Attribute-Based Multi-Filter Bloom Tabu Search for Evaluation-Limited Flexible Flowshop Scheduling

Levente Áron Fazekas and Károly Nehéz

Institute of Information Technology, University of Miskolc, Miskolc, Hungary
levente.fazekas@uni-miskolc.hu, karoly.nehez@uni-miskolc.hu

Abstract. Tabu search relies on memory to prevent cycling and encourage the exploration of new regions of the search space. In many implementations, however, tabu memory is treated mainly as an implementation detail. Classical tabu lists store complete solutions or moves and identify only exact repetitions. For large combinatorial problems, this may require substantial memory and comparison effort, while still failing to recognize different encodings that produce similar schedules after decoding.

We propose an alternative tabu-memory scheme that combines exact solution information with domain-specific attributes. These attributes may describe permutation fragments, assignment patterns, temporal relations, or aggregated features derived from processing times, setup times, transfer times, and due dates. Each attribute type is stored in a separate Bloom filter, forming a compact multi-filter structure with bounded memory use and constant-time membership queries.

The proposed method extends tabu memory beyond exact repetition detection. A candidate may be rejected before simulation when several of its attributes match patterns observed in previously visited solutions. The degree of generalization can be controlled through the selected attributes, quantization rules, hashing policies, and the threshold used to combine filter matches.

We evaluate the method on permutation-based scheduling problems with complex constraints and simulation-based objective evaluation. The results show that attribute-based Bloom filters reduce the number of costly evaluations in every reported case, while maintaining competitive solution quality in several settings. These findings support treating tabu memory as a configurable search policy based on solution attributes, rather than only as a cache of previously visited solutions.

Keywords: tabu search, Bloom filter, Attribute-based tabu, Multi-filter tabu, flexible flowshop scheduling, discrete-event simulation

1 Introduction

This section motivates why tabu memory design matters when flexible flowshop optimization is dominated by expensive schedule evaluation rather than by neighborhood generation.

© The Author(s) 2026
S. Árvai-Homolya et al. (eds.), *Proceedings of the 2nd Innovation in Engineering Education Conference (IEE 2026)*, Atlantis Highlights in Social Sciences, Education and Humanities 58,
https://doi.org/10.2991/978-94-6239-823-8_8

Flexible flowshop scheduling appears in manufacturing settings where jobs traverse multiple stages with parallel machines, due dates, setup effects, transport delays, and machine eligibility constraints [5, 12, 14, 13]. In the setting considered here, a permutation is only an input to a discrete-event simulator: the simulator constructs the detailed schedule, resolves stage interactions, and returns objective values such as makespan and tardiness [6, 7]. This makes the search evaluation-limited.

Classical tabu search addresses cycling through explicit memory over previously visited solutions [9]. That mechanism is useful when exact revisits are common and comparison cost is modest. In simulation-driven scheduling, however, many permutations are syntactically different while remaining operationally similar after decoding. Exact tabu memory then filters too narrowly: it grows with search history, it still admits many near-duplicates, and it spends memory on representation detail that does not necessarily matter for the simulator.

Bloom filters change the engineering trade-off by replacing explicit storage with constant-time probabilistic membership tests [3, 1]. This enables a fixed-size rejection layer with no false negatives and a controllable False positive rate. More importantly, the same mechanism can be lifted from full permutations to Attribute signatures. Instead of asking whether a candidate has been seen exactly, the search can ask whether it carries structural features that have already been explored.

The contribution of the paper is therefore threefold. First, it separates Classical tabu, Exact Bloom tabu, and Attribute-based tabu at the level of policy versus mechanism. Second, it documents the Attribute signature families exposed by the current implementation and explains how they approximate simulation-relevant equivalence classes. Third, it reports descriptive benchmark evidence that the Attribute-based policy reduces costly evaluations under the same search budget. This implies that tabu memory should be analyzed as part of the evaluation policy, not only as an implementation detail.

1.1 Related Work

Tabu search commonly defines memory over moves or solution attributes rather than complete solutions, allowing related regions of the search space to be excluded without storing every visited solution [10, 2]. Structure-aware tabu memories have been particularly successful in scheduling, where operation order and critical schedule features provide more useful information than complete solution identifiers [11]. Related candidate-restriction methods, such as granular tabu search, also reduce the number of solutions requiring complete evaluation by excluding structurally unpromising neighbours [16]. Bloom filters provide fixed-memory approximate membership tests and have been widely studied as compact alternatives to exact sets [4]. In simulation optimization, reducing the number of calls to an expensive evaluator is itself an important objective [8]. The proposed method combines these ideas by storing multiple scheduling-related attribute signatures in Bloom filters and applying a thresholded rejection rule before discrete-event evaluation.

2 Method

This section defines the search procedure in the same order as the implementation: representation, neighborhood, tabu decision, acceptance, evaluation, and stopping.

2.1 Representation and Neighborhood

We represent a candidate by a permutation $\pi = (\pi_1, \ldots, \pi_n)$ over the jobs. The permutation is not yet a full schedule. Instead, a discrete-event decoder $\mathcal{D}(\pi)$ simulates dispatching, machine assignment, waiting, setup, and transfer effects to obtain an executable schedule and objective vector

$$f(\pi) = \left(C_{\max}, F_{\max}, \sum T_j, T_{\max}, \sum U_j\right). \tag{1}$$

The important point is that multiple permutations can decode to very similar trajectories when the simulator resolves them through the same bottlenecks, setup families, or slack structure.

The local search neighborhood is sampled through adjacent index swaps, which matches the default move generator in the underlying implementation. Adjacent swaps are attractive here because they are cheap to generate and because they create many "almost identical" candidates, making them a useful stress case for memory mechanisms. A move therefore changes the representation minimally while it can leave the decoded schedule nearly unchanged.

This representation-neighborhood pair creates precisely the mismatch that motivates Attribute-based tabu: the search moves in permutation space, but the expensive work happens only after decoding. This implies that useful tabu memory should react to decoded structure, not only to exact permutations.

2.2 Tabu Decision

Classical tabu stores exact solution identifiers and rejects a candidate if its identifier is already present. Exact Bloom tabu keeps the same policy while replacing explicit storage by a Bloom filter bit array of size m queried through k hash probes. If n keys have been inserted, the usual approximation for the False positive probability is

$$p_{\text{fp}} \approx \left(1 - e^{-kn/m}\right)^k. \tag{2}$$

There are no false negatives: if a key has been inserted, the filter will report membership. A False positive rejects a new candidate prematurely, which is a controlled form of search bias rather than a memory error.

Attribute-based tabu extends this idea by combining one exact filter with q attribute filters. Let $h_0(\pi)$ denote the exact permutation hash, B_0 the exact filter. For $i = 1, \ldots, q$, let $\phi_i(\pi)$ denote an Attribute signature extracted from the same candidate π and a B_i filter. The Multi-filter tabu predicate is

$$\text{Tabu}(\pi) = B_0(h_0(\pi)) \ \vee \ \left[\sum_{i=1}^{q} \mathbf{1}\{B_i(\phi_i(\pi))\} \geq \tau\right], \tag{3}$$

where τ is an attribute threshold. The exact Bloom layer therefore acts as a probabilistic replacement of Classical tabu, while the attribute layer introduces a semantic similarity test.

Under an independence approximation, if a single attribute filter has False positive probability p_a, then the attribute part produces

$$p_{\text{attr}} \approx \sum_{j=\tau}^{q} \binom{q}{j} p_a^j (1-p_a)^{q-j}. \tag{4}$$

The independence assumption is only approximate because many signatures are correlated, but the formula still shows the role of τ: a larger threshold reduces accidental matches, while a smaller threshold broadens the filtered region. This suggests that the Multi-filter tabu layer is not a single memory but a thresholded committee of weak similarity tests.

2.3 Acceptance, Evaluation, and Stopping

Given a sampled neighbor π', the search first applies the tabu predicate. If the candidate is tabu, it is rejected immediately unless aspiration is enabled and the candidate improves the incumbent. This aspiration override is supported by the search framework even when the reported benchmark configuration keeps the acceptance rule conservative.

If the candidate is admissible, it is evaluated by the decoder and compared to the current solution under the chosen objective ranking. Accepted solutions are inserted into the exact Bloom filter and into every attribute filter. The search stops when it reaches the iteration limit or the Evaluation budget.

In its tested form, Exact Bloom tabu and each component of the Multi-filter tabu memory are instantiated as fixed-size Bloom arrays with three hash probes. The exact and attribute hashers, the Attribute signature set, and the threshold τ remain configurable. This means that the search loop stays unchanged while the memory policy can still be exchanged.

The resulting control flow makes the rejection step cheaper than the evaluation step by design. This implies that any attribute family that catches redundant candidates before simulation can increase effective search depth under the same Evaluation budget.

2.4 Policy-Based Design

This section frames the proposed search as a composition of policies rather than as a monolithic algorithm.

A metaheuristic can be described as a tuple of interchangeable policies for representation, neighborhood generation, tabu memory, acceptance, and evaluation. In the present search, the permutation representation, adjacent-swap

neighborhood, aspiration-aware acceptance rule, and simulation-based evaluator remain fixed while the memory policy changes. Classical tabu and Exact Bloom tabu share the same exact-revisit policy and differ only in storage mechanism: explicit identifiers versus probabilistic storage. Attribute-based tabu changes the policy itself by filtering candidates through a thresholded set of Attribute signatures.

This distinction matters in simulation-driven optimization because the evaluation stage dominates the budget. When mechanism and policy are separated, one can change the memory policy without changing the evaluator, or change the evaluator without redesigning the memory interface. This suggests that policy-based design is a useful way to extend metaheuristics under tight Evaluation budgets.

3 Attribute Signatures in the Implementation

This section explains the attribute-extraction methods implemented in the companion scheduling codebase and relates each one to the type of equivalence it is meant to approximate.

The code separates two sources of information. The first source is permutation structure: local adjacencies, coarse positions, and partial order relations. The second source is a term storage derived from the discrete-event model, currently including release times, due dates, total processing times, total transfer times, slack hints, and per-stage processing and transfer vectors. Quantization is used whenever exact numeric equality would be too strict. For a term value v and quantum q, the quantized value is

$$\tilde{v} = \left\lfloor \frac{v}{q} \right\rfloor q \tag{5}$$

for non-negative values, with the analogous symmetric rule for negative values.

The permutation-structure signatures are useful when structural similarity should be detected without consulting the simulator. `adjacent_pairs` is the strictest variant: it reacts strongly to local edge changes and is therefore useful when adjacency itself is meaningful. `perm_pattern` is less literal: it slides a small window over the permutation and hashes the induced pairwise order pattern, so two permutations can match even when the job ids differ as long as the local ordering pattern is preserved. `inversion_sample` goes further by sampling precedence relations across the whole permutation. This reduces cost and makes the signature react to global order while tolerating some local noise.

The term-driven signatures aim at simulator-relevant equivalence classes. `job_proc_seq` hashes the quantized sequence of total processing times, which means that swapping jobs within the same processing bucket is invisible to that filter. `job_transfer_hist` is even more tolerant because it forgets order and remembers only the distribution of transfer burdens. `edge_signature` is the most expressive of the default term-driven signatures: for every consecutive pair it

Signature	Inputs	What stays stable	Main effect
`adjacent_pairs`	consecutive job ids	edits away from the touched edge	captures exact local adjacency patterns
`inversion_sample`	sampled precedence relations	unrelated local swaps	captures coarse global order without hashing all pairs
`position_buckets`	job ids and coarse positions	small moves inside the same bucket	captures front/middle/tail placement instead of exact index
`perm_pattern`	sliding windows over the permutation	edits outside the active window	captures local order motifs through pairwise comparison bits
`job_proc_seq`	quantized `proc_sum` in permutation order	swaps between jobs in the same processing bucket	captures long/short job patterns seen by the decoder
`job_transfer_hist`	histogram of quantized `transfer_sum`	any reordering that preserves bin counts	captures aggregate transport burden rather than exact order
`edge_signature`	consecutive pairs, quantized terms, and term deltas	swaps between jobs with similar term buckets	captures transition patterns that can affect setups, transfers, and bottleneck pressure

Table 1. Attribute signatures exposed by the current registry. The implementation also stores stage-wise terms, which makes additional domain-specific signatures possible without changing the Multi-filter tabu logic.

mixes the current term values and their deltas, optionally with position information. This makes it sensitive to how "heavy" jobs follow one another, which is often closer to the simulator's behavior than exact job identity.

Two design consequences follow from Table 1. First, some filters are intentionally redundant: a thresholded committee is often more robust than any single signature. Second, quantization and threshold choice are part of the model, not just low-level tuning, because they determine which permutations collapse into the same Attribute signature. This implies that Attribute-based tabu is best understood as feature engineering for tabu memory.

4 Examples of Attribute Design

This section gives domain examples that clarify how Attribute signatures can merge syntactically different candidates that are operationally equivalent or nearly equivalent.

4.1 Manufacturing Scheduling Example

Consider a flexible flowshop bottleneck stage with four jobs A, B, C, D and quantized term buckets

$$(\tilde{p}, \tilde{t})_A = (10, 2),\ (\tilde{p}, \tilde{t})_B = (10, 2),\ (\tilde{p}, \tilde{t})_C = (6, 1),\ (\tilde{p}, \tilde{t})_D = (6, 1),$$

where $\tilde{p}$ is a processing-time bucket and $\tilde{t}$ is a transfer-time bucket. The permutations

$$\pi_1 = (A, B, C, D), \qquad \pi_2 = (B, A, C, D)$$

are distinct for Classical tabu and for Exact Bloom tabu because the job ids differ in the first two positions. Yet `job_proc_seq` sees the same quantized sequence $(10, 10, 6, 6)$ for both, and `job_transfer_hist` sees the same histogram. If A and B also have similar release dates, due dates, and stage-level processing vectors, then the decoder can create almost the same queues and completion times from both permutations.

This is the manufacturing interpretation of Attribute-based tabu: the search does not claim that π_1 and π_2 are equal, only that they are similar enough to compete for the same part of the Evaluation budget. This reduces simulator calls when the key driver is stage load pattern rather than exact job identity.

4.2 TSP Loop Example

The same idea becomes even clearer on the traveling salesman problem. In a symmetric TSP, the tours

$$\pi_1 = (1, 2, 3, 4, 5), \qquad \pi_2 = (3, 4, 5, 1, 2), \qquad \pi_3 = (1, 5, 4, 3, 2)$$

describe the same Hamiltonian loop: π_2 is only a cyclic shift of the starting city, and π_3 traverses the same loop in reverse. Exact permutation memory treats these tours as different because the sequence representation changes. A TSP-specific Attribute signature built from the undirected edge set

$$\big\{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 1\}\big\}$$

would map all three tours to the same class.

This example also shows that attribute design is domain-specific. The current implementation contains path-oriented permutation signatures, not a dedicated cyclic edge-set extractor. For TSP use, the attribute layer would therefore need an edge-normalized signature with wrap-around and orientation invariance. The implication is that Attribute-based tabu is general, but the correct Attribute signature depends on the symmetry of the underlying solution space.

The same framework can also encode setup families, machine groups, stage-wise load histograms, due-date slack classes, and assignment patterns between

jobs and resource groups. These remain natural manufacturing attributes because they survive many harmless representation changes while still controlling queueing and tardiness. This suggests that the main modeling task is to choose attributes that collapse irrelevant variation without hiding meaningful alternatives.

5 Experimental Design

This section restates the experimental goal and describes the benchmark evidence available in the current paper directory.

The goal of the experiment is to compare three tabu-memory variants under the same search setting: Classical tabu implemented by an exact set-based memory (`set`), Exact Bloom tabu (`bloom_exact`), and Attribute-based tabu implemented as a hybrid Multi-filter tabu memory (`multibloom`). The summary file contributes one aggregated observation per method and matched benchmark setting, yielding eight matched comparisons.

The benchmark cases use Taillard-derived permutation scheduling instances [15] mapped to flexible flowshop scenarios. The four reported case families are: `standard_m2_none`, a standard two-machine-per-stage layout without correlated extras; `standard_m2_corr`, the same layout with correlated due, transfer, and setup perturbations; `bottleneck_s1_none`, a stage-1 bottleneck without correlated extras; and `bottleneck_s1_corr`, the bottleneck layout with correlated extras. The reported problem sizes are $n \in \{20, 50\}$ and $m = 5$ stages.

The outcome variables are makespan, max flow time, sum tardiness, max tardiness, number of tardy jobs, and evaluation count. Because the file is an aggregated benchmark summary rather than a full event log, it does not contain runtime measurements, repeated runs for each matched case, or a full per-row record of the Attribute configuration used in the Multi-filter tabu memory. Consequently, the paper interprets the results descriptively rather than inferentially and uses evaluation count as the primary efficiency measure. The setup is sufficient to compare the dominant behavioral trend among the three memory policies, but not to support a full ablation of filter size, threshold, quantization, or extractor subsets. This implies that the results should be read as evidence about the memory design principle rather than as a final calibration of every Multi-filter tabu parameter.

6 Results

This section reports the main empirical trend, then explains the mechanism behind the observed trade-off.

The comparison separates implementation replacement from policy change. Exact Bloom tabu remains close to Classical tabu. Attribute-based tabu, in contrast, consistently reduces the number of expensive evaluations. Across the eight matched settings, Exact Bloom tabu reproduces the Classical tabu outcomes exactly in seven cases and differs only once. The single deviation appears in

n Case	Classical	Exact Bloom	Attribute-based	Reduction (%)
20 standard / no corr.	14866	14866	12745	14.3
20 standard / corr.	14684	14684	11472	21.9
20 bottleneck / no corr.	15089	15089	11850	21.5
20 bottleneck / corr.	14495	14361	11484	20.8
50 standard / no corr.	15661	15661	11438	27.0
50 standard / corr.	15664	15664	12610	19.5
50 bottleneck / no corr.	15671	15671	10995	29.8
50 bottleneck / corr.	15581	15581	12822	17.7

Table 2. Evaluation counts. Reduction is computed relative to Classical tabu.

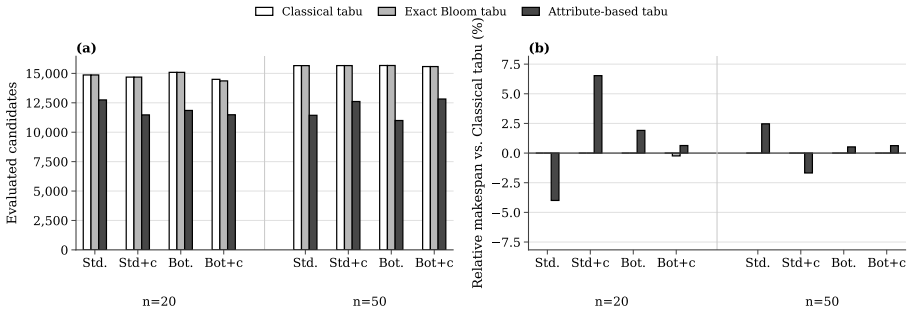


Fig. 1. Benchmark summary plots. In both panels, the left block corresponds to $n = 20$ and the right block to $n = 50$; within each block, the cases are standard, standard correlated, bottleneck, and bottleneck correlated. Panel (a) shows that Attribute-based tabu lowers evaluation count in every matched case, while Exact Bloom tabu stays almost indistinguishable from Classical tabu. Panel (b) shows that makespan changes are mixed and smaller in magnitude than the evaluation-count reduction.

the $n = 20$ bottleneck-correlated case, where the Bloom variant evaluates 0.9% fewer candidates and reaches a 0.24% lower makespan. This suggests that the probabilistic mechanism largely preserves the classical policy at the reported scale.

The size effect is also clear. For $n = 20$, the mean reduction is 19.6%; for $n = 50$, it rises to 23.5%. The non-correlated cases tend to show stronger evaluation savings than the correlated cases (23.1% versus 20.0% on average). The strongest reduction appears in the bottleneck / no-correlation family, which averages 25.7% and peaks at 29.8% for $n = 50$. Figure 1 shows the same pattern graphically. This suggests that the Multi-filter tabu layer is most effective when coarse structural signatures align well with the queueing structure seen by the simulator.

The quality trend is more mixed, which is expected for Attribute-based tabu. Table 3 reports the relative change of the Attribute-based tabu method against the Classical tabu baseline. Attribute-based tabu improves makespan in two

n Case	$\Delta C_{\max}$	$\Delta \sum T$	$\Delta T_{\max}$	$\Delta \sum U$
20 standard / no corr.	−4.0%	n/a	n/a	+0
20 standard / corr.	+6.5%	+13.8%	+7.7%	+0
20 bottleneck / no corr.	+1.9%	n/a	n/a	+0
20 bottleneck / corr.	+0.6%	+0.5%	+2.1%	+0
50 standard / no corr.	+2.5%	+83.3%	+31.1%	+2
50 standard / corr.	−1.7%	−3.7%	−2.0%	+0
50 bottleneck / no corr.	+0.5%	+25.7%	+0.9%	+4
50 bottleneck / corr.	+0.6%	+1.2%	+0.6%	+0

Table 3. Relative change of Attribute-based tabu against Classical tabu. Entries marked *n/a* correspond to zero baseline tardiness values. The table makes the mixed quality effect explicit rather than collapsing it into a single average.

cases and worsens it in six, with a mean makespan change of +0.9%. The trade-off is therefore real and should not be hidden by the average evaluation savings. Two cases are especially informative. For $n = 20$ in the standard / no-correlation family, Attribute-based tabu reduces evaluation count by 14.3% and improves makespan by 4.0%. For $n = 50$ in the standard / correlated family, it reduces evaluations by 19.5% while improving every reported time-based objective. At the other end of the spectrum, the $n = 50$ standard / no-correlation case reduces evaluations by 27.0% but increases sum tardiness markedly. This is consistent with over-filtering or with an Attribute signature set that is too coarse for that instance family.

The mechanism behind this trade-off is consistent with the method design. When the chosen Attribute signatures align with simulator-relevant equivalence classes, the search spends less effort on redundant candidates and can still reach competitive or better solutions. When the signatures are too coarse, the attribute layer can reject candidates that would have improved the incumbent, which appears as over-filtering. The correlated cases tend to show slightly smaller evaluation savings and larger average makespan movement than the non-correlated cases, which is consistent with more complex interactions between due dates, transfer times, and setup effects. This suggests that Attribute-based tabu is most valuable when evaluation cost dominates and when Attribute design matches the true structure of the decoded schedules.

7 Threats to Validity and Discussion

The reported evidence is limited to four benchmark families and two problem sizes. Although the evaluation-count reduction is consistent across the matched cases, the results do not establish uniform behaviour across all flexible flowshop configurations, objective functions, or constraint combinations. The results are also influenced by attribute-design bias: the implemented signatures reflect the information exposed by the current scheduling model, and different applications may require different notions of structural similarity.

The behaviour of Attribute-based tabu depends on the granularity of the selected signatures and on the Multi-filter decision parameters. An excessively permissive configuration may cause over-filtering by rejecting candidates that are sufficiently different to produce better schedules. For example, setting $\tau = 0$ would make every candidate tabu, while a very small positive threshold may allow a single weak or accidental match to trigger rejection. Conversely, setting τ close to q makes the attribute layer increasingly conservative and reduces the number of structurally similar candidates it can detect. Extractor parameters introduce a similar trade-off. Very short sliding windows produce common local patterns shared by many unrelated permutations, whereas very long windows approach exact permutation matching and provide little useful generalization. Coarse quantization may merge jobs with meaningfully different processing times, transfer requirements, or due-date properties, while excessively fine quantization rarely detects operationally similar candidates.

Bloom-filter capacity and extractor dependence introduce further sources of over-filtering. Small or saturated filters increase the probability of accidental rejection, while several strongly correlated extractors may count the same underlying structural similarity multiple times. These effects can be controlled by increasing τ, refining signature or quantization granularity, enlarging or resetting the filters, removing redundant extractors, and applying an aspiration criterion. The increased tardiness observed in the $n = 50$ standard case is consistent with a configuration that filtered candidates too broadly for the tardiness objectives.

Attribute-based tabu is therefore most suitable when schedule evaluation is expensive, the neighbourhood generates many candidates that decode into similar schedules, and the selected attributes correspond to the structures that dominate the simulated objective values. The benchmark summary contains evaluation counts rather than wall-clock measurements or repeated-run statistics. Evaluation count is the primary outcome because the method specifically targets calls to the discrete-event evaluator; no end-to-end runtime speedup is claimed. Repeated experiments, parameter-sensitivity analysis, and extractor-level ablations remain necessary to determine how the method should be calibrated for other problem families.

8 Conclusions

This section summarizes the main conclusion of the paper.

Exact Bloom tabu is a practical replacement for Classical tabu when the goal is to keep the same memory policy with fixed memory usage and constant-time queries. Attribute-based tabu changes the policy itself by turning tabu memory into a thresholded similarity filter over multiple Attribute signatures. On the reported flexible flowshop benchmarks, that change reduces the number of expensive evaluations by 21.5% on average while keeping solution quality competitive in several cases and degrading it in others.

The central lesson is therefore not that one attribute set is universally best, but that tabu memory can be designed around structural redundancy rather

than exact repetition. When schedule evaluation is the dominant cost, filtering redundant candidates before simulation can be more valuable than remembering exact permutations after the fact. This suggests that future tabu-search design should treat memory as a policy component with interchangeable mechanisms, not only as a data structure.

References

1. Almeida, P.S., Baquero, C., Preguiça, N., Hutchison, D.: Scalable bloom filters. Information Processing Letters **101**(6), 255–261 (2007). DOI https://doi.org/10.1016/j.ipl.2006.10.007. URL `https://www.sciencedirect.com/science/article/pii/S0020019006003127`
2. Battiti, R., Tecchiolli, G.: The reactive tabu search. ORSA Journal on Computing **6**(2), 126–140 (1994). DOI 10.1287/ijoc.6.2.126
3. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970). DOI 10.1145/362686.362692. URL `https://doi.org/10.1145/362686.362692`
4. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. Internet mathematics **1**(4), 485–509 (2004). DOI 10.1080/15427951.2004.10129096
5. Brucker, P.: Scheduling algorithms, 4 edn. Springer Berlin, Heidelberg (1999). DOI https://doi.org/10.1007/978-3-540-24804-0
6. Fazekas, L., Kulcsar, G.: Data structures and simulation algorithms for flow-shop scheduling with resource availability constraints. Production Systems and Information Engineering **10**(1), 29–47 (2022). DOI 10.32968/psaie.2022.1.3
7. Fazekas, L.A., Nehez, K.: Multi-objective genetic and memetic algorithms in flexible flowshop scheduling. In: Annales Mathematicae et Informaticae, vol. 61, pp. 94–107 (2025). DOI 10.33039/ami.2025.10.011
8. Fu, M.C.: Optimization for simulation: Theory vs. practice. INFORMS Journal on Computing **14**(3), 192–215 (2002). DOI 10.1287/ijoc.14.3.192.113
9. Glover, F.: Tabu search: A tutorial. Interfaces **20**(4), 74–94 (1990). DOI https://doi.org/10.1287/inte.20.4.74
10. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1997). DOI 10.1007/978-1-4615-6089-0
11. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management Science **42**(6), 797–813 (1996). DOI 10.1287/mnsc.42.6.797
12. Pinedo, M.L.: Scheduling, 6 edn. Springer Cham (2012). DOI https://doi.org/10.1007/978-3-031-05921-6
13. Ribas, I., Leisten, R., nan, J.M.F.: Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research **37**(8), 1439–1454 (2010). DOI https://doi.org/10.1016/j.cor.2009.11.001. URL `https://www.sciencedirect.com/science/article/pii/S0305054809002883`. Operations Research and Data Mining in Biological Systems
14. Ruiz, R., Vázquez-Rodríguez, J.A.: The hybrid flow shop scheduling problem. European Journal of Operational Research **205**(1), 1–18 (2010). DOI https://doi.org/10.1016/j.ejor.2009.09.024. URL `https://www.sciencedirect.com/science/article/pii/S0377221709006390`

15. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993). DOI https://doi.org/10.1016/0377-2217(93)90182-M. URL `https://www.sciencedirect.com/science/article/pii/037722179390182M`. Project Management anf Scheduling
16. Toth, P., Vigo, D.: The granular tabu search and its application to the vehicle-routing problem. INFORMS Journal on Computing **15**(4), 333–346 (2003). DOI 10.1287/ijoc.15.4.333.24890

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (http://creativecommons.org/licenses/by-nc/4.0/), which permits any noncommercial use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

