. Semut akan membangun lintasan dengan memilih lintasan secara probabilistic untuk setiap titik s yang belum dikunjungi dengan itensitasi dari *pheromone*. Semut akan mencari tingkat *pheromone* yang lebih tinggi untuk membuat jalur terpendek dengan menggunakan *tabu list* (memori yang digunakan untuk membangun jalur). Setelah semut membangun *tour, pheromone* di *update* untuk mengurangi tingkat *pheromone* degan factor konstan dan meletakan *pheromonei* pada *edge* yang dilewati *update* dilakukan sehingga

lintasan yang lebih pendek dapat dilewati. Semakin banyak jumlah *pheromone* yang diterima oleh lintasan makan semakin banyak juga semut yang lewat.

```
procedure ACO Algorithm
        initialize parameter, initialize Pheromone trails
        while (terminator condition not met) do
                ContructuctANtSolutions
                ApplyLocalSearch (optional)
                UpdatePheromones
        Endwhile
End procedure ACO Algorithm
```

2.3 Algoritme Genetika

Algoritme Genetika adalah algoritma kecerdasan buatan mengenai optimasi dan teknik pencarian berdasarkan evolusi dan seleksi yang terjadi pada alam (Katoch et al., 2021). Pada algoritme genetika kromosom dipresentasikan sebagai individu. Salah satu permasalahan yang dapat diselesaikan oleh Algoritme Genetika yaitu dapat melakukan pencarian kromosom terbaik dengan memanipuasi isi dalam kromosom dengan melakukan apa yang terjadi pada alam. Evaluasi yang dihasilkan berguna dalam memcari kromosom terbaik untuk dikembangkan lebih banyak.

Genetic Algorithm memiliki lima komponen yaitu:

1. Mempresentasikan genetika sebagai solusi potensial dalam permasalahan.
2. metode untuk membuat populasi awal dari metode.
3. Parameter nilai yang bervariasi: banyak gen dalam kromosom, kromosom, laju *crossover*, dan laju mutasi.
4. Operator – operator genetika yaitu *crossover* (perkawinan silang) dan mutasi.
5. Evaluasi.

```
Begin
Generate initial population Pt
Evaluate population Pt
While stopping criteria not satisfied Repeat
        Begin
                Pt = Pt+l
                Select elements from Pt-1 to copy into Pt
                Crossover elements of Pt
                Mutation elements of Pt
                Evaluate new population Pt+l
        End;
End;
```

Dari *procedure* diatas dapat dijelaskan lakukan inisialisasi awal Pt = 0. Pembentukan populasi awal kromosom dapat dilakukan dengan pemilihan metode tertentu atau secara random. Setiap kromosom mempresentasikan kandidat solusi pada setiap permasalahan. *Evaluate population* Pt digunakan untuk mengevalusi kromosom pada populasi awal dengan nilai *fitness*. *While stopping criteria not satisfied Repeat* menunjukan ketika kondisi belum memenuhi kriteria, maka masuk dalam *loop* proses genetika. Masukan kondisi Pt = Pt+1. *Select elements from Pt-1 to copy into Pt* digunakan untuk memilih kromosom berdasarkan nilai *fitness*, semakin besar nilai *fitness* maka semakin besar kemungkinan diseleksi. Populasi terbentuk berdasarkan pemilihan secara acak. *Crossover elements of Pt* digunakan untuk menggabungkan dua kromoso menjadi dua keeruan, dengan penentuan titik *crossover* secara acak. *Mutation elements of Pt* bertujuan untuk proses pembuhan satu atau lebih gen. *Evaluate new population Pt+l* digunakan untuk memeriksa kromosom yang terbaik.
Karakteristk *Genetic Algorithm* (Xue et al., 2021) yaitu:

1. Mengkodekan satu parameter, bukan per parameter.

2. Proses pencarian dilakukan pada sekumpulan titik bukan pada titik tunggal. Proses pencarian dilakukan dari generasi awal sampai generasi yang menghasilkan solusi potensial.
3. Penggunaan fungsi *fitness*. Nilai fungsi *fitness* kembali pada pencarian secara langsung**.**

3. RESULT AND DISCUSSION

Program GA dan ACO diimplementasikan dalam bahasa Python 3 dan akan dilakukan beberapa simulasi terhadap data benchmark instances JSSP. Pada penelitian ini, dibagi menjadi tiga skenario. Skenario pertama, data *instances* diselesaikan hanya dengan menggunakan GA dan dilakukan beberapa kali percobaan (*per instances*) untuk mendapatkan nilai rata-rata. Skenario kedua, ACO digunakan untuk menyelesaikan data instances yang sama dan bertujuan untuk mengetahui parameter-parameter mana yang menghasilkan *output* paling optimal. Skenario ketiga, kedua algoritme dibandingkan dengan algoritme lain dengan menggunakan parameter-parameter optimum dari segi *makespan time*, *error rate*, dan waktu eksekusi program.

Terdapat 10 data *benchmark instances* yang akan diselesaikan pada simulasi-simulasi ini, yaitu ABZ5, ABZ6, FT06, FT10, LA01, LA02, LA03, LA04, LA05, LA16 (Sayoti et al., 2016).

Skenario 1: Penyelesaian JSSP dengan *Genetic Algorithm* (GA)

Simulasi-simulasi ini dilakukan menggunakan parameter-parameter yang ditentukan dengan nilai tetap. Parameter-parameter tersebut adalah jumlah populasi sebanyak 30, nilai *crossover rate* sebesar 0.7, peluang mutasi (*mutation rate*) sebesar 0.1, dan jumlah pengulangan sebanyak 1500 iterasi. Untuk setiap *instances* dilakukan 5 kali percobaan simulasi. Dengan mengambil solusi terbaik, solusi rata-rata, dan solusi terburuk dari 5 simulasi yang dilakukan pada tiap *instance*, berikut adalah hasil dari simulasi tersebut.

Tabel 1. Hasil simulasi penyelesaian data JSSP dengan Algoritme Genetika

Instance	*Size (JxM)*	*BKS*	*Genetic Algorithm*				
			Best	*Average*	*Worst*	*% error rate*	*Time elapsed*
ABZ5	10x10	1234	1442	1472	1507	19.30	130.0
ABZ6	10x10	943	1066	1099	1129	16.59	168.7
FT06	6x6	55	55	56	58	2.55	55.2
FT10	10x10	930	1153	1167	1185	25.48	156.7
LA01	10x5	666	683	698	722	4.77	73.1
LA02	10x5	655	706	726	748	10.81	74.1
LA03	10x5	597	651	664	681	11.26	79.0
LA04	10x5	590	628	645	675	9.29	84.8
LA05	10x5	593	593	593	593	0.00	87.5
LA16	10x10	945	1092	1106	1120	17.04	146.6
Average error rate:						11.71%	
J = Jumlah tugas (*jobs*)						BKS = *Best Known Solution*	
M = Jumlah mesin (*machines*)							

Tabel 1 menunjukkan bahwa dari setiap *instances* yang diselesaikan menggunakan Algoritme Genetika menghasilkan rata-rata persentase *error rate* yang cukup tinggi, kecuali pada instance LA05 yang solusi tepatnya berhasil ditemukan oleh Algoritme Genetika. Dari Tabel 1, nilai persentase *error rate* yang paling rendah selain 0% adalah pada instance FT06 yaitu sebesar 2.55%, sampai yang tertinggi pada instance FT10 dengan persentase mencapai 25.48%. Hal ini dapat disebabkan karena semakin banyak total tugas dan mesin dari suatu *instance*, maka akan menyebabkan proses penyelesaiannya menjadi semakin kompleks dan *error* yang dihasilkan menjadi semakin besar. Waktu proses penyelesaian instance juga bergantung pada kompleksitas dari *instances* tersebut. Memungkinkan juga bahwa Algoritme Genetika tidak cocok untuk kasus JSSP dengan *size* yang besar karena semakin banyak jumlah gen yang ada dalam satu kromosom, yang menjadikan Algoritme Genetika ini sangat sensitif terhadap perubahan.

Skenario 2: Penyelesaian JSSP dengan *Ant Colony Optimization* (ACO)

Simulasi-simulasi ini dilakukan terhadap 10 *instances* menggunakan parameter-parameter yang ditentukan dengan nilai tetap, yaitu jumlah semut sebanyak 25, jumlah pengulangan sebanyak 1500 iterasi,

konstanta decay sebesar 0.2, batasan eksploitasi ($Q_{threshold}$) sebesar 0.9, dan bobot feromon dan jarak sebesar 1.

Tabel 2. Hasil simulasi penyelesaian data JSSP dengan ACO

Instance	*Size (JxM)*	*BKS*	*Genetic Algorithm*				
			Best	*Average*	*Worst*	*% error rate*	*Time elapsed*
ABZ5	10x10	1234	2065	2143	2201	73.70	235.5
ABZ6	10x10	943	1597	1646	1683	74.57	221.2
FT06	6x6	55	74	77	81	39.64	58.4
FT10	10x10	930	1794	1825	1868	96.19	229.4
LA01	10x5	666	1100	1109	1143	66.58	100.3
LA02	10x5	655	1057	1071	1082	63.57	100.4
LA03	10x5	597	1104	1128	1144	88.88	100.7
LA04	10x5	590	1038	1075	1090	82.17	101.9
LA05	10x5	593	969	999	1025	68.40	100.4
LA16	10x10	945	1644	1665	1697	76.23	234.2
Average error rate:						72.99%	
J = Jumlah tugas (*jobs*)					BKS = *Best Known Solution*		
M = Jumlah mesin (*machines*)							

Tabel 2 menunjukkan bahwa dari seluruh *instances* yang diselesaikan menggunakan ACO menghasilkan rata-rata persentase *error rate* yang sangat tinggi. Dari yang paling terkecil yaitu pada *instance* FT06 sebesar 39.64% sampai yang terbesar pada *instance* FT10 dengan *error rate* yang mencapai 96.19%. Hal ini dapat disebabkan oleh kesalahan representasi masalah *instance* dan/atau konsep dari JSSP itu sendiri terhadap program ACO yang telah dikembangkan. Waktu proses program ACO terhadap seluruh *instances* juga lebih besar dibandingkan waktu proses program GA.

Pembandingan dengan Algoritme Lain

Hasil-hasil yang telah didapatkan sebelumnya dibandingkan dengan hasil algoritme lain yang diambil dari penelitian-penelitian yang terkait. Berikut adalah tabel perbandingan algoritme GA, ACO, dengan algoritme lainnya.

Tabel 3. Perbandingan Hasil Algoritme Genetika, Aco dengan algoritme lain

Metode	ABZ5	ABZ6	FT06	FT10	LA01	LA02	LA03	LA04	LA05	LA16
Genetic Algorithm	1442	1066	55	1153	683	706	651	628	593	1092
Ant Colony Optimization	2065	1597	74	1794	1100	1057	1104	1038	969	1644
Golden Ball Algorithm	1234	943	55	946	666	655	597	590	593	945
Multiple Colony Ant Algorithm	-	-	55	944	666	658	603	590	593	977
Artificial Immune System	-	-	55	930	666	655	597	590	593	-

Tabel 3 menggambarkan hasil perbandingan antara GA, ACO, dengan algoritme lain dari penelitian yang terkait. Hasil dari GA dan ACO yang dikembangkan pada penelitian ini dapat dikatakan buruk dibandingan dengan algoritme lainnya. Sehingga, GA dan ACO tidak baik digunakan terhadap masalah JSSP terkecuali masalah tersebut dapat direpresentasikan dengan baik.

4. CONCLUSION

Simpulan dari penelitian ini adalah program GA yang dikembangkan menghasilkan rata-rata error rate sebesar 11.71%, sedangkan program ACO yang dikembangkan menghasilkan rata-rata sebesar 72.99%. Kedua algoritme menghasilkan rata-rata error rate yang cukup tinggi. Hal ini dapat disebabkan oleh kesalahan representasi masalah JSSP dari data *benchmark instances* ke dalam bentuk lain untuk diproses pada kedua algoritme. Disarankan untuk penelitian selanjutnya, bahwa perlunya pakar untuk merepresentasikan masalah JSSP ke dalam bentuk yang dapat diproses oleh tiap algoritme.

5. REFERENCES

Alfandianto, A., Nugroho, Y. A., & Setiafindari, W. (2017). Penjadwalan Produksi Menggunakan Pendekatan Algoritma Genetika di PT Pertani (Persero) Cabang D.I. Yogyakarta. *Jurnal DISPROTEK*, *8*(2), 1–7.

B Baker, K.R., & Trietsch, D. (2018). Principles of sequencing and scheduling. 2nd edn. Wiley.

Demir, H. I., & Erden, C. (2020). Dynamic integrated process planning, scheduling and due-date assignment using ant colony optimization. *Computers and Industrial Engineering*, *149*(September), 106799. https://doi.org/10.1016/j.cie.2020.106799

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). Katoch2021_Article_AReviewOnGeneticAlgorithmPastP.pdf. In *Multimedia Tools and Applications* (Vol. 80). Multimedia Tools and Applications.

Mahapatra DK. 2012. Job shop scheduling using artificial immune system [tesis]. Rourkela (IN): National Institute of Technology.

Rui, Z., Shilong, W., Zheqi,Z., Lili, Y. (2014).An ant colony algorithm for job shop scheduling problem with tool flow. in Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, 228(8). Available at: https://doi.org/10.1177/0954405413514398.

Sayoti, F., Riffi, M. E., & Labani, H. (2016). Optimization of makespan in job shop scheduling problem by golden ball algorithm. *Indonesian Journal of Electrical Engineering and Computer Science*, *4*(3), 542–547. https://doi.org/10.11591/ijeecs.v4.i3.pp542-547.

Udomsakdigool A, Kachitvichyanukul V. 2008. Multiple colony ant algorithm for job-shop scheduling problem. *International Journal of Production Research*. 46(15): 4155-4175.

Xue, Y., Zhu, H., Liang, J., & Słowik, A. (2021). Adaptive crossover operator based multi-objective binary genetic algorithm for feature selection in classification[Formula presented]. *Knowledge-Based Systems*, *227*(June). https://doi.org/10.1016/j.knosys.2021.107218

Yu, Y. (2021). A Research Review on Job Shop Scheduling Problem. *E3S Web of Conferences*, *253*, 1–5. https://doi.org/10.1051/e3sconf/202125302024

Zhang, Z. Q., Zhang, J., Zhang, X., & Li, S. J. (2010). Improved ant colony optimization algorithm for job shop scheduling problem. *Yingyong Kexue Xuebao/Journal of Applied Sciences*, *28*(2), 182–188.