Solving Job Shop Scheduling Problems Without Using a Bias for Good Solutions

Thomas Weise
tweise@hfuu.edu.cn
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

Xinlu Li
xinlu.li@vip.163.com
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

Yan Chen
chenyan@hfuu.edu.cn
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

Zhize Wu*
wuzhize@mail.ustc.edu.cn
Institute of Applied Optimization, School of Artificial Intelligence and Big Data, Hefei University
Hefei, Anhui, China

ABSTRACT

The most basic concept of (meta-)heuristic optimization is to prefer better solutions over worse ones. Algorithms utilizing Frequency Fitness Assignment (FFA) break with this idea and instead move towards solutions whose objective value has been encountered less often so far. We investigate whether this approach can be applied to solve the classical Job Shop Scheduling Problem (JSSP) by plugging FFA into the (1+1)-EA, i.e., the most basic local search. As representation, we use permutations with repetitions. Within the budget chosen in our experiments, the resulting (1+1)-FEA can obtain better solutions in average on the Fisher-Thompson, Lawrence, Applegate-Cook, Storer-Wu-Vaccari, and Yamada-Nakano benchmark sets, while performing worse on the larger Taillard and Demirkol-Mehta-Uzsoy benchmarks. We find that while the simple local search with FFA does not outperform the pure algorithm, it can deliver surprisingly good results, especially since it is not directly biased towards searching for them.

CCS CONCEPTS

• **Theory of computation** → **Random search heuristics**; **Theory of randomized search heuristics**; • **Applied computing** → **Operations research**; • **Mathematics of computing** → *Combinatorial algorithms*; **Permutations and combinations**; **Combinatoric problems**; • **Computing methodologies** → **Planning and scheduling**.

ACM Reference Format:
Thomas Weise, Xinlu Li, Yan Chen, and Zhize Wu. 2021. Solving Job Shop Scheduling Problems Without Using a Bias for Good Solutions. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3449726.3463124

*Corresponding author.

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463124

1 INTRODUCTION

The Job Shop Scheduling Problem (JSSP) [8, 23] is one of the most prominent and well-studied scheduling tasks. In a JSSP instance, there are m machines and n jobs. Each job must be processed once by each machine in a job-specific sequence and has a job-specific processing time on each machine. The goal is to find an assignment of jobs to machines that results in an overall shortest makespan, i.e., the schedule which can complete all the jobs in the shortest time.

The JSSP is $\mathcal{NP}$-hard [9, 23]. This means that solving JSSP instances to guaranteed optimality may not be feasible in practical applications. Reaching the optimal makespans may often take too long in real-world scenarios. Instead, JSSPs are often approached heuristically, by algorithms that try to find good approximate solutions within an acceptably short time. While heuristics cannot guarantee the optimality of their results, the comprehensive meta-studies in [29, 32, 34] show that quite a few of the commonly used JSSP benchmark instances can be solved to optimality by the state-of-the-art heuristics.

The most common method for heuristically solving the JSSP is to adapt local searches or other metaheuristics such as Simulated Annealing (SA) [22], Tabu Search (TS) [17], and Evolutionary Algorithms (EAs) [6]. These algorithms generate a set of initial solutions and then attempt to refine them. Usually, in each iteration, they derive one or multiple new points in the search space from the set of current solutions. They then select the solutions for the next iteration from the joint set of current and new candidates. All of these algorithms have in common that they select solutions with better objective values with higher probability. This is the most basic principle upon which all metaheuristics are built. While many algorithms such as SA, TS, and EAs complement it with diversity preservation or generation measures, the only traditional algorithms completely without this bias are random walks, random sampling, and exhaustive enumeration – and they are not considered as efficient approaches to the JSSP.

In [36], it was shown that several optimization problems can be solved efficiently even *without* the bias towards better solutions. It

was uncovered that Frequency Fitness Assignment (FFA), proposed in [35], has this property and also renders the optimization process invariant under any bijective transformation of the objective function value. Invariance properties are generally beneficial for optimization [19, 37], as they generalize the results on one problem to a class of problems [27]. While we describe the concept of FFA in more detail in Section 2, in a nutshell, this behavior is achieved by preferring solutions not with *better* objective values, but with *less frequently encountered* objective values. In [36], it was further found that FFA makes a simple local search slower on easy problems but can speed it up significantly on the $\mathcal{NP}$-hard [15] Max-Sat problem. There, FFA also improved the performance of a Memetic Algorithm (MA) for the JSSP, into which it was inserted only as diversity preservation mechanism. That MA, however, still used a conventional local search, i.e., was biased towards better solutions and thus not invariant under bijections of the objective function value.

In this paper, we investigate whether an optimization process based only on FFA can yield good results on the JSSP. Similar to what was done in [36] for the problems defined over binary search spaces, we plug FFA into the simplest local search possible, the (1+1)-EA and obtain the (1+1)-FEA. As representation, permutations with repetitions are used, which will later be discussed in detail and illustrated in Figure 2. We find that, while the (1+1)-FEA cannot outperform (1+1)-EA within the computational budget of our experiments, it still delivers surprisingly good results which are less than 1.5% worse in average over 242 common JSSP benchmark instances. Moreover, its average results are even better than those of the (1+1)-EA on six of the eight benchmark sets used and it can also outperform several recently published algorithms.

This finding is significant: On one hand, it shows that, besides the Max-Sat, there exists at least one other $\mathcal{NP}$-hard problem on which an FFA-based algorithm can outperform its pure variant. On the other hand, it indicates that the JSSP has an underlying structure allowing for a search towards "hard-to-find" objective values to also yield good solutions. This discovery opens up a new pathway to tackle the JSSP: We now know that a search for solutions with unseen characteristics can guide the optimization process towards better results and that this approach is not necessarily worse than directly searching for better solutions. In other words, in this paper we show that a search paradigm different from every non-trivial approach that was ever applied to the JSSP can perform surprisingly well.

The rest of this paper is organized as follows. In Section 2, we discuss FFA and the algorithms used in this study. The experimental setup and results are given in Section 3. Finally, Section 4 concludes the paper with a summary and outlook on future work.

2 ALGORITHMS STUDIED

Assume that we are solving an optimization problem with a given space $\mathcal{X}$ of possible candidate solutions and an objective function $f : \mathcal{X} \mapsto \mathcal{Y}$. Further assume that f be subject to minimization and can take on *integer* values from the interval from 0 to a given upper bound UB, i.e., $\mathcal{Y} \subseteq 0..UB$. This is the case for the JSSP, but also for many other classical problems such as Max-Sat.

The (1+1)-EA is a very simple local search which starts with a randomly generated solution $x_c \in \mathcal{X}$. In each step, it applies the (randomized) unary search operator *move* to x_c to derive a new candidate solution $x_n \in \mathcal{X}$. If the objective value $y_n = f(x_n)$ of x_n is not worse than the quality y_c of x_c, then x_n is selected, i.e., replaces x_c (otherwise it is discarded). This algorithm is illustrated in Figure 1(a).

FFA is implemented as a fitness assignment process, i.e., an algorithm phase taking place before the selection step. As such, it can be plugged into almost arbitrary optimization methods. In FFA, the fitness corresponding to an objective value is its absolute encounter frequency so far in fitness assignment steps and it is subject to minimization.

In Figure 1(b), we plug FFA into the (1+1)-EA and obtain the (1+1)-FEA. For this, the map H for counting the encounter frequency of each objective value y during the search is needed as additional data structure. The (1+1)-FEA starts exactly like the (1+1)-EA and also maintains a "current solution" x_c (with objective value y_c) from which a new solution x_n is derived in each step. After computing the objective value y_n of x_n, the encounter frequencies $H[y_c]$ and $H[y_n]$ of both solutions are incremented. They are always positive integer numbers. Instead of selecting x_n if its objective value y_n is not larger than y_c, it is selected if the *encounter frequency* $H[y_n]$ of y_n is not larger than $H[y_c]$. Whether x_n is better or worse than x_c does not matter. By using the objective values as indices into H, they are only compared for equality and their order plays no role. This means that our (1+1)-FEA would follow an identical path in the search space even if we would apply *any* bijection g to f and optimize $g(f(x))$ instead of $f(x)$.

While the objective values of the solutions remain constant during the search (e.g., the makespan of a specific schedule in the JSSP always remains the same), the frequency fitness values change. From the perspective of the algorithm, FFA thus turns a static optimization problem into a dynamic one where schedules with previously unseen makespans appear as temporary optima but successively get worse the more often their objective values are encountered. As a result, a local optimum in the search space can be an optimum under FFA when discovered for the first time, but its basin of attraction will be "filled" over time and the search will eventually depart from it. This also means that worse solutions could be selected into x_c and we need to keep track on the best-so-far solution in a variable x_b. This variable does not have any impact on the direction of the search and is only used as final return value.

We now choose the search space $\mathcal{X}$, objective function f, and search operator *move* for the JSSP. The classical JSSP does not permit preemption, each machine can process either exactly one job or be idle, and the operations of each job must be performed strictly in the right order and cannot be parallelized. The objective function f is the makespan, the time when all jobs are completed. A trivial upper bound UB for f is the sum of the processing times of all jobs for all machines.

We use permutations with repetition as search space $\mathcal{X}$, i.e., integer strings where each of the n job IDs occurs exactly m times [7, 16, 28]. To manifest such a solution x as Gantt chart or to compute its makespan $f(x)$, it is processed from front to end. This is illustrated in Figure 2. When encountering job i, we know to which machine j it needs to go next based on the given job-specific machine sequence

```
1:  proc (1+1)-EA(f : X ↦ 0..UB)
2:
3:      randomly sample x_c from X; y_c ← f(x_c);
4:
5:      while ¬ terminate do
6:          x_n ← move(x_c); y_n ← f(x_n);
7:
8:
9:          if y_n ≤ y_c then x_c ← x_n; y_c ← y_n;
10:     return (x_c, y_c);
```

(a) (1+1)-EA

```
1:  proc (1+1)-FEA(f : X ↦ 0..UB)
2:      H[0..UB] ← (0, 0, ···, 0);
3:      randomly sample x_c from X; y_c ← f(x_c);
4:      x_b ← x_c; y_b ← y_c;
5:      while ¬ terminate do
6:          x_n ← move(x_c); y_n ← f(y_c);
7:          if y_n < y_b then x_b ← x_n; y_b ← y_n;
8:          H[y_c] ← H[y_c] + 1; H[y_n] ← H[y_n] + 1;
9:          if H[y_n] ≤ H[y_c] then x_c ← x_n; y_n ← y_c;
10:     return (x_b, y_b);
```

(b) (1+1)-FEA

Figure 1: The pseudo code of the investigated algorithms for the JSSP: the (1+1)-EA and the (1+1)-FEA with FFA.

textual representation of JSSP instance

```
4 5
0 10 1 20 2 20 3 40 4 10    job 0
1 20 0 10 3 30 2 50 4 30    job 1
2 30 1 20 4 12 3 40 0 10    job 2
4 50 3 30 2 15 0 20 1 15    job 3
```

This demo instance has 4 jobs and 5 machines, as written in the second row. Each of the following 4 rows contains 5 tuples of "(machine, time)" pairs. Each pair holds the index of the machine to which the job need to go and the time that it will spend on the machine. Each job must pass through the machines in that order.

example point in the search space: permutation with repetitions:

x=(0, 2, 1, 0, 3, 1, 0, 1, 2, 3, 2, 1, 1, 2, 3, 0, 2, 0, 3, 3)

The point can be transformed to the Gantt chart below by processing it from front to end. First comes job 0 and from the instance data we know that it needs to go to machine 0 for 10 time units. Then comes job 2 and we know from the instance data that it needs to go to machine 2 for 30 time units. Then comes job 1, which needs to go to machine 1 for 20 time units. Then, job 0 appears again, which now needs to go to machine 1. Before it can start there, it needs to wait until job 1 is finished there. And so on.

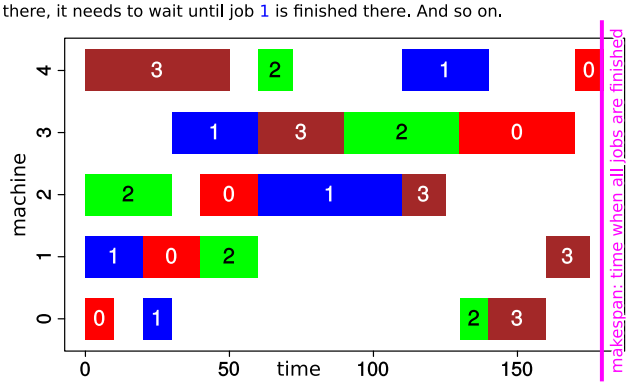

Figure 2: The process of translating a point x from the search space $\mathcal{X}$ to a Gantt chart based on a simple demo JSSP instance.

and on how often we already saw i in x before. We can start it on j at a time which is the maximum of *1)* when the previous operation assigned to j will finish and *2)* when the previous operation of i completes on its corresponding machine.

As search move $move(x)$, we apply a "1-swap" operation randomly picking two indices at which different job IDs are located and exchanging these IDs.

A similar encoding has also been employed in [36], but there a Memetic Algorithm was used, which only applied FFA in the selection step of the global search and not in the local search. That algorithm therefore still was strongly biased towards better solutions, whereas we here explore the fully bijection-invariant, FFA-based optimization.

Since there are $n!$ possible ways to arrange the n jobs on each of the m machines, there can be at most $(n!)^m$ different feasible job-machine assignments for a JSSP instance. However, the size of our search space $\mathcal{X}$ is $\frac{(m*n)!}{(m!)^n}$ [28], which is (exponentially) larger for $n > 1, m > 1$. This means that many different solutions would map to the same schedules and yield the same Gantt charts. In [36], it was found that a (1+1)-FEA does not provide any advantage over the (1+1)-EA on Plateau functions [3], i.e., discrete optimization problems with a large degree of neutrality in the search space. It will therefore be interesting to see whether the (1+1)-FEA can perform well on the JSSP, given that the encoding chosen exhibits much neutrality.

3 EXPERIMENTAL RESULTS

We now apply both algorithms to 242 common benchmark instances, namely the sets abz* [2], dmu* [11], ft* [12], la* [24], orb* [4], swv* [30], ta* [31], and yn* [38]. We conduct five runs of every algorithm-instance combination with a maximum budget of 2^{30} objective function evaluations (FEs), i.e., $1'073'741'824 \approx 10^9$ FEs per run. In Tables 1 to 5, we present information both about the instances as well as the results of our study. Column "*inst*" holds the instance ID. "*BKS*" is the best known solution, taken from [34] and marked with + if it is optimal. Finally, in "$m \times n$," the number of machines and jobs is given.

For each algorithm-instance pair, we provide the *best* makespan reached in any of the five runs, the arithmetic *mean* of the end quality over all runs, and the mean number *conv* of FEs until the runs could no longer improve their solutions (within the budget).

For each instance set, we provide summary statistics. To get an impression about the relative performance, on each instance, we divide the value of each of the three performance indicators

for the (1+1)-FEA by the corresponding value for the (1+1)-EA. In [13], it is recommended to use geometric means to summarize such normalized statistics. Therefore, the geometric mean of these values minus 1 is presented in row "*vs.* (1+1)-EA". If this value is negative, it means that the (1+1)-FEA yields a smaller indicator value in average, if it is positive, the (1+1)-EA has the smaller value. Since the differences between the result qualities are small, we present them in percent (%) units for "*best*" and "*mean*," while leaving them unscaled for "*conv*." We also count how often each algorithm has reached the smallest indicator values (marked in **bold** in each row) in row "*#best*."

The end result quality delivered by (1+1)-FEA is better in average on the abz*, ft*, la*, orb*, and yn4* instance sets, both in terms of *best* and *mean*. On swv*, the average for *mean* is better for (1+1)-FEA, while (1+1)-EA has a slight lead in *best*. The (1+1)-EA performs better on the dmu* and ta* instances. Since these two sets are larger (holding 160 out of the 242 instances), the (1+1)-EA comes out ahead in the overall averages, but with no more than a 1.5% advantage.

From the tables, we can also immediately see that the (1+1)-FEA usually has improvements later in the runs than the (1+1)-EA. From the overall summary at the bottom of Table 5, we find that it makes its last improvement after approximately $12 + 1 = 13$ times as many FEs as the (1+1)-EA in average.

In Figure 3, we plot the best-so-far solution quality over time in each of the five runs of the (1+1)-FEA and (1+1)-EA for six JSSP instances. These have been selected based on the maximum and minimum ratio (minus 1) of the three performance metrics that were also used in the result tables: the *best* solution quality reached by any run of the setup, the *mean* solution quality over all runs, and the mean FE index *conv* of the last improvement. This selection criterion is fair and provides a good impression of the different possible algorithm behaviors.

The highest advantage in terms of the best schedule discovered in any run on an instance for the (1+1)-EA over the (1+1)-FEA was observed on dmu80, illustrated in the top-left corner of Figure 3. Here, the (1+1)-FEA has a 15% higher (worse) makespan than the best schedule of the (1+1)-EA. The largest (1+1)-FEA-lead in terms of this statistic happened on swv09, where it delivered a best makespan that was 4.7% shorter (bottom-left chart). From the right-most column, we see that the (1+1)-EA may at most converge about 3400 times faster than the (1+1)-FEA, which never stops improving more than 50% earlier than the (1+1)-EA. The latter was observed due to a run of (1+1)-EA making one very late improvement on orb02.

Whether smaller or larger times to convergence are bad is not that clear: The (1+1)-FEA is certainly reaching its best solution later in the runs. However, it also seems possible that it may reach even better makespans if a larger budget was available, which is unlikely the case for the (1+1)-EA. Since the *BKS* is reached on 36% and 26% of the instances by the (1+1)-EA and (1+1)-FEA, respectively, there is such room for improvement. This potential is clearly visible in the charts where the (1+1)-FEA performed the worst in comparison, namely dmu80 and dmu79. There, the (1+1)-EA clearly can no longer improve tangibly near the end of the runs, whereas the (1+1)-FEA is still in a phase of steady progress.

Table 1: Results on the JSSP, part 1.

instance information			(1+1)-EA			(1+1)-FEA		
inst	*BKS*	*m*×*n*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
abz5	1234+	10×10	1239	1245	**1.4E6**	**1234**	**1234**	1.3E8
abz6	943+	10×10	**943**	954	**8.6E4**	**943**	**943**	4.7E7
abz7	656+	15×20	669	674	**1.5E8**	**665**	**672**	6.5E8
abz8	665	15×20	678	692	**1.0E8**	**674**	**677**	6.6E8
abz9	678+	15×20	690	703	**1.3E8**	**689**	**692**	6.0E8
		#best	1	0	5	5	5	0
			vs. (1+1)-EA on abz*			-0.3%	-1.2%	22
dmu01	2563	15×20	2641	2684	**1.4E8**	**2592**	**2615**	6.1E8
dmu02	2706	15×20	2762	2799	**1.8E8**	**2746**	**2763**	6.9E8
dmu03	2731+	15×20	2816	2853	**2.9E8**	**2814**	**2827**	5.9E8
dmu04	2669	15×20	2730	2749	**1.8E8**	**2683**	**2716**	5.5E8
dmu05	2749+	15×20	2853	2875	**4.4E7**	**2803**	**2814**	7.7E8
dmu06	3244	20×20	**3276**	3332	**2.4E8**	3289	**3310**	9.1E8
dmu07	3046	20×20	3112	3171	**8.4E7**	**3103**	**3124**	7.2E8
dmu08	3188	20×20	3261	3287	**1.5E8**	**3257**	**3263**	5.1E8
dmu09	3092	20×20	3176	3242	**7.2E7**	**3164**	**3178**	7.3E8
dmu10	2984	20×20	**3006**	3086	**2.2E8**	3019	**3056**	7.5E8
dmu11	3430	15×30	**3502**	**3544**	**5.0E8**	3579	3613	7.9E8
dmu12	3492	15×30	**3542**	**3558**	**1.9E8**	3642	3673	8.6E8
dmu13	3681+	15×30	**3718**	**3750**	**4.6E8**	3843	3855	8.1E8
dmu14	3394+	15×30	**3397**	**3408**	**2.7E8**	3443	3484	7.6E8
dmu15	3343+	15×30	**3343**	**3350**	**2.6E8**	3386	3397	7.9E8
dmu16	3751	20×30	**3823**	**3836**	**4.6E8**	3918	3941	8.9E8
dmu17	3814	20×30	**3907**	**3954**	**3.7E8**	4067	4085	7.8E8
dmu18	3844+	20×30	**3867**	**3936**	**3.5E8**	4055	4067	8.5E8
dmu19	3765	20×30	**3876**	**3916**	**3.6E8**	3960	4003	7.0E8
dmu20	3710	20×30	**3792**	**3810**	**6.0E8**	3904	3945	8.4E8
dmu21	4380+	15×40	**4380**	**4380**	**1.5E7**	4442	4451	9.5E8
dmu22	4725+	15×40	**4725**	**4725**	**7.1E7**	4738	4740	8.4E8
dmu23	4668+	15×40	**4668**	**4671**	**1.7E7**	4700	4724	8.4E8
dmu24	4648+	15×40	**4648**	**4648**	**3.0E7**	4669	4669	8.1E8
dmu25	4164+	15×40	**4164**	**4164**	**5.2E5**	**4164**	**4164**	6.4E8
dmu26	4647+	20×40	**4725**	**4767**	**5.3E8**	4971	5014	9.3E8
dmu27	4848+	20×40	**4848**	**4848**	**1.9E8**	5201	5210	8.5E8
dmu28	4692+	20×40	**4692**	**4708**	**1.2E8**	4946	4972	8.7E8
dmu29	4691+	20×40	**4691**	**4719**	**4.4E8**	4915	4973	9.6E8
dmu30	4732+	20×40	**4749**	**4775**	**2.3E8**	4976	5030	9.1E8
dmu31	5640+	15×50	**5640**	**5640**	**1.1E6**	5734	5755	8.6E8
dmu32	5927+	15×50	**5927**	**5927**	**3.7E4**	**5927**	**5927**	3.0E7
dmu33	5728+	15×50	**5728**	**5728**	**1.6E5**	**5728**	**5728**	2.9E8
dmu34	5385+	15×50	**5385**	**5385**	**4.8E5**	**5385**	5394	8.9E8
dmu35	5635+	15×50	**5635**	**5635**	**2.0E5**	**5635**	5640	6.7E8
dmu36	5621+	20×50	**5621**	**5621**	**7.2E7**	5996	6046	8.8E8
dmu37	5851+	20×50	**5851**	**5851**	**6.7E7**	6191	6236	9.0E8
dmu38	5713+	20×50	**5713**	**5713**	**9.0E7**	6115	6162	9.3E8
dmu39	5747+	20×50	**5747**	**5747**	**1.8E7**	6025	6043	9.8E8
dmu40	5577+	20×50	**5577**	**5577**	**1.2E7**	5939	5985	8.5E8
dmu41	3248	15×20	**3373**	3428	**2.1E8**	3383	**3405**	6.8E8
dmu42	3390	15×20	3525	3586	**7.0E7**	**3455**	**3507**	4.5E8
dmu43	3441	15×20	3662	3704	**8.1E7**	**3559**	**3585**	8.1E8
dmu44	3475	15×20	3652	3710	**6.9E7**	**3589**	**3618**	7.9E8
dmu45	3272	15×20	**3368**	3454	**3.1E8**	3387	**3396**	6.4E8
dmu46	4035	20×20	**4194**	4274	**4.0E8**	4196	**4212**	6.7E8
dmu47	3939	20×20	4131	4200	**9.4E7**	**4084**	**4122**	7.8E8
dmu48	3763	20×20	3957	4028	**3.1E8**	**3941**	**3955**	6.4E8
dmu49	3710	20×20	3924	3953	**5.1E8**	**3839**	**3876**	7.7E8
dmu50	3729	20×20	3958	4032	**5.3E8**	**3851**	**3901**	7.7E8

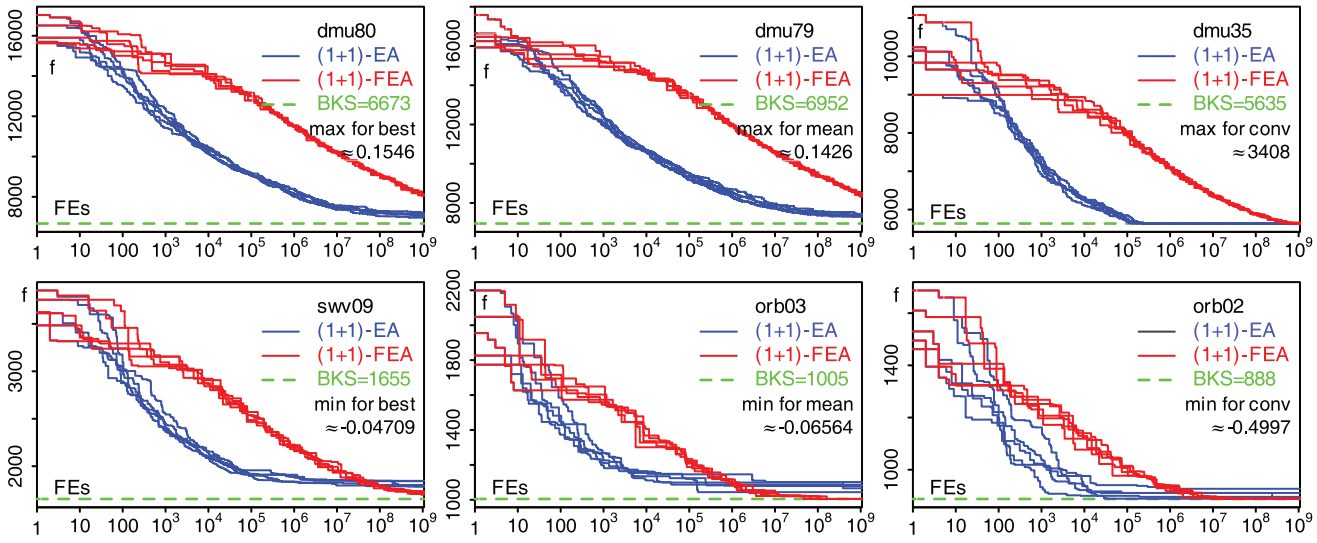

Figure 3: Progress of the best-so-far result quality on instances with the extremes of the observed performance indicator ratios (1+1)-FEA to (1+1)-EA over log-scaled FE axes: ratios minus 1 of best (left) and mean (middle) end quality as well as the FEs to convergence (right).

Let us now compare the result qualities reached by the (1+1)-FEA with the related work. Most of the literature use vastly different termination criteria, sometimes based on iterations, sometimes based on runtime. Since we use rather large budgets, we can expect that they may conduct less FEs. However, an algorithm using only FFA does not perform any optimization targeted towards better results. As shown in [36], such an algorithm may spend a good share of the runtime to follow trails over iteratively worsening solutions before "turning around" and then progressing towards better results. The search direction may change several times. Together with the fact that we have plugged FFA into the most primitive local search we can imagine and use a trivial search operator spanning a relatively restricted neighborhood, one may ask whether the (1+1)-FEA can yield any result quality even remotely comparable to what algorithms published in the last decade can achieve. We find that the mean and best results of the (1+1)-FEA, on every single instance, are at least as good and sometimes better than *all* corresponding mean and best results of all algorithms contributed in [1] (2010), of the aLSGA in [5] (2015), the EAS in [14] (2013), of all four GAs in [21] (2014), of the GWO method proposed in [20] (2018), of the SAFA in [25] (2018), of the HBFO in [26] (2012), and of all three algorithms in [33] (2018). These works report results only for fewer instances than used in our experiment, so we can only compare with the results actually reported (which can be found in the corresponding publications and in the online repository [34]). While, again, the termination criteria used in these works are different, the fact that an algorithm both as trivial and unbiased like the (1+1)-FEA can achieve results in this range is astonishing.

4 CONCLUSIONS

Frequency Fitness Assignment (FFA) translates objective values to their encounter frequency so far in the optimization process. These frequencies replace the objective values in the decision which solution to accept. FFA can therefore be plugged into almost arbitrary optimization processes. Since it does not make any order-based comparison of objective values, the path that an FFA-based algorithm takes through the search space is invariant under all bijections of the objective function value – a feature only known of trivial algorithms like random sampling, random walks, and exhaustive enumeration.

In this work, we plugged FFA into the (1+1)-EA, a simple local search, and obtained the (1+1)-FEA. In the algorithms, we use permutations with repetitions as representation. We applied them to the JSSP and found that the (1+1)-FEA can outperform the (1+1)-EA on several benchmark instance sets in terms of end result quality, while losing on others (under our budget constraints). This finding is very interesting, as FFA is free of any direct bias towards better solutions. It is biased only towards rarely-seen objective values. Yet, the (1+1)-FEA performs surprisingly well.

Our results also complement those from [36] and mark the JSSP as the second $\mathcal{NP}$-hard problem where driving the optimization process towards rarely-seen objective values alone can yield good results. In many discrete domains, good solutions are indeed rare and the better the solutions get, the rarer they are. FFA seems to be one of the few approaches to optimization which can exploit this as underlying concept driving the search.[1]

One important limitation of FFA that should be mentioned again is that it can only work if there are not too many different possible objective values. In its form defined here, it would therefore not be suitable to continuous optimization and probably also not for domains such as the Traveling Salesman Problems where the number of different solution qualities for an instance is very high. In

[1] Overviews on related works can be found in [10, 18, 36].

Table 2: Results on the JSSP, part 2.

instance information			(1+1)-EA			(1+1)-FEA		
inst	*BKS*	*m*×*n*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
dmu51	4156	15×30	**4406**	**4468**	**4.5E8**	4513	4550	7.9E8
dmu52	4311	15×30	**4550**	**4642**	**2.9E8**	4652	4687	9.0E8
dmu53	4390	15×30	**4614**	**4665**	**5.4E8**	4720	4744	8.4E8
dmu54	4362	15×30	**4580**	**4649**	**4.1E8**	4709	4747	8.6E8
dmu55	4270	15×30	**4436**	**4490**	**7.6E8**	4588	4629	8.8E8
dmu56	4941	20×30	**5227**	**5289**	**6.3E8**	5415	5484	8.2E8
dmu57	4663	20×30	**5044**	**5090**	**6.2E8**	5152	5181	8.7E8
dmu58	4708	20×30	**5002**	**5054**	**5.1E8**	5208	5223	7.9E8
dmu59	4619	20×30	**4911**	**4956**	**5.1E8**	5015	5102	9.3E8
dmu60	4739	20×30	**5065**	**5124**	**5.4E8**	5247	5293	8.3E8
dmu61	5172	15×40	**5522**	**5566**	**4.2E8**	5787	5851	9.7E8
dmu62	5251	15×40	**5412**	**5517**	**3.0E8**	5844	5864	8.6E8
dmu63	5323	15×40	**5503**	**5620**	**5.1E8**	5948	6025	7.6E8
dmu64	5240	15×40	**5501**	**5647**	**6.6E8**	5893	5947	9.2E8
dmu65	5190	15×40	**5403**	**5470**	**2.2E8**	5752	5788	8.1E8
dmu66	5717	20×40	**5951**	**6051**	**7.9E8**	6543	6592	9.0E8
dmu67	5779	20×40	**6102**	**6135**	**6.3E8**	6727	6788	8.7E8
dmu68	5765	20×40	**6095**	**6162**	**5.6E8**	6634	6727	9.5E8
dmu69	5709	20×40	**6016**	**6115**	**5.1E8**	6638	6673	8.7E8
dmu70	5889	20×40	**6237**	**6319**	**4.5E8**	6880	6921	8.7E8
dmu71	6223	15×50	**6517**	**6563**	**6.5E8**	7204	7264	9.3E8
dmu72	6463	15×50	**6703**	**6792**	**7.0E8**	7481	7511	9.2E8
dmu73	6153	15×50	**6453**	**6533**	**7.0E8**	7164	7221	8.6E8
dmu74	6196	15×50	**6443**	**6511**	**8.5E8**	7185	7231	8.7E8
dmu75	6189	15×50	**6547**	**6644**	**5.5E8**	7154	7224	9.7E8
dmu76	6807	20×50	**7135**	**7214**	**6.9E8**	8075	8156	9.3E8
dmu77	6792	20×50	**7217**	**7284**	1.0E9	8217	8288	**8.9E8**
dmu78	6770	20×50	**7119**	**7218**	**8.0E8**	8099	8210	8.4E8
dmu79	6952	20×50	**7294**	**7360**	**7.7E8**	8316	8409	9.0E8
dmu80	6673	20×50	**6967**	**7098**	**7.3E8**	8044	8085	8.6E8
		#best	65	60	79	20	23	1
			vs. (1+1)-EA on dmu*			3.8%	3.4%	4.0
ft06	55+	6×6	**55**	**55**	**1.5E3**	**55**	**55**	3.0E4
ft10	930+	10×10	937	964	**2.9E6**	**930**	**930**	4.5E7
ft20	1165+	5×20	**1165**	1169	2.8E8	**1165**	**1165**	**2.0E8**
		#best	2	1	2	3	3	1
			vs. (1+1)-EA on ft*			-0.2%	-1.3%	5.0
la01	666+	5×10	**666**	**666**	**2.0E4**	**666**	**666**	7.9E4
la02	655+	5×10	**655**	657	**9.3E5**	**655**	**655**	1.5E6
la03	597+	5×10	**597**	602	**1.1E6**	**597**	**597**	5.5E6
la04	590+	5×10	**590**	591	**9.4E4**	**590**	**590**	2.7E6
la05	593+	5×10	**593**	**593**	**2.5E2**	**593**	**593**	2.5E3
la06	926+	5×15	**926**	**926**	**3.9E2**	**926**	**926**	3.4E4
la07	890+	5×15	**890**	**890**	**4.3E3**	**890**	**890**	4.8E5
la08	863+	5×15	**863**	**863**	**9.1E2**	**863**	**863**	1.2E5
la09	951+	5×15	**951**	**951**	**5.2E2**	**951**	**951**	4.2E4
la10	958+	5×15	**958**	**958**	**2.4E2**	**958**	**958**	1.4E4
la11	1222+	5×20	**1222**	**1222**	**7.8E2**	**1222**	**1222**	8.4E4
la12	1039+	5×20	**1039**	**1039**	**4.3E2**	**1039**	**1039**	4.1E4
la13	1150+	5×20	**1150**	**1150**	**6.6E2**	**1150**	**1150**	6.5E4
la14	1292+	5×20	**1292**	**1292**	**1.1E2**	**1292**	**1292**	8.9E3
la15	1207+	5×20	**1207**	**1207**	**5.1E3**	**1207**	**1207**	1.7E6
la16	945+	10×10	946	963	**2.7E7**	**945**	**945**	2.9E8
la17	784+	10×10	**784**	785	**3.2E5**	**784**	**784**	3.7E7
la18	848+	10×10	**848**	856	**1.4E6**	**848**	**848**	1.1E7
la19	842+	10×10	**842**	860	8.5E7	**842**	**842**	**5.6E7**
la20	902+	10×10	907	925	**8.1E6**	**902**	**904**	3.2E8

Table 3: Results on the JSSP, part 3.

instance information			(1+1)-EA			(1+1)-FEA		
inst	*BKS*	*m*×*n*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
la21	1046+	10×15	1055	1067	**2.1E8**	**1047**	**1047**	3.0E8
la22	927+	10×15	**927**	**930**	**1.8E8**	932	933	4.3E8
la23	1032+	10×15	**1032**	**1032**	**1.0E5**	**1032**	**1032**	8.0E6
la24	935+	10×15	946	956	**2.3E6**	**938**	**939**	6.2E8
la25	977+	10×15	991	1005	**2.0E7**	**977**	**980**	5.6E8
la26	1218+	10×20	**1218**	**1218**	**5.4E5**	**1218**	**1218**	4.1E7
la27	1235+	10×20	1236	**1244**	**3.1E8**	**1235**	1245	7.4E8
la28	1216+	10×20	**1216**	**1217**	**2.9E7**	**1216**	1218	5.5E8
la29	1152+	10×20	**1163**	1177	**5.3E7**	1164	**1171**	6.0E8
la30	1355+	10×20	**1355**	**1355**	**1.6E5**	**1355**	**1355**	2.0E7
la31	1784+	10×30	**1784**	**1784**	**9.7E3**	**1784**	**1784**	4.5E6
la32	1850+	10×30	**1850**	**1850**	**1.2E4**	**1850**	**1850**	8.4E6
la33	1719+	10×30	**1719**	**1719**	**1.2E4**	**1719**	**1719**	5.7E6
la34	1721+	10×30	**1721**	**1721**	**3.6E4**	**1721**	**1721**	3.1E7
la35	1888+	10×30	**1888**	**1888**	**1.6E4**	**1888**	**1888**	9.8E6
la36	1268+	15×15	**1268**	1287	2.5E8	1278	**1283**	**2.5E8**
la37	1397+	15×15	1418	1421	**3.4E7**	**1410**	**1412**	5.3E8
la38	1196+	15×15	**1196**	1224	**2.3E8**	1202	**1206**	5.1E8
la39	1233+	15×15	**1238**	1248	**4.2E7**	1240	**1245**	4.8E8
la40	1222+	15×15	1229	1245	**1.4E7**	**1228**	**1228**	4.2E8
		#best	32	23	38	35	37	2
			vs. (1+1)-EA on la*			-0.1%	-0.5%	32
orb01	1059+	10×10	1094	1110	**9.7E7**	**1059**	**1059**	1.4E8
orb02	888+	10×10	**889**	902	4.9E7	**889**	**889**	**2.4E7**
orb03	1005+	10×10	1044	1076	**7.9E7**	**1005**	**1005**	9.8E7
orb04	1005+	10×10	1020	1037	**6.4E5**	**1005**	**1006**	3.8E8
orb05	887+	10×10	890	917	**5.8E6**	**889**	**889**	7.6E7
orb06	1010+	10×10	1033	1047	**1.1E6**	**1010**	**1010**	6.6E8
orb07	397+	10×10	401	405	**7.6E4**	**397**	**397**	3.4E7
orb08	899+	10×10	916	935	**3.4E7**	**899**	**899**	3.5E7
orb09	934+	10×10	950	970	**5.9E5**	**934**	**934**	1.2E8
orb10	944+	10×10	**944**	985	4.6E7	**944**	**944**	**3.7E7**
		#best	2	0	8	10	10	2
			vs. (1+1)-EA on orb*			-1.5%	-3.6%	13
swv01	1407+	10×20	**1418**	1459	**3.0E8**	1435	**1450**	5.9E8
swv02	1475+	10×20	**1485**	1509	**1.7E8**	1495	**1498**	6.1E8
swv03	1398+	10×20	1450	1484	4.3E8	**1433**	**1443**	**4.0E8**
swv04	1464+	10×20	1523	1543	**4.0E8**	**1509**	**1514**	5.7E8
swv05	1424+	10×20	1494	1533	**1.4E8**	**1466**	**1470**	5.5E8
swv06	1671	15×20	1766	1785	**1.9E8**	**1720**	**1729**	5.8E8
swv07	1594	15×20	1711	1725	**1.3E8**	**1665**	**1673**	7.1E8
swv08	1752	15×20	1879	1902	**2.4E8**	**1809**	**1819**	7.6E8
swv09	1655	15×20	1784	1813	**1.8E8**	**1700**	**1716**	6.8E8
swv10	1743	15×20	1833	1871	**9.2E7**	**1798**	**1810**	5.0E8
swv11	2983+	10×50	**2997**	**3021**	**6.1E8**	3143	3155	7.7E8
swv12	2977	10×50	**3047**	**3100**	**3.8E8**	3178	3211	7.5E8
swv13	3104+	10×50	**3124**	**3152**	**4.1E8**	3252	3264	8.4E8
swv14	2968+	10×50	**2979**	**3003**	**4.0E8**	3048	3074	7.8E8
swv15	2885+	10×50	**2917**	**2958**	**4.2E8**	3081	3094	9.0E8
swv16	2924+	10×50	**2924**	**2924**	**4.1E3**	**2924**	**2924**	1.8E6
swv17	2794+	10×50	**2794**	**2794**	**8.6E3**	**2794**	**2794**	4.6E6
swv18	2852+	10×50	**2852**	**2852**	**5.8E3**	**2852**	**2852**	2.4E6
swv19	2843+	10×50	**2843**	**2843**	**1.4E4**	**2843**	**2843**	1.2E7
swv20	2823+	10×50	**2823**	**2823**	**8.9E3**	**2823**	**2823**	4.3E6
		#best	12	10	19	13	15	1
			vs. (1+1)-EA on swv*			0.1%	-0.6%	8.6

Table 4: Results on the JSSP, part 4.

instance information			(1+1)-EA			(1+1)-FEA		
inst	*BKS*	*m*×*n*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
ta01	1231+	15×15	1254	1264	**1.9E7**	**1248**	**1248**	3.7E8
ta02	1244+	15×15	1258	1269	**9.1E7**	**1244**	**1249**	3.0E8
ta03	1218+	15×15	1261	1276	**1.5E8**	**1218**	**1221**	5.1E8
ta04	1175+	15×15	1185	1213	**2.5E8**	**1181**	**1186**	4.7E8
ta05	1224+	15×15	1243	1258	**3.5E6**	**1233**	**1239**	4.7E8
ta06	1238+	15×15	1259	1277	**6.2E6**	**1246**	**1250**	4.4E8
ta07	1227+	15×15	1250	1255	**3.5E7**	**1228**	**1236**	5.0E8
ta08	1217+	15×15	1241	1255	**2.7E7**	**1219**	**1230**	5.4E8
ta09	1274+	15×15	1296	1322	**2.1E8**	**1281**	**1291**	4.9E8
ta10	1241+	15×15	1277	1291	**1.3E8**	**1244**	**1262**	6.5E8
ta11	1357+	15×20	**1389**	1408	**3.9E8**	**1389**	**1401**	7.0E8
ta12	1367+	15×20	**1379**	1400	**3.2E7**	1388	**1397**	5.2E8
ta13	1342+	15×20	1381	1400	**5.2E7**	**1361**	**1376**	8.5E8
ta14	1345+	15×20	**1355**	**1365**	**1.8E8**	1363	**1365**	5.8E8
ta15	1339+	15×20	1364	1390	**1.8E8**	**1362**	**1375**	5.9E8
ta16	1360+	15×20	**1375**	1399	**5.4E7**	**1375**	**1381**	5.0E8
ta17	1462+	15×20	**1489**	1503	**3.3E8**	1494	**1500**	5.6E8
ta18	1396	15×20	**1424**	1438	**3.1E8**	1429	**1437**	5.7E8
ta19	1332+	15×20	1380	1400	**9.3E7**	**1343**	**1372**	6.8E8
ta20	1348+	15×20	1374	1395	**1.2E8**	**1373**	**1381**	5.0E8
ta21	1642+	20×20	**1671**	1710	**2.1E8**	1678	**1686**	5.2E8
ta22	1600	20×20	1634	1649	**1.8E8**	**1632**	**1640**	7.2E8
ta23	1557	20×20	1602	1617	**2.2E8**	**1589**	**1595**	7.2E8
ta24	1644+	20×20	**1675**	1696	**2.0E8**	1676	**1688**	4.8E8
ta25	1595	20×20	1634	1651	**4.6E7**	**1625**	**1642**	5.5E8
ta26	1643	20×20	1698	1709	**3.4E8**	**1664**	**1684**	6.4E8
ta27	1680	20×20	1747	1785	**1.0E8**	**1704**	**1721**	4.8E8
ta28	1603+	20×20	1647	1668	**1.4E8**	**1627**	**1645**	7.6E8
ta29	1625	20×20	1652	1693	**2.7E7**	**1644**	**1649**	7.9E8
ta30	1584	20×20	**1624**	**1640**	**1.9E8**	1637	**1640**	7.0E8
ta31	1764+	15×30	**1766**	**1766**	**1.6E8**	1793	1805	8.6E8
ta32	1784	15×30	**1835**	**1845**	**4.4E8**	1868	1878	6.6E8
ta33	1791	15×30	**1804**	**1826**	**4.9E8**	1867	1871	9.1E8
ta34	1829	15×30	**1844**	**1873**	**3.0E8**	1888	1919	7.9E8
ta35	2007+	15×30	**2007**	**2012**	**1.0E8**	2015	2018	6.5E8
ta36	1819+	15×30	**1820**	**1834**	**3.1E8**	1846	1861	7.5E8
ta37	1771+	15×30	**1795**	**1801**	**7.1E8**	1841	1852	7.2E8
ta38	1673+	15×30	**1696**	**1708**	**3.4E8**	1737	1744	7.9E8
ta39	1795+	15×30	**1807**	**1815**	**2.8E8**	1841	1848	5.9E8
ta40	1669	15×30	**1703**	**1718**	**5.5E8**	1737	1750	7.3E8
ta41	2005	20×30	**2057**	**2088**	**4.8E8**	2115	2131	8.3E8
ta42	1937	20×30	**2006**	**2027**	**3.8E8**	2023	2051	8.0E8
ta43	1846	20×30	**1915**	**1930**	**7.2E8**	1959	1978	8.2E8
ta44	1979	20×30	**2030**	**2051**	**5.1E8**	2065	2091	7.7E8
ta45	2000	20×30	**2016**	**2033**	**3.9E8**	2073	2086	8.7E8
ta46	2004	20×30	**2064**	**2100**	**5.7E8**	2109	2135	7.5E8
ta47	1889	20×30	**1970**	**1980**	**2.1E8**	2004	2022	8.3E8
ta48	1937	20×30	**1998**	**2008**	**4.9E8**	2066	2076	7.1E8
ta49	1961	20×30	**2010**	**2026**	**5.0E8**	2060	2081	7.9E8
ta50	1923	20×30	**1974**	**1995**	**2.8E8**	2041	2057	7.3E8
ta51	2760+	15×50	**2760**	**2760**	**1.0E6**	**2760**	2769	8.1E8
ta52	2756+	15×50	**2756**	**2756**	**1.0E6**	**2756**	2772	8.9E8
ta53	2717+	15×50	**2717**	**2717**	**5.5E5**	**2717**	**2717**	8.6E8
ta54	2839+	15×50	**2839**	**2839**	**2.7E5**	**2839**	**2839**	3.5E8
ta55	2679+	15×50	**2679**	**2679**	**2.7E6**	2719	2730	8.7E8
ta56	2781+	15×50	**2781**	**2781**	**1.1E6**	2795	2803	9.0E8

Table 5: Results on the JSSP, part 5.

instance information			(1+1)-EA			(1+1)-FEA		
inst	*BKS*	*m*×*n*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
ta57	2943+	15×50	**2943**	**2943**	**4.7E5**	**2943**	2945	8.3E8
ta58	2885+	15×50	**2885**	**2885**	**3.9E5**	**2885**	**2885**	9.4E8
ta59	2655+	15×50	**2655**	**2655**	**2.0E6**	2703	2712	8.8E8
ta60	2723+	15×50	**2723**	**2723**	**1.1E6**	2746	2752	9.3E8
ta61	2868+	20×50	**2868**	**2868**	**7.4E6**	2984	3000	9.2E8
ta62	2869+	20×50	**2872**	**2883**	**8.2E8**	3060	3088	8.3E8
ta63	2755+	20×50	**2755**	**2755**	**8.4E6**	2865	2884	8.4E8
ta64	2702+	20×50	**2702**	**2702**	**6.2E6**	2813	2836	7.6E8
ta65	2725+	20×50	**2725**	**2725**	**2.8E7**	2854	2867	8.8E8
ta66	2845+	20×50	**2845**	**2845**	**3.0E7**	2968	2983	8.2E8
ta67	2825+	20×50	**2826**	**2826**	**2.4E7**	2949	2968	7.8E8
ta68	2784+	20×50	**2784**	**2784**	**6.3E6**	2865	2884	9.1E8
ta69	3071+	20×50	**3071**	**3071**	**2.3E6**	3140	3160	8.2E8
ta70	2995+	20×50	**2995**	**2995**	**1.3E7**	3144	3169	8.2E8
ta71	5464+	20×100	**5464**	**5464**	**1.1E6**	5561	5644	8.5E8
ta72	5181+	20×100	**5181**	**5181**	**8.7E5**	5285	5319	8.1E8
ta73	5568+	20×100	**5568**	**5568**	**1.3E6**	5728	5741	7.8E8
ta74	5339+	20×100	**5339**	**5339**	**7.5E5**	5441	5457	8.8E8
ta75	5392+	20×100	**5392**	**5392**	**1.3E6**	5596	5630	8.6E8
ta76	5342+	20×100	**5342**	**5342**	**1.1E6**	5489	5503	9.1E8
ta77	5436+	20×100	**5436**	**5436**	**6.1E5**	5509	5538	9.3E8
ta78	5394+	20×100	**5394**	**5394**	**8.2E5**	5494	5517	8.0E8
ta79	5358+	20×100	**5358**	**5358**	**7.5E5**	5432	5459	9.2E8
ta80	5183+	20×100	**5183**	**5183**	**1.1E6**	5318	5377	9.5E8
		#best	59	52	80	29	33	0
			vs. (1+1)-EA on ta*			1.1%	1.0%	20
yn1	884+	20×20	907	913	**5.6E7**	**901**	**904**	5.3E8
yn2	904	20×20	929	934	**2.4E8**	**922**	**927**	6.8E8
yn3	892	20×20	905	912	**1.3E8**	**899**	**910**	7.5E8
yn4	968	20×20	**977**	991	**1.7E8**	978	**987**	7.0E8
		#best	1	0	4	3	4	0
			vs. (1+1)-EA on yn*			-0.5%	-0.6%	4.0
			overall					
		#best	174	146	235	118	130	7
			vs. (1+1)-EA on all			1.5%	1.2%	12

such scenarios, an FFA-based search would degenerate to perform similar to a Random Walk.

In a next step, we will plug FFA into some of the sate-of-the-art algorithms on the JSSP domain. The present paper shows that a simple local search can benefit (or at least not suffer) from using FFA. Whether it can help or will be detrimental for highly-optimized algorithms must be investigated next.

ACKNOWLEDGMENTS

We acknowledge support from the National Natural Science Foundation of China under grant 61673359, the Youth Project of the Provincial Natural Science Foundation of Anhui 1908085QF285, the Talent Fund of Hefei University 18-19RC26, the University Natural Sciences Research Project of Anhui Province KJ2020A0661, as well as the Hefei Specially Recruited Foreign Expert program.

REFERENCES

[1] Tamer F. Abdelmaguid. 2010. Representations in Genetic Algorithm for the Job Shop Scheduling Problem: A Computational Study. *Journal of Software*

Engineering and Applications 3, 12 (2010), 1155–1162. https://doi.org/10.4236/jsea.2010.312135

[2] Joseph Adams, Egon Balas, and Daniel Zawack. 1988. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science* 34, 3 (1988), 391–401. https://doi.org/10.1287/mnsc.34.3.391

[3] Denis Antipov and Benjamin Doerr. 2018. Precise Runtime Analysis for Plateaus. In *15th Intl. Conf. on Parallel Problem Solving from Nature (PPSN XV), Part II, Sept. 8–12, 2018, Coimbra, Portugal*. Springer, Cham, Switzerland, 117–128. https://doi.org/10.1007/978-3-319-99259-4_10

[4] David Lee Applegate and William John Cook. 1991. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing* 3, 2 (1991), 149–156. https://doi.org/10.1287/ijoc.3.2.149

[5] Leila Asadzadeh. 2015. A Local Search Genetic Algorithm for the Job Shop Scheduling Problem with Intelligent Agents. *Computers & Industrial Engineering* 85 (2015), 376–383. https://doi.org/10.1016/j.cie.2015.04.006

[6] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz (Eds.). 1997. *Handbook of Evolutionary Computation*. Bristol, UK: Institute of Physics Publishing and New York, NY, USA: Oxford University Press.

[7] Christian Bierwirth. 1995. A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms. *The Journal of the Operational Research Society* 17, 2–3 (1995), 87–92. https://doi.org/10.1007/BF01719250

[8] Jacek Błażewicz, Wolfgang Domschke, and Erwin Pesch. 1996. The Job Shop Scheduling Problem: Conventional and New Solution Techniques. *European Journal of Operational Research* 93, 1 (1996), 1–33. https://doi.org/10.1016/0377-2217(95)00362-2

[9] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. 1998. A Review of Machine Scheduling: Complexity, Algorithms and Approximability. In *Handbook of Combinatorial Optimization*, Ding-Zhu Du and Panos M. Pardalos (Eds.). Springer, Boston, MA, USA, 1493–1641. https://doi.org/10.1007/978-1-4613-0303-9_25

[10] Antoine Cully and Yiannis Demiris. 2018. Quality and Diversity Optimization: A Unifying Modular Framework. *IEEE Transactions on Evolutionary Computation* 22, 2 (2018), 245–259. https://doi.org/10.1109/TEVC.2017.2704781

[11] Ebru Demirkol, Sanjay V. Mehta, and Reha Uzsoy. 1998. Benchmarks for Shop Scheduling Problems. *European Journal of Operational Research* 109, 1 (1998), 137–141. https://doi.org/10.1016/S0377-2217(97)00019-2

[12] Henry Fisher and Gerald L. Thompson. 1963. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In *Industrial Scheduling*, John F. Muth and Gerald L. Thompson (Eds.). Prentice-Hall, Englewood Cliffs, NJ, USA, 225–251.

[13] Philip J. Fleming and John J. Wallace. 1986. How Not to Lie with Statistics: The Correct Way to Summarize Benchmark Results. *Commun. ACM* 29, 3 (1986), 218–221. https://doi.org/10.1145/5666.5673

[14] Edson Flórez, Wilfredo Gómez, and Lola Bautista. 2013. *An Ant Colony Optimization Algorithm for Job Shop Scheduling Problem*. Computing Research Repository (CoRR) abs/1309.5110. arxiv. https://arxiv.org/pdf/1309.5110.pdf

[15] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. W. H. Freeman and Company, New York, NY, USA.

[16] Mitsuo Gen, Yasuhiro Tsujimura, and Erika Kubota. 1994. Solving Job-Shop Scheduling Problems by Genetic Algorithm. In *Humans, Information and Technology: IEEE Intl. Conf. on Systems, Man and Cybernetics, Oct. 2–5, 1994, San Antonio, TX, USA*, Vol. 2. IEEE. https://doi.org/10.1109/ICSMC.1994.400072

[17] Fred W. Glover, Éric D. Taillard, and Dominique de Werra. 1993. A User's Guide to Tabu Search. *Annals of Operations Research* 41, 1 (1993), 3–28. https://doi.org/10.1007/BF02078647

[18] Daniele Gravina, Antonios Liapis, and Georgios N. Yannakakis. 2019. Quality Diversity Through Surprise. *IEEE Transactions on Evolutionary Computation* 23, 4 (2019), 603–616. https://doi.org/10.1109/TEVC.2018.2877215

[19] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. 2011. Impacts of Invariance in Search: When CMA-ES and PSO Face Ill-Conditioned and Non-Separable Problems. *Applied Soft Computing* 11, 8 (2011), 5755–5769. https://doi.org/10.1016/j.asoc.2011.03.001

[20] Tianhua Jiang and Chao Zhang. 2018. Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases. *IEEE Access* 6 (2018), 26231–26240. https://doi.org/10.1109/ACCESS.2018.2833552

[21] Vedavyasrao Jorapur, V. S. Puranik, A. S. Deshpande, and M. R. Sharma. 2014. Comparative Study of Different Representations in Genetic Algorithms for Job Shop Scheduling Problem. *Journal of Software Engineering and Applications* 7, 7 (2014), 571–580. https://doi.org/10.4236/jsea.2014.77053

[22] Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Science Magazine* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671

[23] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. 1993. Sequencing and Scheduling: Algorithms and Complexity. In *Handbook of Operations Research and Management Science*, Stephen C. Graves, Alexander Hendrik George Rinnooy Kan, and Paul H. Zipkin (Eds.). Vol. IV. North-Holland Scientific Publishers Ltd., Amsterdam, The Netherlands, 445–522. https://doi.org/10.1016/S0927-0507(05)80189-6

[24] Stephen R. Lawrence. 1984. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*. Ph.D. Dissertation. Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, USA.

[25] Joss Miller-Todd, Kathleen Steinhöfel, and Patrick Veenstra. 2018. Firefly-Inspired Algorithm for Job Shop Scheduling. In *Adventures Between Lower Bounds and Higher Altitudes – Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger (Eds.). Springer, 423–433. https://doi.org/10.1007/978-3-319-98355-4_24

[26] S. Narendhar and T. Amudha. 2012. A Hybrid Bacterial Foraging Algorithm For Solving Job Shop Scheduling Problems. *Intl. Journal of Programming Languages and Applications* 2, 4 (2012), 1–11. https://doi.org/10.5121/ijpla.2012.2401

[27] Yann Ollivier, Ludovic Arnold, Anne Auger, and Nikolaus Hansen. 2017. Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles. *Journal of Machine Learning Research* 18 (2017), 1–65. http://jmlr.org/papers/v18/14-467.html

[28] Guoyong Shi, Hitoshi Iima, and Nobuo Sannomiya. 1997. New Encoding Scheme for Solving Job Shop Problems by Genetic Algorithm. In *35th IEEE Conf. on Decision and Control (CDC'96), Dec. 11–13, 1996, Kobe, Japan*, Vol. 4. IEEE, 4395–4400. https://doi.org/10.1109/CDC.1996.577484

[29] Oleg V. Shylo. 2019. Job Shop Scheduling. http://optimizizer.com/jobshop.php

[30] Robert H. Storer, S. David Wu, and Renzo Vaccari. 1992. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Management Science* 38, 10 (1992), 1495–1509. https://doi.org/10.1287/mnsc.38.10.1495

[31] Éric D. Taillard. 1993. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research* 64, 2 (1993), 278–285. https://doi.org/10.1016/0377-2217(93)90182-M

[32] Jelke Jeroen van Hoorn. 2016. Job Shop Instances and Solutions. http://jobshop.jjvh.nl

[33] Shao-Juan Wang, Chun-Wei Tsai, and Ming-Chao Chiang. 2018. A High Performance Search Algorithm for Job-Shop Scheduling Problem. In *9th Intl. Conf. on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN'18) / 8th Intl. Conf. on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH'18) / Affiliated Workshops, Nov. 5–8, 2018, Leuven, Belgium*. Elsevier, 119–126. https://doi.org/10.1016/j.procs.2018.10.157

[34] Thomas Weise. 2019–2020. jsspInstancesAndResults: Results, Data, and Instances of the Job Shop Scheduling Problem. https://github.com/thomasWeise/jsspInstancesAndResults as viewed on 2020-11-25.

[35] Thomas Weise, Mingxu Wan, Ke Tang, Pu Wang, Alexandre Devert, and Xin Yao. 2014. Frequency Fitness Assignment. *IEEE Transactions on Evolutionary Computation* 18, 2 (2014), 226–243. https://doi.org/10.1109/TEVC.2013.2251885

[36] Thomas Weise, Zhize Wu, Xinlu Li, and Yan Chen. 2021. Frequency Fitness Assignment: Making Optimization Algorithms Invariant under Bijective Transformations of the Objective Function Value. *IEEE Transactions on Evolutionary Computation* 25, 2 (April 2021). https://doi.org/10.1109/TEVC.2020.3032090 preprint available at arXiv:2001.01416v5 [cs.NE] 15 Oct 2020.

[37] L. Darrell Whitley. 1989. The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In *3rd Intl. Conf. on Genetic Algorithms (ICGA'89), June 4–7, 1989, Fairfax, VA, USA*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 116–121.

[38] Takeshi Yamada and Ryohei Nakano. 1992. A Genetic Algorithm Applicable to Large-Scale Job-Shop Instances. In *Parallel Problem Solving from Nature 2 (PPSN II), Sept. 28–30, 1992, Brussels, Belgium*. Elsevier, Amsterdam, The Netherlands, 281–290.