Journal of Systems Engineering and Electronics
Vol. 37, No. 3, June 2026, pp.897 – 903

Heterogeneous multi-core task scheduling based on adaptive simulated annealing

ZHU Guoliang[1], ZHANG Jinjian[2], LIU Xuan[2], WANG Guojun[2], ZHANG Xiaodong[2], CHEN Kaiyu[3], and WANG Yu[3,*]

1. School of Information and Electronics, Beijing Institute of Technology, Beijing 100081, China;
2. Beijing Research Institute of Telemetry, Beijing 100076, China;
3. School of Microelectronics, Northwestern Polytechnical University, Xi'an 710072, China

Abstract: To address the energy consumption issues caused by task lengths in task scheduling on heterogeneous multi-core systems, this paper proposes an adaptive parameterized improved simulated annealing algorithm based on the directed acyclic graph task model. The algorithm employs feedback from acceptance rates to dynamically adjust the temperature and neighborhood size of the simulated annealing process. Additionally, it introduces a security mechanism to enhance convergence speed and global search capabilities. Compared against classical simulated annealing and standard heuristic algorithms, the proposed algorithm achieves reductions exceeding 54% in convergence generations, 50% in task slots, and 10% in scheduling time, providing a direction for low-power task scheduling.

Keywords: heterogeneous multi-core, task scheduling, schedule length, simulated annealing.

DOI: 10.23919/JSEE.2026.000042

1. Introduction

Heterogeneous multi-core processor chips [1], a prominent example of integrated circuits [2], find extensive applications in various emerging technologies [3], including artificial intelligence (AI) [4], big data [5], cloud computing [6], and the Internet of Things (IoT) [7]. The efficient scheduling of applications onto multiple processors for execution [8], while controlling system energy consumption through energy-saving techniques [9] and ensuring rational resource utilization alongside the high-performance advantages of multiprocessor systems [10], represents a critical focus in both academic and engineering domains. Employing global optimization algorithms to solve the problem of low-power task scheduling in multiprocessor systems demonstrates high efficiency and practicality [11]. Compared to non-heuristic algorithms [12], heuristic algorithms are widely used for solving multiprocessor energy optimization problems due to their advantages of simple implementation [13], strong adaptability, and fast computational speed [14–18].

The energy consumption of heterogeneous multi-core systems is co-determined by multiple factors, such as voltage and frequency [19], communication power [20], schedule length, and computational power [21]. Among these, the task schedule length significantly influences the total system energy consumption [22–24]. Consequently, reducing the schedule length represents an imperative need for addressing the energy optimization problem in task scheduling [25].

Currently, numerous researchers have conducted explorations and studies in the field of low-power task scheduling for heterogeneous multi-core systems based on classical heuristic algorithms [26], leading to the development of various hybrid and adaptive scheduling algorithms [27].

Zhou et al. [28] proposed an improved genetic-simulated annealing (SA) algorithm for scheduling ordered tasks with task duplication. In the enhanced genetic algorithm, the SA approach is employed to mitigate selection pressure, strengthen global convergence, and prevent the search process from being trapped in local optima. Cheng et al. [29] introduced a task scheduling algorithm for heterogeneous multi-core processors based on the crazy and adaptive salp swarm algorithm (CASSA). Aiming to minimize the overall task completion time, this algorithm designs an encoding scheme for task allocation according to task priority rules, demonstrating higher convergence efficiency and superior solution quality in addressing heterogeneous multi-core task scheduling problems.

Manuscript received October 23, 2025.
*Corresponding author.

Feng et al. [30] presented an evolutionary adaptive bat algorithm (EABA), which incorporates an attenuated pulse strategy and an evolutionary adaptive mechanism. These strategies enhance the search capability of the bat population, prevent premature convergence to local optima, and thereby yield higher-quality solutions.

Given that existing heuristic scheduling algorithms for minimizing task completion time seldom consider flexible handling of dynamic changes or imperfect information in scheduling scenarios [31], thus limiting their practical applicability, this paper proposes an improved adaptive SA (ASA) algorithm to generate schedules with minimized execution time. By dynamically adjusting both temperature and neighborhood size parameters while incorporating a safety mechanism, the proposed approach enhances convergence speed and global search capability, ultimately achieving reduced schedule length and lower system energy consumption.

2. Task scheduling model

2.1 Task model

This paper uses a directed acyclic graph (DAG) to represent the application model. We define a set $M=\{m_1, m_2,\cdots,m_p\}$ to denote the multiprocessor cores in the system. Fig. 1 shows an example application modeled as a DAG, containing 11 subtasks where each node represents one. The set of all subtasks is defined as $N=\{n_1,n_2,\cdots,n_q\}$. Due to processor heterogeneity, the execution time of a given subtask differs across processors, even at their maximum frequency. Consequently, we introduce a matrix lengths of size $p\cdot q$, whose element $\text{lengths}(i,j)$ represents the execution time of subtask n_j on processor m_i.

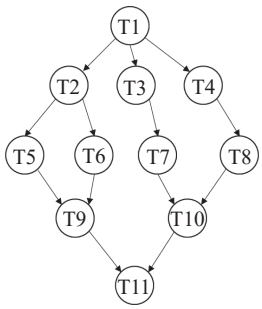

Fig. 1 Application model representation by DAG

2.2 Task scheduling and allocation model

The optimization process for task schedule length in heterogeneous multi-core processors, as illustrated in Fig. 2, aims to assign n subtasks $N=\{n_1,n_2,\cdots,n_q\}$ of an application to m distinct processing cores $M=\{m_1,m_2,\cdots,m_p\}$ according to a specific scheduling scheme. This is done under the premise of ensuring execution efficiency, with the goal of identifying the shortest possible schedule length through a scheduling algorithm, thereby achieving reduced energy consumption and efficient resource utilization.

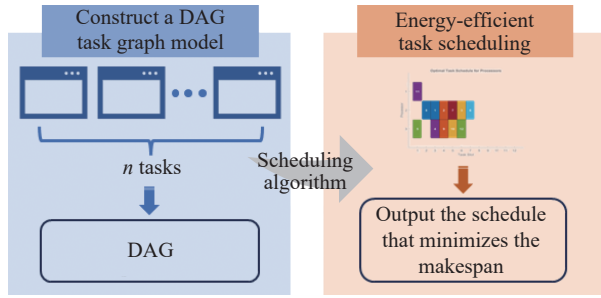

Fig. 2 Energy-efficient task scheduling process schematic

In matrix lengths, each row corresponds to one processor, and each column corresponds to one subtask. Therefore, element $\text{lengths}(i,j)$ represents the processing time of subtask n_j on processor m_i. A scheduling matrix $\textbf{schedule}\in\mathbf{Z}^{p\times q}$ is defined, where each row represents the task queue of a processor. Element $\text{schedule}(i,k)=j(j\neq 0)$ indicates that location k on processor m_i is allocated to task n_j, while element $\text{schedule}(i,k)=0$ indicates that location k is unassigned.

For a given processor m_i, all assigned task indices are retrieved as follows:

$$S_i=\{j|\text{schedule}(i,k)=j,k=1,2,\cdots,q\}. \tag{1}$$

The total processing time is given by the cumulative sum of the processing times for all its tasks:

$$T_i=\sum_{j\in S_i}\text{lengths}(i,j). \tag{2}$$

The makespan, which is the maximum completion time among all processors, serves as the scheduling length for the subtask set and constitutes the objective function value:

$$\text{makespan}=\max(T_1,T_2,\cdots,T_p). \tag{3}$$

3. ASA scheduling algorithm

The classical SA algorithm offers distinct advantages in processor task scheduling problems due to its powerful global optimization capability, flexibility in handling complex constraints, and dynamic balance between exploration and exploitation. However, when applied to task scheduling on heterogeneous multi-core processors, the classical SA exhibits several limitations: a fixed cooling rate often leads to insufficient high-temperature

phases or excessively long low-temperature phases; a fixed neighborhood size struggles to balance global search and local optimization; and uncontrolled parameters can cause algorithmic instability. This paper proposes an ASA scheduling algorithm. In addition to introducing dynamic temperature and neighborhood adjustment strategies guided by the monitoring of acceptance rate feedback to balance exploration and exploitation and avoid premature convergence, the algorithm also incorporates a safety mechanism and an adaptive cooling strategy. The optimized scheduling algorithm demonstrates significant superiority over classical SA in terms of convergence speed, solution quality, robustness, and interpretability, making it particularly suitable for multi-processor task scheduling problems.

3.1 Algorithmic framework

Fig. 3 shows the flowchart for the ASA scheduling algorithm.

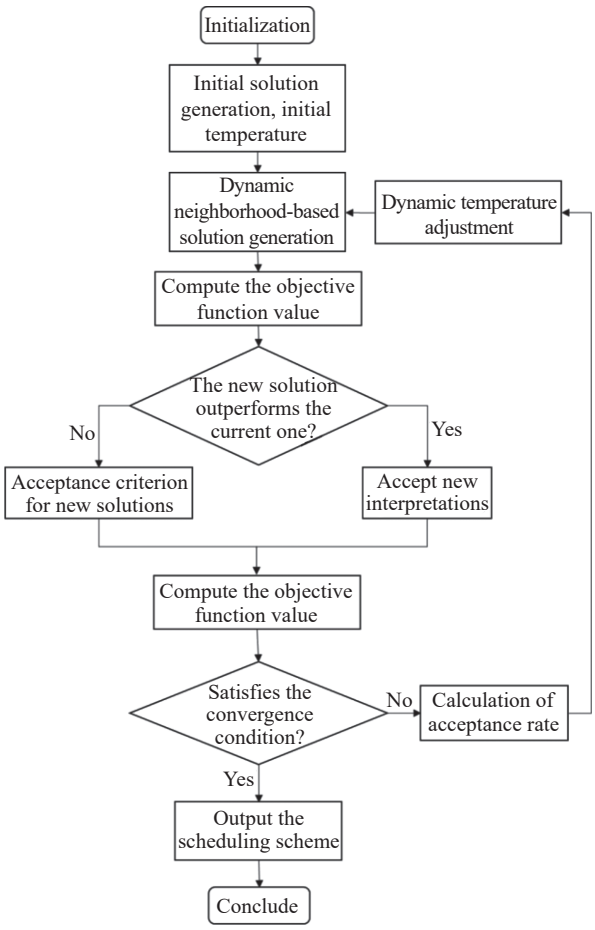

Fig. 3 ASA scheduling algorithm flowchart

The algorithm is described as follows:

Step 1 An initial solution is generated by allocating tasks uniformly across processors to ensure load balancing. The algorithm parameters are set as follows: initial temperature $T_0 = 20\,000$, neighborhood size $N_{\text{size}} = 5$, cooling rate $\alpha = 0.95$, and target acceptance ratio $\beta = 0.25$.

Step 2 New solutions are generated by a dynamic neighborhood strategy. The strategy for dynamic neighborhood adjustment is selected based on the acceptance rate feedback, employing hybrid neighborhood operations (such as task relocation or task exchange), while ensuring solution validity.

Step 3 The dynamic temperature adjustment strategy adaptively tunes its parameters by comparing the current and target acceptance rates to select an appropriate strategy.

Step 4 The SA algorithm commences its iterative process. The loop is exited immediately upon satisfaction of the convergence criteria, outputting the incumbent optimal scheduling solution.

3.2 Adaptive neighborhood strategy

Neighborhood size defines the scope of new solutions generated per iteration, with larger sizes enabling exploration and smaller ones favoring local refinement. Acknowledging the limitation of classical SA, its reliance on a fixed neighborhood size N_{size} and a single operation, which hampers the exploration-exploitation balance, this paper introduces a strategy that dynamically adjusts the neighborhood size and hybridizes operations according to the current acceptance rat P_{accept}.

The neighborhood size is dynamically adjusted as a function of P_{accept} to balance exploration and exploitation. An increase in P_{accept} leads to a larger neighborhood, thereby promoting exploration; conversely, the neighborhood is narrowed to concentrate on exploitation. The formula governing this adjustment is described below:

$$N_{\text{size}}^{\text{new}} = N_{\text{size}} + 2\left(P_{\text{accept}} - \beta\right). \tag{4}$$

The constant 2, which serves as the sensitivity coefficient, controls the magnitude of the adjustment. P_{accept} represents the current acceptance rate, defined as the proportion of solutions accepted within the current window. Parameter $N_{\min} = 2, N_{\max} = 12$ is defined to ensure that the neighborhood size remains within a reasonable range, thereby preventing excessive adjustments.

To mitigate oscillations and prevent abrupt changes, the actual neighborhood size is updated using a weighted average approach.

$$N'_{\text{size}} = 0.3N_{\text{size}}^{\text{new}} + 0.7N_{\text{size}} \tag{5}$$

In SA, a neighborhood operation refers to the mechanism employed to generate new candidate solutions from

the current solution. Common neighborhood operations include task moving (relocating a task from one processor to another) and task swapping (exchanging tasks between two processors). This paper adopts a hybrid neighborhood operation that integrates the move operation with the swap operation. Specifically, a random number $\mu \in (0,2)$ is generated. When condition $\mu \leqslant 1$ is met, the task move operation is executed, transferring a task from a heavily-loaded processor to a lightly-loaded one to optimize load balancing. When condition $\mu > 1$ is satisfied, the task swap operation is performed, which randomly exchanges tasks to enhance the diversity of the solution space, thereby mitigating the issue of low search efficiency associated with using a single neighborhood operation.

3.3 Adaptive temperature policy

In SA, the temperature parameter controls the probability of accepting inferior solutions. At high temperatures, the algorithm accepts more suboptimal solutions, facilitating a broad exploration of the solution space; at low temperatures, it shifts focus to local refinement, driving convergence towards high-quality solutions. The dynamic temperature adjustment strategy aims to adaptively optimize the cooling rate in real-time through a feedback mechanism, enabling the algorithm to autonomously balance the trade-off between exploration and exploitation during the search process.

The temperature update formula of the proposed dynamic temperature adjustment strategy is as follows:

$$T_{\text{new}} = \max\{100, T_{\text{old}} \cdot \exp[C_R \cdot (T_F - 1)]\}. \tag{6}$$

The constant 100 ensures that the temperature does not fall below this threshold, thereby preventing the algorithm from stalling due to an excessively small value.C_R is a dynamically adjusted cooling rate, which assumes different values based on P_{accept}, as defined below:

$$C_R = \begin{cases} 0.98, & P_{\text{accept}} < 0.1 \\ 0.85, & P_{\text{accept}} > 0.4 \\ 0.93, & P_{\text{accept}} \in [0.1, 0.4] \end{cases}. \tag{7}$$

When condition $P_{\text{accept}} < 0.1$ leads to outcome $C_R = 0.98$, a slower cooling rate is employed, with the high-temperature stage prolonged to facilitate sufficient exploration. Conversely, when condition $P_{\text{accept}} > 0.4$ results in outcome $C_R = 0.85$, the cooling rate is accelerated to achieve rapid convergence to high-quality solutions. Under condition $C_R \in [0.1, 0.4]$, the default cooling rate (0.93) is applied to maintain a balance between exploration and exploitation.

Factor T_F, which is a scaling factor computed based on the difference between P_{accept} and β, is defined as follows:

$$T_F = 1 + \frac{\beta - P_{\text{accept}}}{\max(\beta, \varepsilon)} \tag{8}$$

where ε is a small constant to prevent division by zero. As derived from (8), when $\beta > P_{\text{accept}}$ leads to $T_F > 1$, the exponent term in (7) becomes greater than 0, resulting in a temperature increase; conversely, it leads to a temperature decrease.

4. Simulation and experimental analysis

4.1 Verification environment and experimental setup

To validate the superiority of the proposed ASA scheduling algorithm in the field of heterogeneous multi-core low-power computing, experiments are conducted utilizing the Windows Subsystem for Linux (WSL) running the Ubuntu 22.04.3 LTS distribution to create DAG models via the TGFF tool. The host system is Windows 10 Version 22H2. Subsequent simulations are performed under the Windows 10 operating system, with the hardware platform configured with an Intel Core i7-12700H processor and 32 GB of RAM.

This study designs two sets of simulation experiments to verify the effectiveness of the proposed algorithm. The first set comprises a comparative experiment on convergence generation and the number of task slots, while the second set involves a comparative analysis of scheduling time. The benchmark for comparison in these experiments is the classical SA algorithm.

The experimental parameters are configured as follows: initial temperature $T_0 = 20\,000$, neighborhood size $N_{\text{size}} = 5$, cooling rate $\alpha = 0.95$, target acceptance rate $\beta = 0.25$.

4.2 Impact of task slot number on convergence iterations

To evaluate the convergence generation of the proposed algorithm compared to classical SA in multi-processor task scheduling, as well as the effectiveness of the resulting optimal schedule in terms of the number of task slots per processing core, five distinct DAGs are randomly generated using the TGFF tool. Five comparative experiments are conducted accordingly.

Fig. 4 presents a comparison between SA and the proposed algorithm regarding the convergence generation and the number of task slots per core for the five randomly generated task sets. Here, ASA denotes the algorithm proposed in this paper. It can be observed that, when considering both convergence speed and task slot allocation, the proposed algorithm achieves a balanced

low-power demand across different metrics in all experimental groups, demonstrating significantly higher efficiency compared to SA.

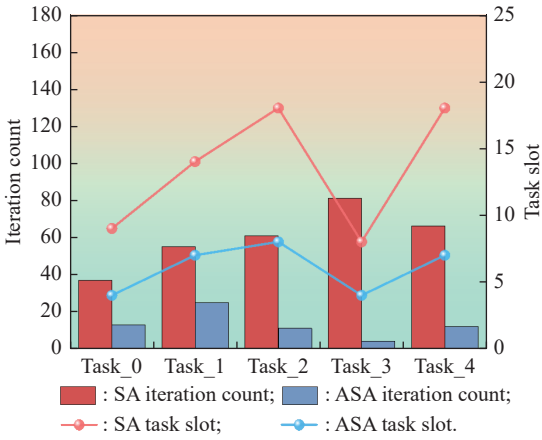

Fig. 4 Convergence generation and task slots comparison

Table 1 and Table 2 respectively demonstrate the superior performance of the proposed algorithm over SA in terms of convergence generation and the number of task slots. The results show that across five randomly generated DAG task scheduling scenarios, ASA reduces the convergence generation by more than 54% (up to 95%) and decreases the number of task slots by over 50% compared to SA.

Table 1 Convergence generation comparison

Algorithm	0	1	2	3	4
SA	37	55	61	81	66
ASA	13	25	11	4	12

Table 2 Task slots comparison

Algorithm	0	1	2	3	4
SA	9	14	18	8	18
ASA	4	7	8	4	7

4.3 Scheduling time comparison experiment

To demonstrate the optimization effect of ASA compared to SA in terms of scheduling time in task scheduling, as well as its effective balance among the three metrics convergence generation, number of task slots, and scheduling time, a comparative experiment on scheduling time is conducted using the five DAG task graphs from Subsection 3.2.

Fig. 5 illustrates the comparison of scheduling times between ASA and SA. It can be observed that the proposed algorithm demonstrates consistent effectiveness in reducing scheduling time across randomly generated task scheduling scenarios, meeting energy-saving requirements for tasks of various complexity levels. As shown in Table 3, which compares scheduling times before and after algorithm optimization, ASA reduces scheduling time by over 20% compared to SA in tasks with lower complexity, while for highly complex tasks, the reduction remains above 10%. These results further validate the significant superiority of the proposed algorithm in global optimization capability and its high stability when applied to task scheduling across varying complexity levels.

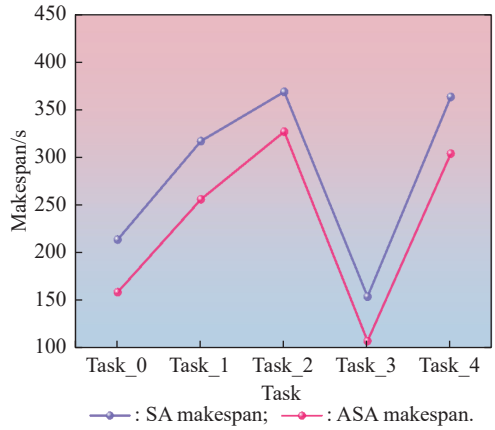

Fig. 5 Scheduling time comparison

Table 3 Makespan comparison s

Algorithm	0	1	2	3	4
SA	213.5	317.3	369.2	153.4	363.8
ASA	158.1	255.9	327.3	106.6	304.1

5. Conclusions

This study addresses the issue of low-power task scheduling in current heterogeneous multi-core processors. Considering the significant impact of task scheduling length on energy consumption, a mathematical model for optimizing task scheduling length is established based on the DAG. To solve this model, an improved SA scheduling algorithm with adaptive parameters is proposed. By utilizing feedback from the current acceptance rate, the algorithm dynamically adjusts the neighborhood size and temperature, while incorporating safety mechanisms such as neighborhood size clamping and minimum temperature protection. These enhancements lead to accelerated convergence speed and improved global optimization capability. Experimental results demonstrate that the proposed algorithm is well-suited for low-power task scheduling in heterogeneous multi-core processors,

exhibiting significant superiority across various performance metrics, along with the flexibility to be applied to complex engineering optimization problems.

However, although the proposed adaptive scheduling algorithm achieves favorable results in optimizing task scheduling length for heterogeneous multi-core processors, it does not fully account for other multi-processor low-power task scheduling problems with complex constraints. In practical applications, dependency constraints among tasks and other conditional restrictions can substantially influence task scheduling outcomes. Therefore, future research will focus on solving multi-processor low-power task scheduling problems under complex constraints.

References

[1] XIE X Y, REN X, ZHU Y, et al. A resource-adaptive tensor decomposition method for convolutional neural networks. High Technology Letters, 2025, 31(4): 355–364.

[2] FANG J, LIN S, YANG H J, et al. A perceptual and predictive batch-processing memory scheduling strategy for a CPU-GPU heterogeneous system. Frontiers of Information Technology & Electronic Engineering, 2023, 24(7): 994–1007.

[3] SONG Y, DAI H, JIANG J, et al. Multikernel: operating system solution to generalized functional safety. Security and Safety, 2023, 2(2): 30−43.

[4] YAN J, TANG F F, ZHANG Z G, et al. Low-power design for heterogeneous multi-core AI SoC chip. Aerospace Control, 2020, 38(2): 6268. (in Chinese)

[5] LI B, ZHOU Q L, SI X M, et al. Design method of big data high-efficiency platform based on mimic computing. Application Research of Computers, 2019, 36(7): 2059–2064. (in Chinese)

[6] JIA G Y, HAN G J, JIANG J F, et al. Dynamic resource partitioning for heterogeneous multi-core-based cloud computing in smart cities. IEEE Access, 2016, 4: 108–118.

[7] QIU T, ZHAO A Y, MA R X, et al. A task-efficient sink node based on embedded multi-core SoC for Internet of Things. Future Generation Computer Systems, 2018, 82: 656–666.

[8] NING J H, YANG T T, ZHENG C, et al. DeAOff: dependence-aware offloading of decoder-based generative models for edge computing. China Communications, 2025, 22(7): 14−29.

[9] KIM Y G, KIM M, CHUNG S W. Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors. IEEE Trans. on Computers, 2017, 66(11): 1878–1889.

[10] ZHANG S B, GAO H Y, SU Y M, et al. Physical-layer secure hybrid task scheduling and resource management for fog computing IoT networks. Journal of Systems Engineering and Electronics, 2025, 36(5): 1146–1160.

[11] ZHOU Z, LI F M, ZHU H X, et al. An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments. Neural Computing and Applications, 2020, 32: 1531–1541.

[12] ZHOU Y H, ZENG W, ZHENG Q F, et al. A survey on task scheduling of CPU-GPU heterogeneous cluster. ZTE Communications, 2024, 22(3): 83−90. (in Chinese)

[13] MADNI S H H, ABD LATIFF M S, ABDULLAHI M, et al. Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment. PloS One, 2017, 12(5): e0176321.

[14] XIE G Q, XIAO X R, PENG H, et al. A survey of low-energy parallel scheduling algorithms. IEEE Trans. on Sustainable Computing, 2021, 7(1): 27–46.

[15] WU J, CHEN Y N, HE Y M, et al. Survey on autonomous task scheduling technology for Earth observation satellites. Journal of Systems Engineering and Electronics, 2022, 33(6): 1176–1189.

[16] YANG T T, ZHANG Y Q, YUE F L, et al. D-scheduler: a scheduler in time-triggered distributed system through decoupling dependencies between tasks and messages. Science China (Technological Sciences), 2024, 67(1): 183−196. (in Chinese)

[17] LIU W, JIN Y F, ZHANG L, et al. Dynamic access task scheduling of LEO constellation based on space-based distributed computing. Journal of Systems Engineering and Electronics, 2024, 35(4): 842–854.

[18] YANG M, WANG Y, WANG J, et al. Optimal scheduling of integrated energy systems in low-carbon communities considering flexibility of resources and segmental control of solid oxide fuel cells. Journal of Modern Power Systems and Clean Energy, 2025, 13(3): 915−927.

[19] LI K L, TANG X Y, LI K Q. Energy-efficient stochastic task scheduling on heterogeneous computing systems. IEEE Trans. on Parallel and Distributed Systems, 2013, 25(11): 2867–2876.

[20] HAO P T, XIAO F, HUANG S J, et al. The multi-type DAG task scheduling method based on communication overhead. Microelectronics & Computer, 2024, 41(5): 67–75. (in Chinese)

[21] QUAN Z, WANG Z J, YE T, et al. Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems. IEEE Trans. on Parallel and Distributed Systems, 2019, 31(5): 1165–1182.

[22] YANG P, HOU J W, YU L, et al. Edge-coordinated energy-efficient video analytics for digital twin in 6G. China Communications, 2023, 20(2): 14−25.

[23] LONG T Y, MA Y, WU L, et al. A novel fault-tolerant scheduling approach for collaborative workflows in an edge-IoT environment. Digital Communications and Networks, 2022, 8(6): 911−922.

[24] LI W M, HAN D, GAO L, et al. Integrated production and transportation scheduling method in hybrid flow shop. Chinese Journal of Mechanical Engineering, 2022, 35(1): 123−142. (in Chinese)

[25] XIE G Q, XIAO X R, LI R F, et al. Schedule length minimization of parallel applications with energy consumption constraints using heuristics on heterogeneous distributed systems. Concurrency and Computation: Practice and Experience, 2017, 29(16): e4024.

[26] ZHANG L X, LI K L, LI C Y, et al. Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Information Sciences, 2017, 379: 241–256.

[27] FAN X M, LI X H, DING Y M, et al. Demand response scheduling algorithm for smart residential communities considering heterogeneous energy consumption. Energy and Buildings, 2023, 279: 112691.

[28] ZHOU S E, LEI H. An improved genetic-annealing algorithm for task scheduling based on precedence task duplication. Microelectronics & Computer, 2006, 23(10): 62–64.

[29] CHENG X H, LIU T C. Heterogeneous multi-core task scheduling based on crazy adaptive bottle ascidians group algorithm. Computer & Digital Engineering, 2024, 52(10): 2886–2889, 2919. (in Chinese)

[30] FENG S, JIANG B, XU H. Task Scheduling for heterogeneous multi-core processors based on evolutionary adaptive bat algorithm. Computer Engineering, 2025, 51(5): 249–256. (in Chinese)

[31] YAN T, LI W, LI Y C. QoS guided task scheduling heuristic in computational grid environments. Microelectronics & Computer, 2006, 23(10): 107–110. (in Chinese)