Motzkin Neighborhood Evaluation in Tabu Search and Simulated Annealing for the Permutational Flow Shop Problem

Remigiusz Wojewódzki[1] and Wojciech Bożejko[2]

[1] Wrocław University of Science and Technology, Poland
remigiusz.wojewodzki@gmail.com
[2] Wrocław University of Science and Technology, Poland
wojciech.bozejko@pwr.edu.pl

Abstract. In this paper, we investigate the impact of using the *Motzkin* neighborhood in the permutation flow shop problem (PFSP) for two metaheuristic scheduling algorithms: Tabu Search and Simulated Annealing. The goal is to minimize $C_{\max}$, understood as the completion time of all jobs. The Motzkin neighborhood enables performing multiple swaps of permutation elements during a single iteration, i.e., simultaneous swaps of multiple permutation elements with allowed nesting, while prohibiting crossings of the lines connecting the elements. The use of dynamic programming reduces the computational complexity and speeds up the algorithm. In this paper, we compare the *Motzkin* neighborhood with the *Fibonacci* neighborhood and the classical *Dynasearch* and *Adjacent* neighborhoods. The experiments were conducted on Taillard benchmark instances for many different time limits. The obtained results allow us to conclude that the choice of neighborhood has a significant impact on the quality of the result obtained by the algorithm within the given time. For all investigated time limits, *Dynasearch* achieved the best results (the smallest average gap), while the *Fibonacci* neighborhood is at the other end of the ranking. The *Motzkin* neighborhood examined in detail in this work is, in most cases, ranked better than *Adjacent*. This advantage becomes clearly visible for larger allowed computation times. The obtained results suggest that neighborhoods using multi-swap moves constitute an interesting alternative to their classical (simpler) counterparts, noting that further computational optimization of complex neighborhoods is required.

Keywords: Flow shop scheduling · Metaheuristics · Tabu search · Simulated annealing · Composite neighborhood · Motzkin

1 Introduction

The Permutation Flow Shop Problem (PFSP), mathematically defined in [**?**], consists of finding a permutation of n jobs processed on m machines in the same order. The objective is to minimize the completion time of all jobs (makespan)

$C_{\max}$. The problem is known to be NP-hard for $m \geq 3$, which motivates the use of metaheuristic methods. In metaheuristic algorithms, the neighborhood of a solution x is defined as the set of solutions that can be obtained through admissible modifications of x [**?**].

In addition to the choice of the general algorithm, the selection of the neighborhood structure is equally important. In the classical approach, neighborhood elements perform simple moves, which results in the solution space being explored locally and relatively shallowly. In contrast, composite neighborhoods allow for complex moves within a single iteration, potentially leading to faster convergence and better solution quality. However, this complexity comes at a significant cost in terms of the time required for a single iteration.

This work extends the study reported in [**?**], where a new type of neighborhood based on Motzkin numbers, called the *Motzkin* neighborhood, was proposed. This neighborhood enables a complex compound move within a single iteration, allowing swaps of any two elements of the permutation as well as swaps corresponding to pairs of indices nested with respect to one another (i.e., nested arcs). Intersections of the corresponding arcs are forbidden.

In the present paper, the focus is placed on an experimental investigation of the behavior and the impact of the neighborhood on the solutions generated by metaheuristic algorithms. For this purpose, we performed experiments for various input sizes and time limits, which allowed us to examine the behavior of the *Motzkin* neighborhood under different conditions. Additionally, a comparative analysis with other neighborhoods—including the classical *Adjacent* swap, the *Dynasearch* neighborhood, and the *Fibonacci* neighborhood—was performed in order to identify criteria for selecting an appropriate neighborhood for the problem under study.

2 Problem Definition

Permutation Flow Shop Problem. Given a matrix of processing times $P = [p_{k,j}]_{m\times n}$, where $k = 1, \ldots, m$ denotes a machine and $j = 1, \ldots, n$ denotes a job, while a permutation π determines the jobs performed in the same order on all machines.

Properties of permutations and the relationships between them can be analyzed using positive definite functions on permutation groups. This provides theoretical foundations for studying dependencies between job sequences [**?**].

Let $C_{k,j}$ denote the completion time of job j executed in the order π on machine k. The following relationship therefore holds:

$$C_{1,j} = C_{1,j-1} + p_{1,\pi(j)}, \quad C_{k,1} = C_{k-1,1} + p_{k,\pi(1)}, \tag{1}$$

$$C_{k,j} = \max\{C_{k,j-1},\, C_{k-1,j}\} + p_{k,\pi(j)}, \quad \text{for } k = 1, \ldots, m,\ j = 1, \ldots, n. \tag{2}$$

The value of the objective function equals $C_{\max} = C_{m,n}$, i.e., the completion time of the last job on the last machine. Hence, the PFSP reduces to finding a permutation π that minimizes $C_{\max}$.

Neighborhood structures. Following [?], a *neighborhood* of a solution x is the set of solutions obtained by an admissible modification of x. It determines how the solution space is explored and exploited during the search. The choice of the neighborhood structure determines the search pattern in the solution space, balancing exploration and exploitation for a given space.

Characteristics of the Motzkin numbers. The Motzkin numbers M_n, defined in [?], count *planar*, i.e., non-crossing, configurations in a set of n ordered points. Each point may be isolated, be the beginning of an arc, or its end, under the restriction that arcs cannot intersect. In the presented approach, these numbers constitute, in a sense, a generalization of the Catalan numbers, with the reservation that the former allow isolated points.

3 Motzkin Neighborhood Structures

The neighborhood proposed in [?] constitutes a strictly structured neighborhood, in which each swap of two elements is represented by a pair of their corresponding indices (i, j). A set of such pairs is admissible only if the corresponding arcs (created above the index axis) do not intersect and do not share endpoints. Nesting of arcs and elements that are not part of any arc are admissible. In comparison to classical neighborhoods, such an approach allows searching a larger region of solutions in a single iteration.

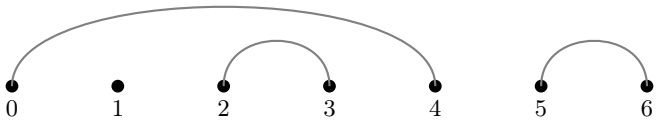


Fig. 1: Example of an admissible composite move: arcs $(0, 4)$, $(2, 3)$, $(5, 6)$.

3.1 Formal definition

Swapping elements. Given a permutation $\pi = (\pi_0, \pi_1, \ldots, \pi_{n-1})$ and a pair of indices (i, j) satisfying $0 \leq i < j < n$, we define the *end swap* as the operation:

$$S_{i,j}(\pi) = (\pi_0, \ldots, \pi_{i-1},\ \pi_j,\ \pi_{i+1}, \ldots, \pi_{j-1},\ \pi_i,\ \pi_{j+1}, \ldots, \pi_{n-1}). \tag{3}$$

This operation swaps the elements indexed by i and j, leaving the intermediate elements $\pi_{i+1}, \ldots, \pi_{j-1}$ unchanged. Locally, the segment $[\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j]$ is transformed into $[\pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i]$.

Composite move. We define it as a set of index pairs $\mathcal{M} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)\}$. The set $\mathcal{M}$ is *admissible* if it satisfies the following conditions:

1. **Index disjointness**: For any two index pairs $(i_a, j_a), (i_b, j_b) \in \mathcal{M}$ we have $\{i_a, j_a\} \cap \{i_b, j_b\} = \varnothing$ (no shared indices).
2. **No crossings**: The pattern $i_a < i_b < j_a < j_b$ is forbidden for any $(i_a, j_a), (i_b, j_b) \in \mathcal{M}$ (arcs cannot intersect).
3. **Nesting allowed**: The configuration $i_a < i_b < j_b < j_a$ is admissible (the inner pair is entirely contained within the outer pair).

3.2 Algorithm for constructing a new permutation

The algorithm determines the set of pairs $\mathcal{M}$ within a single iteration. The goal is to construct a permutation π that leads to makespan minimization.

1. Determine the prefix completion columns $F[k][r]$.
2. Compute $\Delta_{i,j}$ for all pairs (i, j).
3. Fill the table $\mathrm{dp}[L][R]$.
4. Reconstruct the selected set of pairs $\mathcal{M}$.
5. Apply the determined swaps to the permutation π.

4 Full Enumeration of Motzkin Composite Moves

In the further part of the chapter we determine the number of admissible composite moves for a given n, and then we compare it with the Motzkin numbers M_n corresponding to the same value of n.

Table 1: The first Motzkin numbers ($n = 0..10$).

n	0	1	2	3	4	5	6	7	8	9	10
M_n	1	1	2	4	9	21	51	127	323	835	2188

Enumeration of Composite Moves for $n = 0$. There are no indices. Only the empty configuration exists.

1. $\mathcal{M}_1 = \varnothing$.

Verification: $1 = M_0$. ✓

Enumeration of Composite Moves for $n = 1$. There is only one index (index 0), so the only admissible configuration is the empty set.

1. $\mathcal{M}_1 = \varnothing$. (index 0 isolated)

Verification: $1 = M_1$. ✓

Enumeration of Composite Moves for $n = 2$. We have two indices: 0 and 1.
Category I: Empty Composite Move:

1. $\mathcal{M}_1 = \varnothing$. (both indices isolated)

Category II: One Pair:

2. $\mathcal{M}_2 = \{(0, 1)\}$.

Verification: $1 + 1 = 2 = M_2$. ✓

Enumeration of Composite Moves for $n = 3$. We have three indices: 0, 1, 2.
Category I: Empty Composite Move

1. $\mathcal{M}_1 = \varnothing$.

Category II: One Pair:

2. $\mathcal{M}_2 = \{(0, 1)\}$, 3. $\mathcal{M}_3 = \{(0, 2)\}$, 4. $\mathcal{M}_4 = \{(1, 2)\}$.

Verification: $1 + 3 = 4 = M_3$. ✓ Note: For $n = 3$ it is not possible to form two disjoint pairs (this requires at least 4 indices).

Enumeration of Composite Moves for $n = 4$. We have four indices: 0, 1, 2, 3.
Category I: Empty Composite Move:

1. $\mathcal{M}_1 = \varnothing$.

Category II: One Pair:

2. $\mathcal{M}_2 = \{(0, 1)\}$, 4. $\mathcal{M}_4 = \{(0, 3)\}$, 6. $\mathcal{M}_6 = \{(1, 3)\}$,
3. $\mathcal{M}_3 = \{(0, 2)\}$, 5. $\mathcal{M}_5 = \{(1, 2)\}$, 7. $\mathcal{M}_7 = \{(2, 3)\}$.

Category III: Two Disjoint Pairs:

8. $\mathcal{M}_8 = \{(0, 1), (2, 3)\}$.

Category IV: Two Nested Pairs:

9. $\mathcal{M}_9 = \{(0, 3), (1, 2)\}$.

Verification: $1 + 6 + 1 + 1 = 9 = M_4$. ✓

Enumeration of Composite Moves for $n = 5$. We have five indices: 0, 1, 2, 3, 4.
Category I: Empty Composite Move:

1. $\mathcal{M}_1 = \varnothing$.

Category II: One Pair:

2. $\mathcal{M}_2 = \{(0,1)\}$,
3. $\mathcal{M}_3 = \{(0,2)\}$,
4. $\mathcal{M}_4 = \{(0,3)\}$,
5. $\mathcal{M}_5 = \{(0,4)\}$,
6. $\mathcal{M}_6 = \{(1,2)\}$,
7. $\mathcal{M}_7 = \{(1,3)\}$,
8. $\mathcal{M}_8 = \{(1,4)\}$,
9. $\mathcal{M}_9 = \{(2,3)\}$,
10. $\mathcal{M}_{10} = \{(2,4)\}$,
11. $\mathcal{M}_{11} = \{(3,4)\}$.

Category III: Two Disjoint Pairs:

12. $\mathcal{M}_{12} = \{(0,1),(2,3)\}$,
13. $\mathcal{M}_{13} = \{(0,1),(2,4)\}$,
14. $\mathcal{M}_{14} = \{(0,1),(3,4)\}$,
15. $\mathcal{M}_{15} = \{(0,2),(3,4)\}$,
16. $\mathcal{M}_{16} = \{(1,2),(3,4)\}$,

Category IV: Two Nested Pairs:

17. $\mathcal{M}_{17} = \{(0,3),(1,2)\}$,
18. $\mathcal{M}_{18} = \{(0,4),(1,2)\}$,
19. $\mathcal{M}_{19} = \{(0,4),(1,3)\}$,
20. $\mathcal{M}_{20} = \{(0,4),(2,3)\}$,
21. $\mathcal{M}_{21} = \{(1,4),(2,3)\}$.

Verification: $1 + 10 + 5 + 5 = 21 = M_5$. ✓

Enumeration of Composite Moves for $n = 6$. We have six indices: 0, 1, 2, 3, 4, 5. According to theory $M_6 = 51$. Category I: Empty Composite Move:

1. $\mathcal{M}_1 = \varnothing$.

Category II: One Pair. All $\binom{6}{2} = 15$ pairs:

2. $\mathcal{M}_2 = \{(0,1)\}$,
3. $\mathcal{M}_3 = \{(0,2)\}$,
4. $\mathcal{M}_4 = \{(0,3)\}$,
5. $\mathcal{M}_5 = \{(0,4)\}$,
6. $\mathcal{M}_6 = \{(0,5)\}$,
7. $\mathcal{M}_7 = \{(1,2)\}$,
8. $\mathcal{M}_8 = \{(1,3)\}$,
9. $\mathcal{M}_9 = \{(1,4)\}$,
10. $\mathcal{M}_{10} = \{(1,5)\}$,
11. $\mathcal{M}_{11} = \{(2,3)\}$,
12. $\mathcal{M}_{12} = \{(2,4)\}$,
13. $\mathcal{M}_{13} = \{(2,5)\}$,
14. $\mathcal{M}_{14} = \{(3,4)\}$,
15. $\mathcal{M}_{15} = \{(3,5)\}$,
16. $\mathcal{M}_{16} = \{(4,5)\}$.

Category III: Two Disjoint Pairs, (i_1, j_1), (i_2, j_2) with $j_1 < i_2$:

17. $\mathcal{M}_{17} = \{(0,1),(2,3)\}$,
18. $\mathcal{M}_{18} = \{(0,1),(2,4)\}$,
19. $\mathcal{M}_{19} = \{(0,1),(2,5)\}$,
20. $\mathcal{M}_{20} = \{(0,1),(3,4)\}$,
21. $\mathcal{M}_{21} = \{(0,1),(3,5)\}$,
22. $\mathcal{M}_{22} = \{(0,1),(4,5)\}$,
23. $\mathcal{M}_{23} = \{(0,2),(3,4)\}$,
24. $\mathcal{M}_{24} = \{(0,2),(3,5)\}$,
25. $\mathcal{M}_{25} = \{(0,2),(4,5)\}$,
26. $\mathcal{M}_{26} = \{(0,3),(4,5)\}$,
27. $\mathcal{M}_{27} = \{(1,2),(3,4)\}$,
28. $\mathcal{M}_{28} = \{(1,2),(3,5)\}$,
29. $\mathcal{M}_{29} = \{(1,2),(4,5)\}$,
30. $\mathcal{M}_{30} = \{(1,3),(4,5)\}$,
31. $\mathcal{M}_{31} = \{(2,3),(4,5)\}$.

Category IV: Two Nested Pairs. Structure $i_a < i_b < j_b < j_a$:

32. $\mathcal{M}_{32} = \{(0,3),(1,2)\}$,
33. $\mathcal{M}_{33} = \{(0,4),(1,2)\}$,
34. $\mathcal{M}_{34} = \{(0,4),(1,3)\}$,
35. $\mathcal{M}_{35} = \{(0,4),(2,3)\}$,
36. $\mathcal{M}_{36} = \{(0,5),(1,2)\}$,
37. $\mathcal{M}_{37} = \{(0,5),(1,3)\}$,
38. $\mathcal{M}_{38} = \{(0,5),(1,4)\}$,
39. $\mathcal{M}_{39} = \{(0,5),(2,3)\}$,
40. $\mathcal{M}_{40} = \{(0,5),(2,4)\}$,
41. $\mathcal{M}_{41} = \{(0,5),(3,4)\}$,
42. $\mathcal{M}_{42} = \{(1,4),(2,3)\}$,
43. $\mathcal{M}_{43} = \{(1,5),(2,3)\}$,
44. $\mathcal{M}_{44} = \{(1,5),(2,4)\}$,
45. $\mathcal{M}_{45} = \{(1,5),(3,4)\}$,
46. $\mathcal{M}_{46} = \{(2,5),(3,4)\}$.

Category V: Three Disjoint Pairs:

47. $\mathcal{M}_{47} = \{(0,1),(2,3),(4,5)\}$.

Category VI: Two Disjoint + One Nested:

48. $\mathcal{M}_{48} = \{(0,3),(1,2),(4,5)\}$,
49. $\mathcal{M}_{49} = \{(0,1),(2,5),(3,4)\}$.

Category VII: Three Double-Nested Pairs:

50. $\mathcal{M}_{50} = \{(0,5),(1,4),(2,3)\}$.

Category VIII: Mixed Structure

51. $\mathcal{M}_{51} = \{(0,5),(1,2),(3,4)\}$.

Verification: $1 + 15 + 15 + 15 + 1 + 2 + 1 + 1 = 51 = M_6$. ✓

Structure of Composite Moves for $n = 7$. We have $M_7 = 127$ configurations. Full enumeration would take many pages, so we present the combinatorial structure. Main Categories:

- Empty configuration: 1 element.
- One pair: $\binom{7}{2} = 21$ elements.
- Two disjoint pairs: We choose 4 indices out of 7 and split them into two pairs: $\binom{7}{4} \cdot \frac{1}{2}\binom{4}{2} = 35 \cdot 3 = 105$ elements. But we must exclude crossings. The correct number of disjoint pairs (without crossings): 35 elements.
- Two nested pairs: We choose an outer pair (i, j) with span ≥ 3, then an inner pair from $\{i+1, \ldots, j-1\}$: 35 elements.
- Three pairs: Combinations of three pairs (all disjoint or with nestings): 21 elements.
- Four or more pairs: Deeply nested structures and complex combinations: 14 elements.

Verification: $1 + 21 + 35 + 35 + 21 + 14 = 127 = M_7$. ✓

Structure of Composite Moves for $n = 8$. Here $M_8 = 323$ configurations. Category distribution

- Empty: 1.
- One pair: $\binom{8}{2} = 28$.
- Two pairs: 126 (disjoint + nested).
- Three pairs: 112.
- Four pairs: 51.
- More pairs: 5.

Verification by recurrence:

$$M_8 = M_7 + \sum_{k=0}^{6} M_k M_{6-k} = 127 + 196 = 323. \quad \checkmark \tag{4}$$

Structure of Composite Moves for $n = 9$. For $n = 9$ we have $M_9 = 835$ configurations.
Category distribution:

- Empty: 1.
- One pair: $\binom{9}{2} = 36$.
- Two pairs: 330.
- Three pairs: 294.
- Four pairs: 140.
- Five or more: 34.

Verification by recurrence:

$$M_9 = M_8 + \sum_{k=0}^{7} M_k M_{7-k} = 323 + 512 = 835. \quad \checkmark \tag{5}$$

Structure of Composite Moves for $n = 10$. Here $M_{10} = 2188$ configurations.
Category distribution (approximate)

- Empty: 1.
- One pair: $\binom{10}{2} = 45$.
- Two pairs: 825.
- Three pairs: 770.
- Four pairs: 385.
- Five pairs: 126.
- More pairs: 36.

Verification by recurrence:

$$M_{10} = M_9 + \sum_{k=0}^{8} M_k M_{8-k} = 835 + 1353 = 2188. \quad \checkmark \tag{6}$$

Remarks on Combinatorial Growth. For $n \geq 7$ the number of configurations grows exponentially ($M_n \sim c \cdot 3^n/n^{3/2}$), which makes full enumeration impractical. Despite this, the algorithm using dynamic programming remains efficient, running in time $O(n^3)$, without requiring explicit generation of all configurations – the recurrence naturally searches the state space.

The bijection with Motzkin objects remains of key importance for understanding the theoretical structure, despite the lack of explicitly listing all M_n composite moves.

5 Experimental Setup

The experiment was conducted in the test environment defined in [**?**], as follows. All experiments were conducted on a MacBook Pro with an M4 Pro processor, 24 GB of RAM, running macOS 26.0.1 (25A362) and Python 3.11.

Benchmarks Taillard instances [**?**] of varying (n, m) (e.g., $n \in \{50, 100, 200, 500\}$, $m \in \{10, 20\}$). Each run is constrained by a wall-clock time budget (e.g., 100 ms). Seeds are fixed for reproducibility.

Parameters The following parameter values were used in the experiments:

- Tabu tenure: 10
- SA: $T_{\text{init}} = 1000$, $T_{\text{final}} = 1$, $\alpha = 0.95$, reheating factor $r = 1.5$, stagnation threshold 500 (ms).

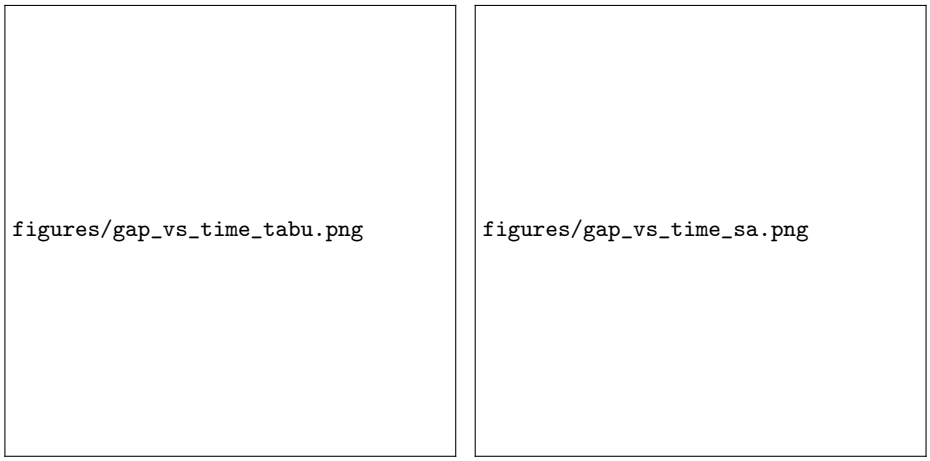
Metrics

- Best $C_{\max}$ vs. time.
- Convergence curves (multi-neighborhood overlay).
- Effective moves per second (normalized).
- Plateau detection latency.

The choice of experimental parameters and metrics was primarily aimed at assessing the impact of the different neighborhood structures on the final solution quality, rather than evaluating the absolute performance of the algorithms.

6 Results and Discussion

The results presented in Tables **??**–**??** and in the plots of the mean gap as a function of the time limit (Fig. **??**) indicate that the choice of the neighborhood structure has a key impact on the final performance of the algorithm as a function of time. The averaged results indicate that, for all analyzed time limits, the ranking of neighborhoods is, in principle, similar. *Dynasearch* has the lowest gap values. It is optimized computationally, which allows for performing a larger number of iterations within the same computation time. Additionally, for large datasets, a single *Dynasearch* iteration takes much more time than the overall time limit. It should be noted that the time-limit condition was checked at the beginning of each algorithm iteration. After the limit was exceeded, it was still possible to finish the already started iteration and include its outcome. Therefore, in some cases the reported results correspond to an effective computation time that exceeds the nominal time limit. This allows us to omit the results for this neighborhood in the further part. In contrast, the *Fibonacci* neighborhood performs the least favorably over the entire examined range. At the same time, in Fig. **??** we observe a systematic improvement in solution quality with increasing T_{limit}. The rate of change depends on the applied neighborhood. For some of the



(a) Tabu Search (b) Simulated Annealing

Fig. 2: Comparison of solution quality (mean gap to the lower bound, LB) as a function of the time limit for two metaheuristics and four neighborhoods.

applied methods, a rapid transition to the plateau phase is observed, which is visible in Fig. **??**. This indicates the occurrence of a saturation effect and limits the possibility of further searching for better solutions. The *Motzkin* neighborhood is relatively large, but it allows for performing many swaps during a single iteration in a strictly structured manner. On average, this approach outperforms the *Adjacent* neighborhood. This advantage becomes more visible for larger time limits, which suggests an advantage of multi-swap moves requiring sufficiently large time over simpler solutions. A natural direction for further research is an attempt to improve the computational efficiency of the implementations of the *Motzkin* and *Fibonacci* neighborhoods in order to increase the number of iterations that can be performed within the same time constraint. For the purpose of increasing the competitiveness of the new neighborhoods with respect to the classical ones.

(a) Tabu Search, t=500ms, n=100, m=20 (b) Simulated Annealing, t=2000ms, n=200, m=10

Fig. 3: Example graphs for individual algorithm runs.

Time limit: $tl_{\text{ms}} = 100$

file	n	m	adj_{TABU}	adj_{SA}	fib_{TABU}	fib_{SA}	dyn_{TABU}	dyn_{SA}	mot_{TABU}	mot_{SA}
tai100_10	100	10	18.41	19.62	**17.34**	**17.34**	17.33	17.33	18.98	18.98
tai100_20	100	20	27.55	27.94	**24.50**	**24.50**	25.50	25.50	27.20	27.20
tai100_5	100	5	11.84	11.98	**11.15**	**11.15**	11.01	11.01	12.48	12.25
tai200_10	200	10	15.40	15.28	**14.01**	**14.01**	13.20	13.20	14.79	14.79
tai200_20	200	20	23.26	23.33	**21.62**	**21.62**	20.71	20.71	22.06	22.06
tai20_10	20	10	19.59	20.73	29.73	29.73	14.68	14.68	**16.61**	17.61
tai20_20	20	20	31.89	30.30	36.37	36.37	24.83	24.83	**25.56**	25.58
tai20_5	20	5	**7.18**	7.98	17.39	17.39	6.18	6.18	8.39	8.39
tai500_20	500	20	16.25	16.03	**15.57**	**15.57**	14.51	14.51	15.93	15.93
tai50_10	50	10	**19.65**	22.10	20.23	20.23	16.84	17.00	19.86	20.17
tai50_20	50	20	32.54	34.23	32.19	32.23	31.14	31.14	**30.57**	**30.57**
tai50_5	50	5	10.71	**8.38**	12.13	12.13	6.20	6.07	10.35	10.45
AVG			19.52	19.82	21.02	21.02	16.84	16.85	18.57	18.66

Table 2: Average results for $T_{\text{limit}} = 100$ ms.

Time limit: $tl_{\text{ms}} = 500$

file	n	m	adj_{TABU}	adj_{SA}	fib_{TABU}	fib_{SA}	dyn_{TABU}	dyn_{SA}	mot_{TABU}	mot_{SA}
tai100_10	100	10	**16.92**	17.16	17.19	17.19	15.69	16.08	17.22	17.51
tai100_20	100	20	24.98	26.33	**23.02**	**23.02**	25.50	25.50	26.30	26.30
tai100_5	100	5	10.89	**6.65**	11.15	11.15	8.73	8.73	11.99	11.99
tai200_10	200	10	14.10	14.72	**12.72**	**12.72**	13.20	13.20	14.79	14.79
tai200_20	200	20	22.09	22.80	**19.74**	19.76	20.71	20.71	22.06	22.06
tai20_10	20	10	**14.91**	20.37	29.73	29.73	14.68	14.68	16.61	17.61
tai20_20	20	20	27.35	30.22	36.37	36.37	24.83	24.83	**25.49**	**25.49**
tai20_5	20	5	**5.23**	6.52	17.39	17.39	6.18	6.18	8.39	8.39
tai500_20	500	20	16.09	16.00	**15.13**	**15.13**	14.51	14.51	15.93	15.93
tai50_10	50	10	17.97	**12.63**	20.23	20.23	9.94	9.94	16.77	16.77
tai50_20	50	20	30.51	26.72	32.19	32.19	23.31	23.31	25.70	**25.64**
tai50_5	50	5	10.22	**3.54**	12.13	12.13	2.95	2.95	8.31	8.92
AVG			17.61	16.97	20.58	20.58	15.02	15.05	17.46	17.62

Table 3: Average results for $T_{\text{limit}} = 500$ ms.

Time limit: $tl_{\text{ms}} = 1000$

file	n	m	adj_{TABU}	adj_{SA}	fib_{TABU}	fib_{SA}	dyn_{TABU}	dyn_{SA}	mot_{TABU}	mot_{SA}
tai100_10	100	10	16.68	**12.14**	17.19	17.19	14.03	14.08	16.35	16.42
tai100_20	100	20	24.06	25.41	**23.02**	**23.02**	23.43	23.43	24.51	24.51
tai100_5	100	5	10.83	**4.43**	11.15	11.15	7.04	7.04	11.53	11.53
tai200_10	200	10	13.44	14.36	**12.71**	**12.71**	13.20	13.20	14.79	14.79
tai200_20	200	20	21.44	22.49	**19.11**	**19.11**	20.71	20.71	22.06	22.06
tai20_10	20	10	**13.34**	20.25	29.73	29.73	14.68	14.68	16.61	17.61
tai20_20	20	20	25.75	30.22	36.37	36.37	24.83	24.83	**25.49**	**25.49**

Time limit: $tl_{\text{ms}} = 2000$

file	n	m	adj_{TABU}	adj_{SA}	fib_{TABU}	fib_{SA}	dyn_{TABU}	dyn_{SA}	mot_{TABU}	mot_{SA}
tai100_10	100	10	16.53	**9.63**	17.19	17.19	11.19	11.10	15.47	15.54
tai100_20	100	20	23.61	**19.84**	23.02	23.02	21.31	21.31	23.38	23.85
tai100_5	100	5	10.81	**3.41**	11.15	11.15	4.83	4.78	11.41	11.41
tai200_10	200	10	13.27	13.77	**12.71**	**12.71**	13.20	13.20	14.35	14.35
tai200_20	200	20	20.70	21.80	**18.65**	**18.65**	20.71	20.71	22.06	22.06
tai20_10	20	10	**12.92**	19.80	29.73	29.73	14.68	14.68	16.61	17.61
tai20_20	20	20	**24.74**	30.22	36.37	36.37	24.83	24.83	25.49	25.49
tai20_5	20	5	**4.40**	6.31	17.39	17.39	6.18	6.18	8.39	8.39
tai500_20	500	20	15.75	15.91	**14.48**	**14.48**	14.51	14.51	15.93	15.93
tai50_10	50	10	17.33	**10.94**	20.23	20.23	6.95	6.95	14.52	16.35
tai50_20	50	20	28.43	24.63	32.19	32.19	16.65	16.69	**22.86**	23.62
tai50_5	50	5	9.80	**1.96**	12.13	12.13	2.95	2.95	8.06	8.52
AVG			16.52	14.85	20.44	20.44	13.17	13.16	16.54	16.93

Table 5: Average results for $T_{\text{limit}} = 2000$ ms.

Time limit: $tl_{\text{ms}} = 5000$

file	n	m	adj_{TABU}	adj_{SA}	fib_{TABU}	fib_{SA}	dyn_{TABU}	dyn_{SA}	mot_{TABU}	mot_{SA}
tai100_10	100	10	16.52	**8.31**	17.19	17.19	6.48	6.57	14.32	14.82
tai100_20	100	20	23.22	**17.52**	23.02	23.02	17.62	17.62	21.75	21.92
tai100_5	100	5	10.72	**2.40**	11.15	11.15	2.34	2.34	9.22	10.73
tai200_10	200	10	13.19	**8.51**	12.71	12.71	13.20	13.20	14.07	14.26
tai200_20	200	20	19.83	20.18	**18.60**	**18.60**	20.71	20.71	20.42	20.85
tai20_10	20	10	**12.58**	19.58	29.73	29.73	14.68	14.68	16.61	17.61
tai20_20	20	20	**22.61**	30.16	36.37	36.37	24.83	24.83	25.49	25.49
tai20_5	20	5	**4.17**	6.24	17.39	17.39	6.18	6.18	8.39	8.39
tai500_20	500	20	15.35	15.82	13.93	**13.91**	14.51	14.51	15.93	15.93
tai50_10	50	10	17.10	**9.71**	20.23	20.23	6.95	6.95	13.35	16.35
tai50_20	50	20	27.65	24.44	32.19	32.19	14.80	14.84	**20.95**	23.44
tai50_5	50	5	9.38	**1.96**	12.13	12.13	2.95	2.95	8.06	8.52
AVG			16.03	13.74	20.39	20.39	12.10	12.12	15.71	16.53

Table 6: Average results for $T_{\text{limit}} = 5000$ ms.

Time limit: $tl_{\text{ms}} = 10000$

file	n	m	adj_{TABU}	adj_{SA}	fib_{TABU}	fib_{SA}	dyn_{TABU}	dyn_{SA}	mot_{TABU}	mot_{SA}
tai100_10	100	10	16.50	**6.19**	17.19	17.19	4.11	4.11	13.58	14.40
tai100_20	100	20	22.92	**17.06**	23.02	23.02	15.50	15.50	19.87	20.69
tai100_5	100	5	10.57	**2.37**	11.15	11.15	1.93	1.93	8.38	10.73
tai200_10	200	10	13.16	**7.23**	12.71	12.71	11.66	11.66	14.07	14.07
tai200_20	200	20	19.63	**14.61**	18.60	18.60	19.32	19.32	20.17	20.17
tai20_10	20	10	**12.07**	19.58	29.73	29.73	14.68	14.68	16.61	17.61
tai20_20	20	20	**22.12**	30.16	36.37	36.37	24.83	24.83	25.49	25.49
tai20_5	20	5	**4.04**	6.24	17.39	17.39	6.18	6.18	8.39	8.39
tai500_20	500	20	14.98	15.53	**13.62**	**13.62**	14.51	14.51	15.93	15.93
tai50_10	50	10	17.10	**9.06**	20.23	20.23	6.95	6.95	13.35	16.35
tai50_20	50	20	26.58	24.32	32.19	32.19	14.73	14.80	**20.37**	23.44
tai50_5	50	5	9.38	**1.81**	12.13	12.13	2.95	2.95	8.06	8.52
AVG			15.75	12.85	20.36	20.36	11.45	11.45	15.36	16.32

Table 7: Average results for $T_{\text{limit}} = 10000$ ms.