# Enhancing Path Efficiency: Innovations in the Artificial Feeding Birds Algorithm for Solving TSP Problems

Tim Cares
*Fakultät IV*
*Hochschule Hannover*
Hanover, Germany
tim.cares@stud.hs-hannover.de

Pit Simon Hüne
*Fakultät IV*
*Hochschule Hannover*
Hanover, Germany
pit-simon.huene@stud.hs-hannover.de

***Abstract*— The process of scientific writing is often tangled up with the intricacies of typesetting, leading to frustration and wasted time for researchers. In this paper, we introduce Typst, a new typesetting system designed specifically for scientific writing. Typst untangles the typesetting process, allowing researchers to compose papers faster. In a series of experiments we demonstrate that Typst offers several advantages, including faster document creation, simplified syntax, and increased ease-of-use.**

***Index terms*—Scientific writing, Typesetting, Document creation, Syntax**

## I. Introduction

## II. Basic AFB Algorithm

## III. Methodology

We structure our experiments in an incremental manner. Each improvement in Section IV builds on the previous adjustments, even if it is not explicitly noted. We verify our improvements using a set of selected TSP problems from TSPLIB, namely eil101, pa561, pr1002, u2156 and pr2392, ranging from 101 to 2392 cities.

For each experiment, we run the problems 10 times each to account for the randomness at initialization, to reduce noise, and make it statistically significant. In order to then compare the performance of different experiments we record the *Mean Percentage Error*, over the 50 tests, with respect to the optimal solutions. We also record the *Mean Runtime* in seconds.

$$\text{Percent Error} = \frac{|\text{measured} - \text{optimal}|}{\text{optimal}} \cdot 100 \qquad (1)$$

All experiments are executed on a MacBook Pro (M1 Pro) 2021.

## IV. Improvements

### A. Considerations about number of birds

If we study the effects of how different number of birds change the solution of different tours, we learn that less birds seem to yield better solutions. At first this may seem counterintuitive, but there is an intuitive explanation to this:

The basic algorithm defines one iteration as the calculation for the cost of one tour. Now, let a phase of the algorithm denote each bird performing one move. So, in each phase each bird performs one move. If we now use more birds the number of phases the algorithm will run through will decrease, as in each phase the length of more tours will be calculated, which will use up more iterations. Therefore, the more birds we have, the more iterations we will use up per phase, which means the algorithm will run through less phases until it stops. Because in one phase each bird can perform one move, fewer phases mean each bird can perform fewer moves. Less moves in turn leads to a less pronounced search of possible solutions, which will make the algorithm worse. This is why the fewer birds we have, the better the results will be. Examples can be seen in Figure … (fig about relationship n birds x cost).

One could avoid this by simply increasing the number of iterations, which balances the relationship between number of birds used and the results obtained. However, this inevitable leads to longer running times.

Consequently, we focus on improving the bird behavior, so that each bird needs fewer overall steps to achieve a good solution.

### B. Swarm Behavior

Currently, if one big bird made the decision to join another bird, he picks one randomly. This means joining any bird without considering how good the position of that bird might be. This contradicts the original idea of the authors that big birds tend to join others that have found a good food source (current solution seems promising) [1]. Therefore, we propose that a big will only be able to join the top-b percent of birds that have the lowest current cost. If one chooses the right ratio, we assume that it will automatically nudge the swarm in the direction of the global minimum.

We implement this by storing the indices $i$ of the birds in an ordered integer array ord and introducing a new hyperparameter $b$, denoting which of the top-b percent to join. When we select the bird to join to, we draw a random uniform number $j$ between 1 and $b \cdot n$ ($n$ denotes the number of birds) and get the index of the bird to join from the ordered array (ord[$j$]).

The main disadvantage this approach has is that at we need to continuously update ord so that we only join the actual top-b percent at the moment of the move. We decide to update the list after each bird has performed one move (after each phase). This will still increase the computational complexity, but we think that this is better than updating the list after each move, as this would be too costly.

We test numerous values for $b$ and decide to build on $b = 0.01$ for future improvement, as this strategy yields the best results (Table 1). For intuitions on why such a low value performs this good, please refer to Section V.A.

| Top-b | 1 | 0.25 | 0.20 | 0.15 | 0.05 | **0.01** |
|---|---|---|---|---|---|---|
| PercentError | 215 | 122 | 5.92 | 6.14 | 6.01 | **5.2** |
| Time (in s) | **7.6** | 8.6 | 8.7 | 8.1 | 8.3 | 8.1 |

Table 1: Comparison of different parameters for our top-join. 1 means 100%, so a bird randomly joins another bird. This is our benchmark and the default behaviour of the algorithm. Time is measured as the median runtime in seconds, over all 860 tests.

### C. 3-opt Walk

For the walk move the algorithm uses a modified version of the 2-opt heuristic as a local search [1]. Instead of using 2-opt, we test a 3-opt variant, as this often yields the best solutions under consideration of computational complexity [2]. At the same time we decide to remove the estimation of local similarity from the algorithm, as the authors did not provide any intuitions why this may be beneficial [1].

With 3-opt, each bird selects the tour with the lowest cost out of the 7 different tours possible. At which 3 points a tour is opened is determined by a random uniform draw of 3 integers, denoting the nodes of the tour. Because we now need to calculate the length of each of the possible 7 tours in just one move, we decide it will still only cost us one iteration. The results can be seen in Table 2.

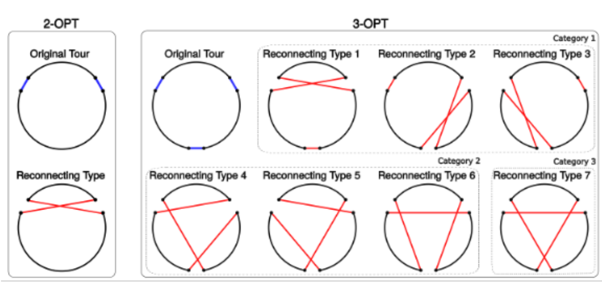


Figure 1: Comparison of 2-opt (left) and 3-opt (right) for TSP. Reconnections of category 1 are 2-opt variants [3].

| Configuration | 2-opt | 3-opt |
|---|---|---|
| PercentError | 5.2 | **4.67** |
| Time (in s) | **8.1** | 9.56 |

Table 2: Comparison of 2-opt and 3-opt. Time is measured as the median runtime in seconds, over all 860 tests.

### D. Delegating Responsibility

Because we are not implementing the 3-opt algorithm as an iterative solver by itself, but rather in a swarm algorithm, in which each bird (or agent respectively) can perform this action, the computational complexity will rise by a margin, as seen in Table 2. In order to now compensate for this increase in complexity, we decide to test if we can delegate the responsibility of performing 3-opt to only a subset of the birds. This sounds promising, as we can reduce the computational complexity while still being able to profit from the increased exploration of 3-opt.

We test the case where only big birds are able to perform a 3-opt walk, while smaller birds are only capable of the usual 2-opt walk as specified in the paper. This should not only reduce the computational complexity, but it also pairs well with the assumption that big birds are "superior", as only they can join other birds and therefore profit from them.

For the sake of completeness we also test the inverse approach: Only small birds can perform 3-opt. Surprisingly, for big birds we observed that we can achieve the same performance as before, while cutting runtime by half (Table 3). Interestingly, for small birds we noticed an increased error rate, even though the small bird ratio is always $r > 0.5$ in our experiments. Based on the results, we decided to use 3-opt only for big birds in our future experiments, as there seems to be no downside to this approach.

| Configuration | 2-opt | 3-opt | 3-opt small birds | 3-opt big birds |
|---|---|---|---|---|
| PercentError | 5.2 | 4.67 | 55% | **55%** |
| Time (in s) | **8.1** | 9.56 | 55 | 55 |

Table 3: Comparison of 2-opt and 3-opt. Time is measured as the median runtime in seconds, over all 860 tests.

### E. Nearest Neighbor Initialization

So far, we have always initialized the birds with a random tour. While this allows for a very wide exploration, it also means that the algorithm needs many iterations to reach the first competitive solutions. To speed up convergence, we decided to use a simple heuristic to generate initial tours. Initially, we tried to implement the Christofides algorithm, which is the best-known polynomial-time heuristic for solving the TSP and provides a $\frac{3}{2}$-approximation. However, due to the complexity of the algorithm and the relatively long worst-case runtime of $O(n^3)$ [4], we decided to use the nearest neighbor heuristic instead. This heuristic is very simple, has a worst-case runtime of $O(n^2)$ [5], and still provides very good solutions.

Adding the heuristic initialization had a negligible impact on the runtime, which makes sense since the initialization is done only once. However, the results improved dramatically from an average error of 44% with random initialization to now 8%.

### F. Early Stopping

If we analyze the convergence behavior by plotting the cost of the best solution over the number of iterations (Figure TODO), we notice that our improved version of the algorithm converges much faster than the original algorithm, even without a nearest neighbor initialization. Because it is difficult to estimate how many iterations are needed for a certain problem, and adapting the number of iterations to the problem at hand would be cumbersome, we decide to implement an early stopping mechanism. That way we do not waste computational resources on iterations that do not improve an existing solution, which will reduce the runtime even further, especially for smaller TSP configurations, while retaining a similar performance.

We implement this by introducing a new hyperparameter $p$, denoting the number of phases without an improvement of the best current solution. If it is exceeded, the algorithm will stop. This requires us to store the best solution over all birds and updating it continuously. Fittingly, this is already implemented through the top-b join (see Section IV.B), as we already need to store the best solution over all birds in order to determine which bird to join to. This is also the reason why we only check if the best solution has improved after each phase, and not after each iteration, as during a phase this solution is not updated. A review during a phase therefore does not make sense.

| EarlyStopping | No (default) | **Yes** |
|---|---|---|
| PercentError | **8** | 10 |
| Time (in s) | 42 | **9** |

Table 4:

### G. Metabirds

Both the original algorithm and our extensions define various hyperparameters like the number of birds, the ratio of small birds, the top-b join percentage, the probabilities of the basic moves We have already discussed analytical approaches to choosing some of these values, but these tests were limited and did not capture the possible complex interplay between different hyperparameters.

To remedy this, we chose to apply an optimization algorithm to the hyperparameters of the TSP solver. The choice of optimization algorithm was simple: we used the same AFB algorithm that the TSP solver itself uses. We call this algorithm *Metabirds*, as every bird in the hyperparameter optimizer contains itself a swarm of birds solving the TSP.

To be more precise, every metabird represents a position in the hyperparameter space made up of values for the number of birds, the ratio of small birds, the top-b join percentage and the various move probabilities. For the walk of a metabird, we sample random deltas from a normal distribution and apply them to the hyperparameters. Flying to a new position is done choosing random values for the hyperparameters.

The problem with these simple implementations is, that they can produce invalid values. Specifically,the following conditions must be met:

- the sum of the move probabilities cannot exceed one
- the top-b join percentage must be large enough to include at least one bird
- the number of birds cannot be negative.

Since these meta-moves do not contribute a lot to the overall runtime, we decided to solve this by simply sampling new configurations until we find one that is valid.

To evaluate the cost of a metabird, we run create a TSP solver with birds configuration, run it 8 times and average the results.

We ran the metabirds algorithm multiple times with different problems (eil101, d493, dsj1000, fnl4461) to optimize for different TSP sizes. To make these runs, which took multiple days, feasible, the algorithm was compiled to native code using GraalVM and executed on cloud resources.

## V. Analysis

### A. Intuitions on our Improvements

Swarm algorithm usually include action of agents which can either be classified as exploitation or exploration. Exploitation means an agent uses its current result and tries to improve it, i.e. the agent continues to go into the direction he previously went in the search tree/space. For AFB, this is can be achieved using the walk move, so a local search.

Exploration means an agent tries to find a better solution not necessarily dependent on its current solution, as is the case with exploitation. Therefore, it can be seen as a global search. For AFB, this can be achieved using the fly move.

For a swarm algorithm to deliver a good approximation with respect to the global minimum it needs a good balance between exploitation and exploration: It needs to be able to improve a good solution, and search for different solutions if the current one does not seem promising.

If exploitation is too dominant, then the algorithm might get stuck in a local minimum, as other solutions (for TSP other, vastly different tours) will not be explored enough, and vice versa.

As might have come apparent in the sections prior, the main focus of this paper was on exploitation (improving a good solution). This is mainly done by the introduction of 3-opt and that big birds can only join successful birds. Even though the latter cannot be seen directly as a local search, is does lead to more (big) birds performing exploitation of the solutions of other birds.

We explicitly do not modify the fly move, i.e. selecting a random tour, as this provides us with a rich selection of other possible solution, which at the same time is completely independent of the current solution (of an agent).

The prior is crucial for the success of our algorithm, because the extreme join behavior modeled by the algorithm has a high danger of getting it stuck in a local minimum: Starting with our improvements on the swarm behavior, a big bird will only be able to join the best bird. Because which bird is the best is not updated within a phase, it could be that during one phase a lot of big birds join the same bird, putting a lot (not all, as a join is a matter of probability) of agents at one place in the search space. The fly move enables birds to escape this single solution if it does not seem promising.

Furthermore, there is only a limited number of big birds, meaning small birds are able to explore the search space elsewhere, while the big birds are focusing on the currently best performing bird.

Therefore, we get an algorithm that has an empirically tested balance between exploitation and exploration. It exploits good solutions in a harsh manner while also being able to switch to completely new solutions if they prove to be better. This process will be repeated until a solution is reached that will either be close to the optimum, or a local minimum. Either way, the algorithm converges.

### B. Exploitators and Explorators

The base algorithm consists of two types of agents, small and big birds. With our improvements it may have been noticeable that we separated the roles of both agent types more and more from each other: While small birds can perform the usual 2-opt local search when they are walking, big birds can perform a more powerful 3-opt walk; big birds can join other birds. We do this in order to make the components contained in each swarm algorithm, exploration and exploitation, a more explicit part of the algorithm: We delegate small birds to the role of explorators, and big birds to the role of exploitators.

Small birds are able to access vastly different areas of the search space for possible better solutions than their current one. Using the fly-move, big birds are able to profit from those birds that have found the best current solution by joining them and improving that solution using 3-opt (walk).

The circumstance that small birds can also perform exploitation, using their own version of the walk move, is owed to the fact that they otherwise would only be able to perform the fly move, i.e. jumping between random solutions. This wouldn't be a good foundation for the join behavior of big birds (see Table 5), which is essential for the performance of our algorithm. Also, since big birds can also join other big birds, and the solutions for small birds would be rather poor, the probability that big birds will exclusively join other big birds would be very high, making small birds essentially useless.

Exactly this can be verified by simply comparing how the algorithm performs when (1) small birds can only fly, (2) all small birds are removed from the algorithm, and only big birds are kept.

Surprisingly, the results show us that configuration (2) performs even better than variant (1), indicating that in (1) the big birds only join other big birds, and that small birds, whose only purpose is to perform the fly move (so not even returning to their best solution), provide no value to the algorithm. This is exactly why we decided that small birds are also able to perform the walk move.

| Configuration | **Regular** | Only fly | No small birds |
|---|---|---|---|
| PercentError | **8** | 15 | 10 |

Table 5: If small birds are only able to fly, the algorithm performs worse than before. Notice however that it still achieves a reasonable performance. For our experiments we continuously used 200 birds, 150 of them being small birds. So by removing all small birds for experiment (2), we are left with 50 (big) birds.

## References


[1] J.-B. Lamy, "Artificial Feeding Birds (AFB): a new metaheuristic inspired by the behavior of pigeons", *Advances in nature-inspired computing and applications*, pp. 43–60, 2019.

[2] S. Lin, "Computer solutions of the traveling salesman problem", *Bell System Technical Journal*, no. 10, pp. 2245–2269, 1965.

[3] J. Sui, S. Ding, R. Liu, L. Xu, and D. Bu, "Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning", 2021, pp. 1301–1316.

[4] R. van Bevern and V. A. Slugina, "A historical note on the 3/2-approximation algorithm for the metric traveling salesman problem", *CoRR*, 2020.

[5] M. M. Flood, "The traveling-salesman problem", *Operations research*, no. 1, pp. 61–75, 1956.