POP - Documentation

Sofiya Yedzeika(335978)
Alesia Filinkova(336180)

Introduction

The project addresses the problem *Hash Code Photo Slideshow* (Kaggle: hashcode-photo-slideshow), in which one must arrange photos into a slide sequence, using assigned tags. The goal is to achieve a high-quality slideshow (evaluated based on transitions between consecutive slides), using an experimental approach: comparing several heuristics and tuning parameters. In the following sections, we analyze the impact of parameters (e.g., number of neighbors k) on quality and computational cost, to select a practically good solution, even if not globally optimal.

Initial data preprocessing

Context and goals of preprocessing

The goal of the preprocessing step is to **understand the data structure**, prepare it for further experiments (e.g., choosing methods for combining photos into slides) and identify potential computational limitations resulting from the dataset's scale. In this project, we do not aim for an absolutely optimal solution (the solution space is too large), but rather a **good and empirically justified solution**. This section serves as a starting point for subsequent notebooks, where algorithmic parameters (e.g., number of neighbors k) and method comparisons will appear.

Initial data format and loading

The input dataset is a text file in which **each line corresponds to a single photo** and contains:

photo identifier (`id`),

orientation: `H` (horizontal) or `V` (vertical),

list of semantic tags (e.g., `t82`, `tx1`, ...).

Loading involves parsing each line into a record $\langle id, orientation, T \rangle$, where T is the set of unique tags for the photo. Next: (1) we split the data into `H` and `V`, (2) compute dataset statistics, (3) save data in an auxiliary format for subsequent experiments. In further computations we treat tags as a set (without repetitions), since the scoring function depends on set operations: $\cap$, $\setminus$, $\cup$.

Dataset Statistics

Basic metrics for the entire dataset are shown in Tab. 1. Key observations include: (i) **large number of records** ($N = 90000$), (ii) **moderate number of tags per photo** (average ≈ 10), (iii) **constant, small vocabulary dimensionality** (220), which implies sparse vector representations.

Tabela 1: Basic statistics for the entire dataset.

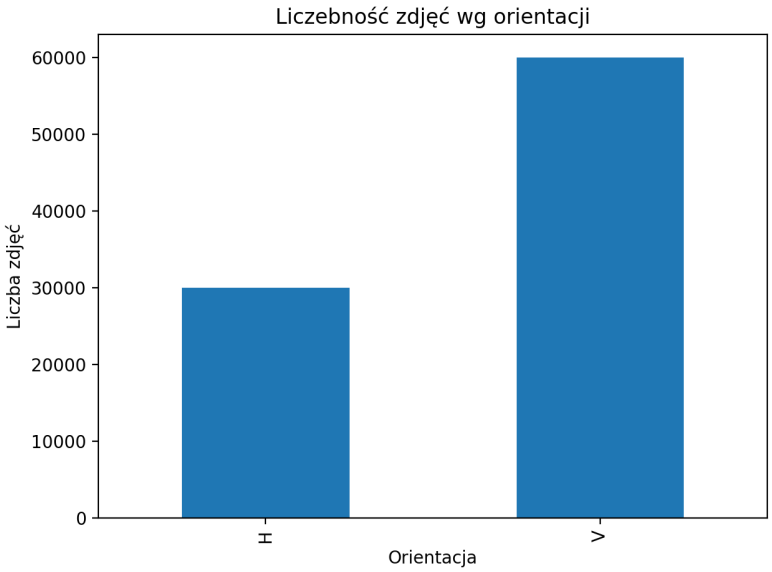
metric	value
N	90000
average number of tags	10.0251
median	10
min	1
max	19
std deviation	4.2585
number of unique tags	220
density (average)	0.0456

Tabela 2: Statistics of tag count by orientation.

Orientation	n	Average	Median	Min	Max	Std Dev
H	30000	10.0400	10	1	19	4.2542
V	60000	10.0176	10	2	19	4.2606

Photo Orientation

The dataset is dominated by vertical orientation, which has practical consequences for slide construction (vertical photos must be combined more frequently).

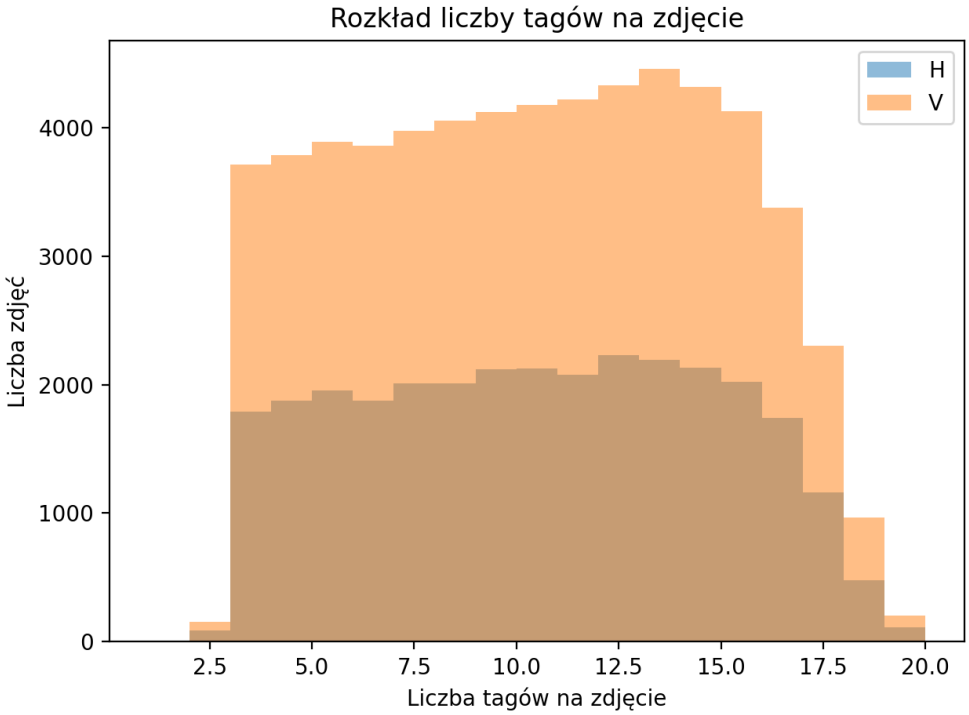


Rysunek 1: Number of photos by orientation.

Consequence for slide construction. Since vertical photos must be paired, with 60000 photos of type `V` we can create at most 30000 vertical slides. Combined with 30000 horizontal slides, this yields approximately 60000 slides in a complete slideshow. This means that the quality of the heuristic for combining vertical photos can have as significant an impact on the final score as the selection of slide sequence order.

Distribution of Tags per Photo

The distribution of tag count (Fig. 2) is wide (from 1 to 19), but the median is 10. This indicates that most photos have similar "density" of semantic information.



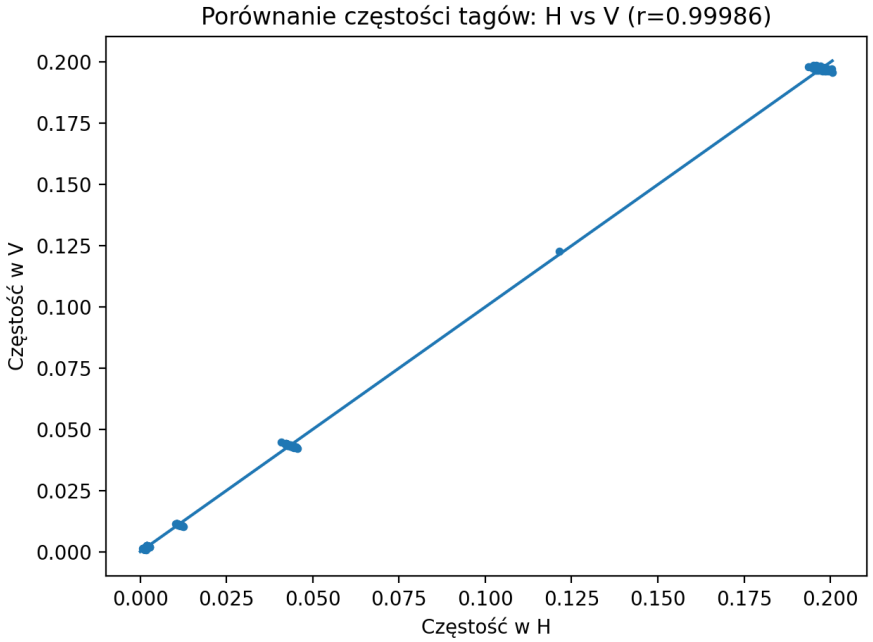
Rysunek 2: Distribution of tag count per photo, separately for `H` and `V`.

Does Orientation Affect Tag Distribution?

We tested whether the tag distribution in `H` and `V` differs significantly:

> Pearson correlation coefficient between tag frequencies is $r = 0.99986$,
>
> on the scatter plot (Fig. 3), points lie close to the line $y = x$.

Conclusion: **orientation practically does not change the semantic profile** of photos in terms of tags. This is good news, as it allows designing photo selection methods without separate models for `H`/`V` (at least at the tag level).



Rysunek 3: Comparison of tag frequencies in datasets `H` and `V`.

Practical Conclusions for Subsequent Stages

Based on the preliminary analysis, we can formulate observations important for heuristic design:

The `H`/`V` proportion determines the number of slides. The dataset contains twice as many vertical photos as horizontal ones, so the vertical pairing stage is critical (resulting in a comparable number of vertical and horizontal slides).

A typical photo has about 10 tags. Operations on tag sets (intersection/difference) are relatively inexpensive per pair of objects, but in subsequent stages we must avoid reviewing all pairs ($O(N^2)$ complexity).

Orientation does not change the tag profile. Since the tag distribution for `H` and `V` is nearly identical, semantic selection methods can be designed jointly, with orientation treated mainly as a slide construction constraint.

Computational Complexity of Preprocessing

Let N denote the number of photos and $T = \sum_{i=1}^{N} |tags_i|$ the total number of tag occurrences (in practice $T \approx 10N$).

File loading and parsing: $O(N + T)$ time; $O(N + T)$ memory for data storage.

Division into `H`/`V`: $O(N)$ time, negligible memory overhead (references/new lists).

Tag frequency counting: $O(T)$ time, $O(|V|)$ memory where $|V| = 220$.

JSON writing: $O(N + T)$ time (serialization), memory dependent on buffering.

These are linear costs and acceptable for $N = 90000$, but in subsequent stages (e.g., finding similar photos) we must be careful with $O(N^2)$ algorithms.

Combining Vertical Photos into Slides

Stage Goal and Tested Heuristics

In the *Photo Slideshow* task, vertical photos (`V`) cannot form a slide individually — each vertical slide consists of a **pair** of photos. With 60 000 photos of type `V`, this requires creating 30 000 pairs, so decisions made in this stage affect half the entire slideshow.

Three heuristics for combining vertical photos were tested:

Random pairing (baseline) — fast, but without control over tag redundancy.

Pairing similar photos — we select photos with the largest possible common tags $|T(a) \cap T(b)|$.

Pairing different photos — we select photos with the smallest possible common tags (often 0 in practice), to maximize $|T(a) \cup T(b)|$.

The last two methods use a parameter k that limits the number of candidates considered for a given photo. We sort the `V` set in ascending order by tag count and for index i we search for a pair in window $[i - k,\ i + k]$. Sorting by tag count means that in window $[i - k, i + k]$ we compare photos with similar "size" of tag set. This makes intersections and unions of tags more stable, and the heuristic avoids extreme matches (e.g., a photo with 2 tags paired with a photo with 19 tags). This is important from a complexity perspective: instead of full review ($O(N^2)$) we get a heuristic of $O(N \cdot k \cdot \bar{t})$, where $\bar{t} \approx 10$ is the average number of tags.

Pair Evaluation Metrics

Since we are **not yet establishing the slide order** at this stage, we evaluated pairing quality through pair statistics:

average intersection $\overline{|T(a) \cap T(b)|}$ (redundancy),

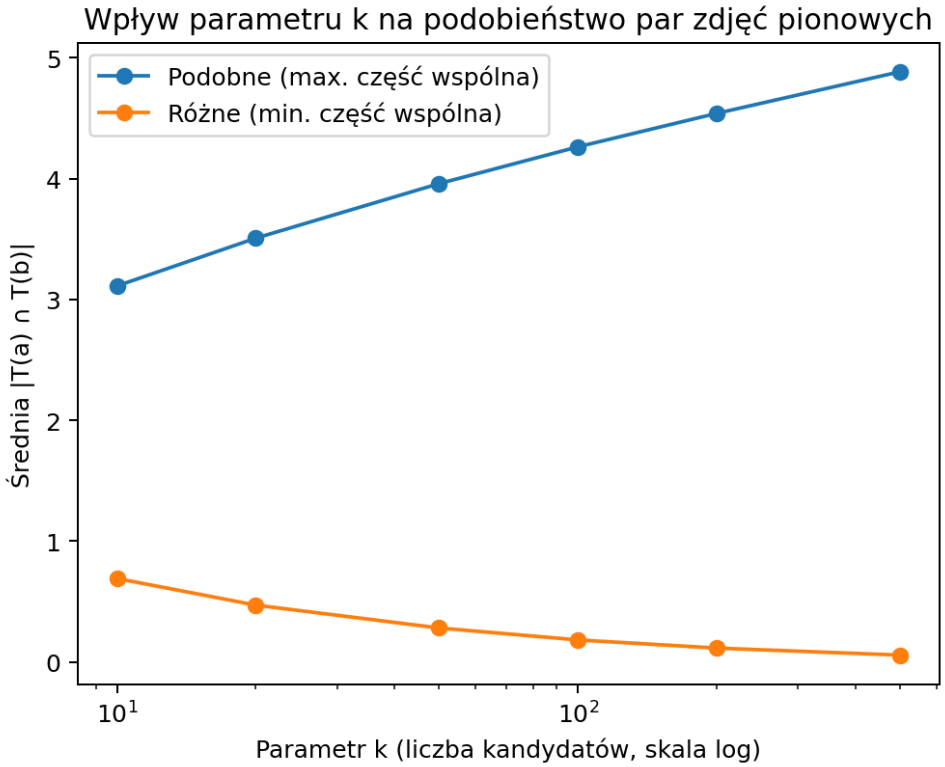
average union $\overline{|T(a) \cup T(b)|}$ (number of slide tags),

fraction of pairs with zero intersection $\%(|T(a) \cap T(b)| = 0)$.

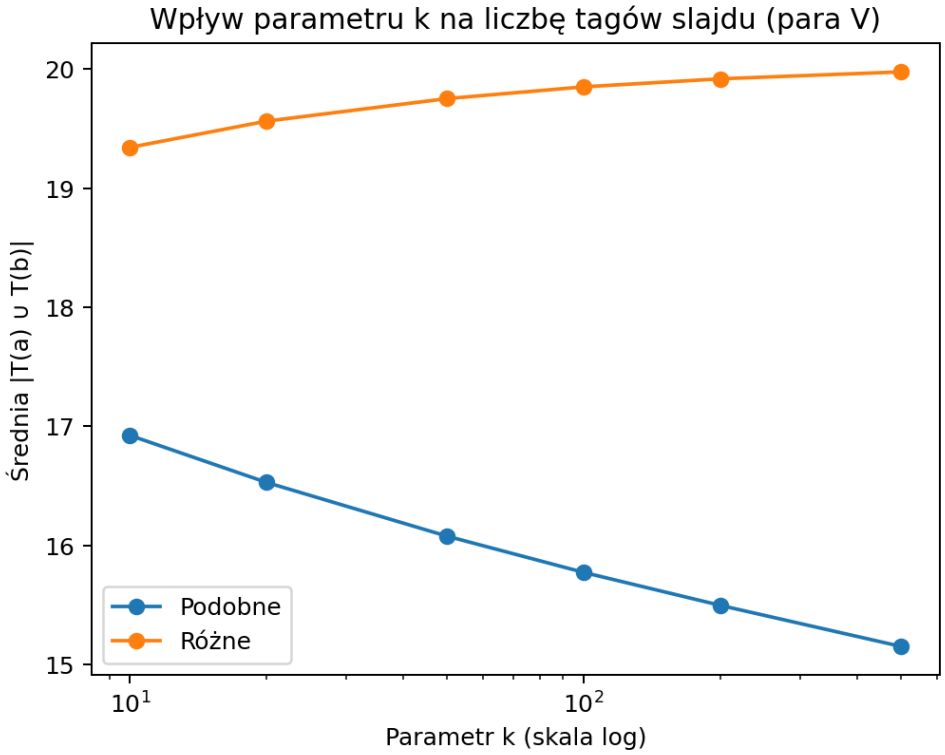
Experiments were performed on the full set $N_V = 60\,000$ vertical photos (i.e., 30 000 pairs). The intuition is that larger $|T(a) \cup T(b)|$ increases the "potential" of a slide to earn transition points, because in the $\min(\cdot)$ function we need both intersection and differences between consecutive slides. Overly homogeneous slides (large $|T(a) \cap T(b)|$) limit the number of unique tags, reducing the space of possible good transitions.

Impact of Parameter k

Tested values: $k \in \{10, 20, 50, 100, 200, 500\}$. Figs. 4 and 5 show that increasing k yields increasingly better matches, but with diminishing returns (saturation effect). Simultaneously, cost grows approximately linearly with k ($O(N \cdot k \cdot \bar{t})$ complexity).



Rysunek 4: Impact of k on average similarity in a pair of vertical photos.



Rysunek 5: Impact of k on average number of slide tags (union of two `V`).

Results Tables and Interpretation

Tabela 3: Combining vertical photos: similar method (max intersection).

k	$\overline{\lvert\cap\rvert}$	$\overline{\lvert\cup\rvert}$	$\%(\lvert\cap\rvert \geq 5)$
10	3.112	16.923	24.12%
20	3.507	16.528	31.17%
50	3.959	16.076	39.36%
100	4.263	15.772	44.32%
200	4.540	15.495	48.60%
500	4.885	15.150	53.86%

Tabela 4: Combining vertical photos: different method (min intersection).

k	$\overline{\lvert\cap\rvert}$	$\overline{\lvert\cup\rvert}$	$\%(\lvert\cap\rvert = 0)$
10	0.691	19.344	58.18%
20	0.471	19.564	67.48%
50	0.282	19.753	77.68%
100	0.183	19.852	84.35%
200	0.116	19.920	89.61%
500	0.058	19.978	94.73%

For the **similar** method, increasing k boosts tag intersection (from ca. 3.11 for $k = 10$ to ca. 4.89 for $k = 500$), but simultaneously decreases the number of unique tags in the slide (decline in $\overline{|T(a) \cup T(b)|}$). This method creates more "homogeneous" slides, at the expense of diversity.

For the **different** method, the trend is reversed: as k increases, intersection rapidly decreases, and the average number of slide tags grows to ca. 20 (approximately $2\times$ the average tags per photo), indicating that tags in pairs largely do not overlap. Simultaneously, the fraction of pairs with zero intersection increases (from ca. 58% for $k = 10$ to ca. 95% for $k = 500$), meaning minimal tag redundancy within a slide.

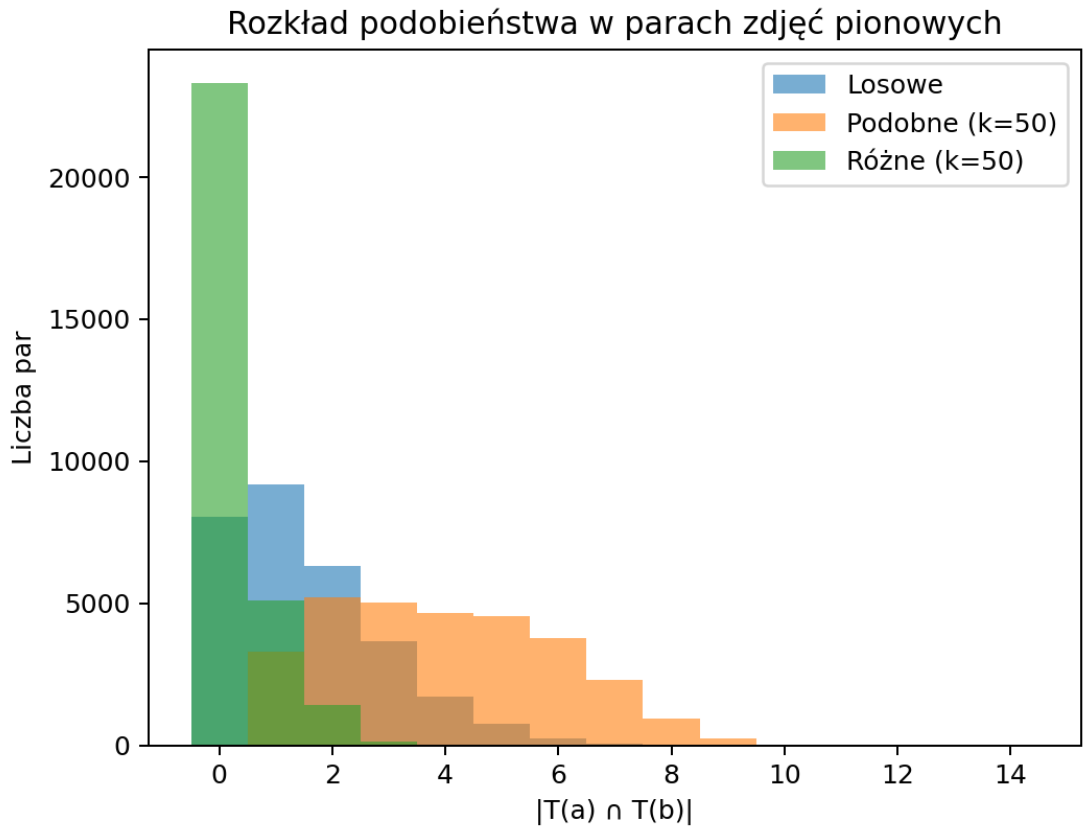
Comparison with Baseline and Parameter Selection for Subsequent Stages

Random baseline (average of 3 runs) gives approximately $\overline{\lvert\cap\rvert} \approx 1.51$ and $\overline{\lvert\cup\rvert} \approx 18.53$, which is clearly worse than the "different" method at moderate k.

Tabela 5: Baseline: random pairing of vertical photos (3 different seeds).

metric	average	std dev
$\overline{\lvert T(a) \cap T(b)\rvert}$	1.509	0.008
$\overline{\lvert T(a) \cup T(b)\rvert}$	18.526	0.008
$\%(\lvert T(a) \cap T(b)\rvert = 0)$	26.93%	0.11%

Practical choice. Based on the comparison of methods at $k = 50$, the "different" method gives the lowest values of $|T(a) \cap T(b)|$, which translates to greater diversity of tags in vertical slides. Therefore, in subsequent experiments we adopted "different" pairing as the baseline variant (parameter k will be tuned further).



Rysunek 6: Distribution of $|T(a) \cap T(b)|$ for random pairing and greedy methods (for $k = 50$).

Counterfactual Analysis of Method Choice

It is worth considering how subsequent stages of the solution would change if a different strategy for combining vertical photos had been chosen. Using a method that maximizes tag similarity would lead to thematically specialized slides, which on one hand would increase local coherence, but on the other would limit transition diversity.

Conversely, random pairing, though very fast, would generate slides of unpredictable quality, making stable sequence optimization difficult. In this context, the chosen method of pairing photos with low tag similarity represents a reasonable compromise between computational cost and usefulness in subsequent algorithmic stages.

Chapter Summary – Implications for Subsequent Stages

Analysis of vertical photo pairing methods revealed that the pairing method has a **large impact on slideshow quality**. Pairing photos with small tag intersection leads to semantically richer slides, which significantly increases the potential for earning high transition points between consecutive slides.

At the same time, exact pairing algorithms are computationally infeasible for the full dataset. Therefore, in subsequent project stages we adopted an approximate heuristic with parameter k, which provides a favorable compromise between solution quality and computation time.

Constructing Slide Sequence Order (Sequence Optimization)

Motivation

After creating slides (30 000 slides from pairs of vertical photos and 30 000 horizontal slides), the critical stage remains: **arranging the slide sequence**, which maximizes the sum of transition points. For two consecutive slides A, B the transition score equals:

$$\text{score}(A, B) = \min\big(|T(A) \cap T(B)|,\ |T(A) \setminus T(B)|,\ |T(B) \setminus T(A)|\big),$$

and the entire slideshow score is the sum over all adjacent pairs. Since the permutation space is gigantic (60 000!), in practice approximate heuristics are used.

Global vs. Local Optimum. Finding the global optimum would require searching an astronomical number of slide permutations, which is computationally infeasible. The applied methods (NN, Mixed) optimize *locally* — they greedily select the next slide based on limited information (candidate pool), so they may get stuck in locally good solutions. In practice, improvement comes from increasing parameter k (better candidate selection) and multiple runs with different random seeds, selecting the best result.

Tested Heuristics

Four approaches to constructing sequence order were compared:

Random (*Random*) – slide permutation as a reference point.

Greedy nearest neighbor (*Nearest neighbor, NN*) – we build the slideshow step by step, at each step selecting the next slide that maximizes the transition score. To limit cost, we do not review all remaining slides, but instead randomly sample a pool of k candidates (**parameter** k).

Grouping by tag (*Grouped*) – we first divide slides into groups by a selected "representative" (smallest tag), and then combine groups into a single sequence.

Mixed method (*Mixed*) – a two-level heuristic: we first group slides by tag "representative", and then build order locally (NN within groups) and globally (NN between groups). In practice it combines the benefits of grouping (block ordering) and greedy neighbor selection (good local transitions). Parameter k in Mixed denotes the number of candidates considered in NN steps (both within groups and when linking groups). The implementation also uses `group_key` (definition of group representative tag; in experiments: `min`) and `k_group` (limit on number of considered "neighboring" groups when switching between groups). In experiments we use the smallest tag as group representative (lexicographic order).

Runtime Implementation (Consistent with Notebooks). For runs on the full dataset we use the `solver.py` script, which implements the same pipeline as the notebooks: (1) pairing vertical photos (random/similar/different) with parameter k_{pair}, (2) constructing order via Random / NN(k) / Grouped / Mixed(k) methods, and (3) optional result evaluation (`-eval`). In comparative experiments in this report we do not apply additional local improvement stages, so comparison depends solely on the described heuristics and parameters.

Experiment Setup

In the method matrix we compare three vertical photo pairing variants (random/similar/different) for fixed $k_{\text{pair}} = 50$, and in parameter tuning tests (sweep of k and k_{pair}) we use "different" pairing as baseline.

Results from Tab. 6, Fig. 7, and parameter tuning plots come from notebook runs (Random, NN(k), Grouped, Mixed(k)) and are consistent with implementations used in `solver.py`. Additional stages (e.g., local improvement) were disabled so interpretation depends solely on heuristics and parameters (k, k_{pair}).

Note on Random Seed. In the table and plots we use a fixed seed for comparability. Since NN/Mixed contain randomness (start and candidate sampling), results may differ between runs.

Method Comparison: Final Score

Table 6 and Fig. 7 show results for combinations: *(vertical photo pairing method)* × *(sequence building method)*. Conclusions are consistent with intuition:

merely improving **pairing V** (random → "different") improves result even with random order,

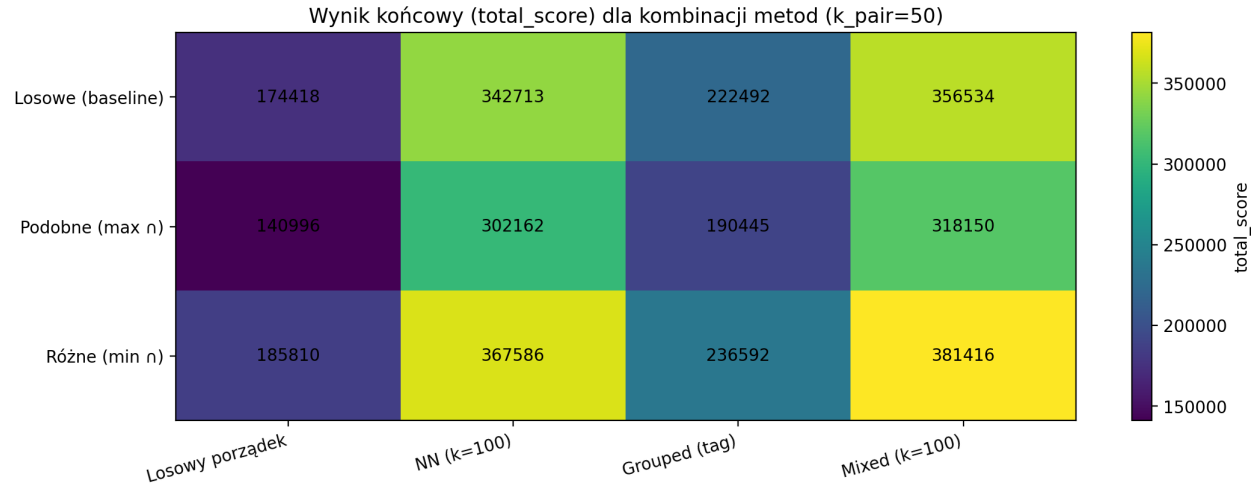
largest gain comes from switching from random order to methods using local transition evaluation (NN / Mixed),

grouping by tag improves result relative to random order, but clearly loses to NN and Mixed,

in our results the **Mixed method performs better than NN for all pairing variants** (e.g., for "different" pairing: 381416 vs 367586), suggesting that ordering slides in "blocks" helps, and local selection (NN) polishes transitions within those blocks.

Tabela 6: Final score (`total_score`) for method combinations (for $k_{\text{pair}} = 50$ and $k = 100$ in NN/Mixed); without local improvement.

Vertical photo pairing	Random order	NN ($k = 100$)	Grouped (tag)	Mixed ($k = 100$)
Random (baseline)	174418	342713	222492	356534
Similar (max ∩)	140996	302162	190445	318150
Different (min ∩)	185810	367586	236592	381416

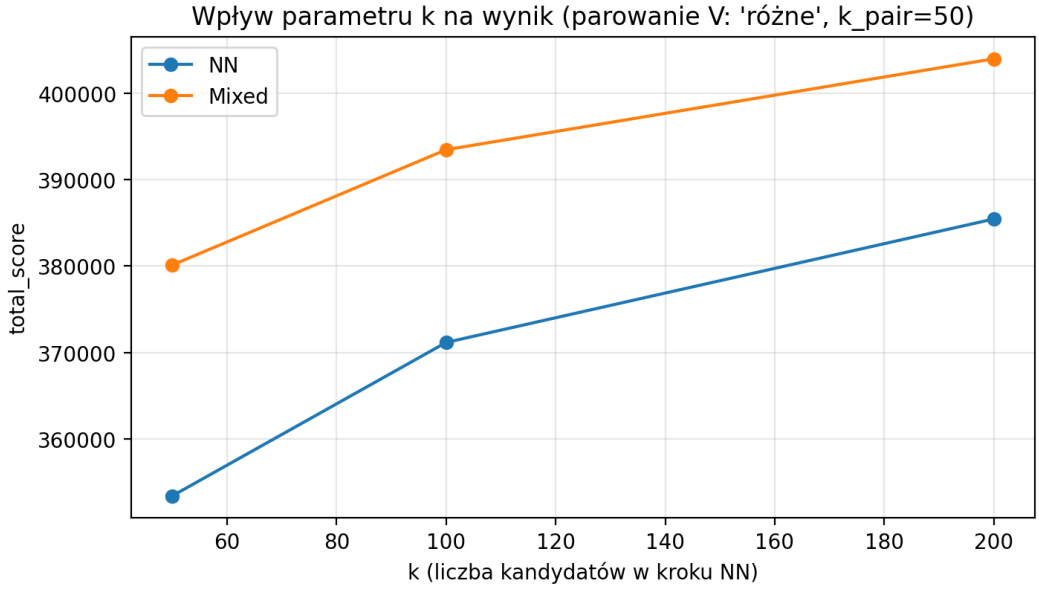


Rysunek 7: Final score (`total_score`) for method combinations (lighter = better). Parameters: $k_{\text{pair}} = 50$, $k = 100$ in NN and Mixed.

Parameter k Tuning (Nearest-Neighbor / Mixed)

In NN-based methods, the key parameter is k (number of candidates considered in each step of selecting the next slide). Larger k usually improves result because we rarely choose a "random" low-quality transition, but computation cost increases (approximately the number of comparisons grows linearly with k).

An additional sweep was performed for V pairing = "different" and $k_{\text{pair}} = 50$ (single run, fixed seed for reproducibility). Fig. 8 shows the **saturation effect**: moving from $k = 50$ to $k = 100$ gives a large gain, and increasing to $k = 200$ still improves result, but more weakly.

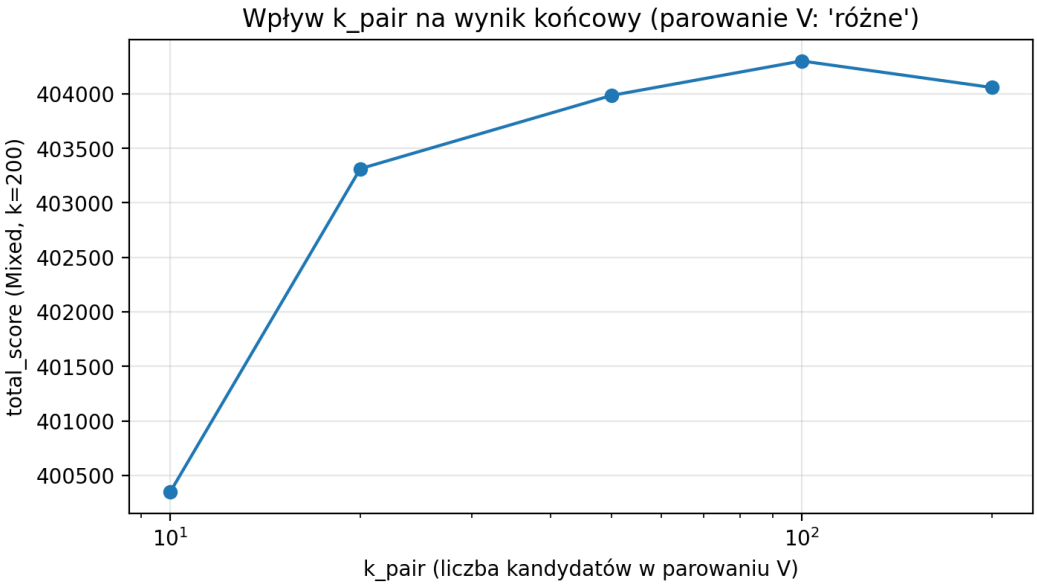


Rysunek 8: Impact of parameter k on result (vertical pairing: "different", $k_{\text{pair}} = 50$).

Note on Randomness. Since NN/Mixed sample candidate pools and depend on start selection, results may vary between runs. Parameter tuning plots were made for fixed seed (comparability), while in practice a simple way to improve result is to perform several runs with the same configuration and select the best outcome.

Impact of Parameter k_{pair} in Vertical Photo Pairing

Parameter k_{pair} (number of candidates considered when combining vertical photos) affects the "quality" of vertical slides, and thus indirectly the maximum possible sequence score. A sweep of k_{pair} was performed for "different" pairing with Mixed order building at $k = 200$. Fig. 9 suggests improvement stabilizes around $k_{\text{pair}} \approx 50$–$200$. Logarithmic x-axis makes it easier to compare increments for small and large k_{pair} values (saturation effect is visible). In this test we keep fixed: order method (Mixed), parameter $k = 200$, and one fixed random seed, so comparison depends solely on k_{pair}.



Rysunek 9: Impact of k_{pair} on final score (vertical pairing: "different", order: Mixed, $k = 200$). x-axis in logarithmic scale.

Global Optimum vs. Practical Solution

Due to the astronomical number of slide permutations, finding a global optimum is computationally infeasible. The applied heuristics construct solutions locally (greedily, over a limited candidate pool), so result depends on parameters and random elements (e.g., starting point). Randomness appears in the choice of starting slide / processing order and sampling a limited candidate pool, so a single run may yield different results even for identical parameters.

Actual Optimization Process: Local Solution Improvements

Constructive heuristics (vertical photo pairing and slide sequence building via `Mixed/NN/Grouped`) provide a good starting solution but do not guarantee optimum. Key stages are **local improvements**, which were selected based on the following method comparison.

Comparison of Local Optimization Methods

In the project (files `optimization.py` and notebook `slideshow_optimalization.ipynb`) we tested several classical local improvement procedures for a completed slideshow. Since the transition scoring function is symmetric, in 2-opt only boundary edges change: $(i-1, i)$ and $(j, j+1)$. In this code version delta for 2-opt is computed safely also for boundary cases.

Tested Methods

Hill climbing (HC) – random swaps of two slides with acceptance only if result improves.

2-opt – random reversal of segment $[i : j]$ (acceptance only if result improves).

Simulated annealing (SA) – random swaps with possible acceptance of worsening moves with probability dependent on temperature.

Results from Notebook

The table below shows an example run (one starting instance, settings: HC: 60k iterations, 2-opt: 60k iterations, SA: 60k iterations).

Method	Parameters	Score	Improvement
Start (after heuristic)	–	413757	0
Hill climbing	iters = 60000	413760	+3
2-opt	iters = 60000	413762	+3
Simulated annealing	iters = 60000	413758	+1

Conclusion

In the final solver we kept fast local improvement (HC with simple moves and short 2-opt), as it provides good quality/time ratio.

Neighborhood (Local Moves)

We optimize the slide permutation. We define neighborhood through three types of moves:

Swap neighbors $(i, i+1)$: swap two consecutive slides.

Random swap (i, j): swap two arbitrary positions.

Short 2-opt: select $i < j$ and reverse segment $[i, \ldots, j]$ (we limit length to maintain speed).

Algorithm: Hill Climbing

We use simple hill climbing: in each iteration we randomly select a move from the above neighborhood, compute quality change and **accept the move only if result improves**.

To speed up computation, we compute the result change only on edges affected by the move:

for swaps: at most 4 neighborhoods (edges at i and j),

for 2-opt: we compare only "entry" and "exit" edges of the reversed segment.

This makes each iteration cheap, allowing the process to be repeated many times.

Parameter and Stopping Criterion

We run the algorithm after building a starting solution. It is controlled by the parameter:

`local_iters` = number of local improvement iterations.

For `local_iters=0` improvements are disabled. Increasing `local_iters` usually improves result, but with diminishing increments.

Experiment: Impact of Iteration Count on Result

To show the **actual iterative optimization process**, we run the solver for the same constructive heuristic configuration (same method and same `seed 42`) and varying `local_iters` values. We compare result **before** and **after** local improvement.

`local_iters`	score before	score after
0	411623	411623
5	411623	412513
10	411623	414050
20	411623	414342
100	411623	414354
1000	411623	414635

Conclusion: local improvement provides positive quality gain, and gains decrease as iteration count increases (saturation), which is typical for local optimization.

Final Conclusions

1. **Vertical photo pairing has a real impact on result.** "Different" pairing generates semantically richer slides (small intersection, large union of tags), increasing transition point potential.
2. **Largest quality jump comes from moving away from randomness in sequence construction.** Heuristics using local transition evaluation (NN/Mixed) significantly improve results versus random order, and grouping by tag is an intermediate solution (better than random, but clearly worse than NN/Mixed).
3. **Parameters exhibit saturation effect.** Increasing k (in NN/Mixed) and k_{pair} improves result, but with diminishing returns, meaning growing computational cost for ever smaller quality improvements.

Best Tested Configuration

The highest result in our runs was achieved with a configuration consistent with notebook analysis: vertical pairing: "different" ($k_{\text{pair}} = 300$) and sequence order: Mixed ($k = 300$). With random seed 42 and local improvement (`local_iters`=1000) we achieved a score of 414 635 points.

```
python solver.py --data_dir ../data --out out.txt --seed 42 \
    --pairing different --order mixed --k 300 --k_group 10 --group_key first \
    --local_iters 1000 --eval
```