合肥学院
HEFEI UNIVERSITY

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

Thomas Weise · 汤卫思
tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn/5

Institute of Applied Optimization (IAO)	应用优化研究所
School of Artificial Intelligence and Big Data	人工智能与大数据学院
Hefei University	合肥学院
Hefei, Anhui, China	中国安徽省合肥市

Outline

Job Shop Scheduling Problem (JSSP)

Job Shop Scheduling Problem (JSSP)

- The JSSP is an $\mathcal{NP}$-hard[3] [4] scheduling task.

Job Shop Scheduling Problem (JSSP)

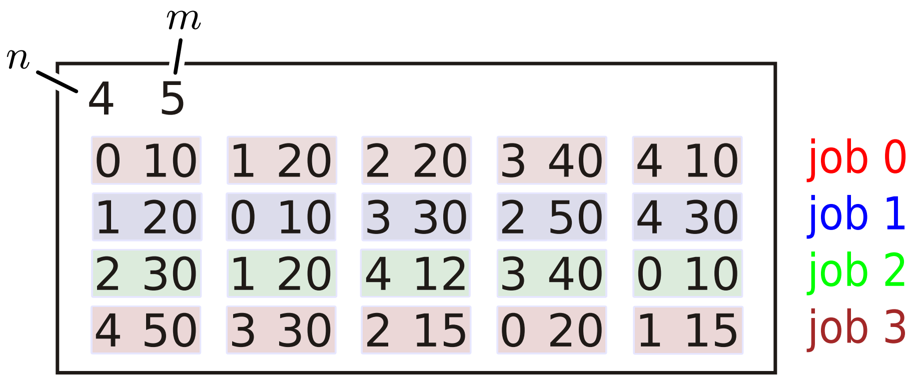
- The JSSP is an $\mathcal{NP}$-hard[3] [4] scheduling task.
- A JSSP instance defined by n jobs with operations that need to be processed by each of the m machines in a job-specific order.

Job Shop Scheduling Problem (JSSP)

- The JSSP is an $\mathcal{NP}$-hard[3] [4] scheduling task.
- A JSSP instance defined by n jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.

Job Shop Scheduling Problem (JSSP)

- The JSSP is an $\mathcal{NP}$-hard[3] [4] scheduling task.
- A JSSP instance defined by n jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.



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

Job Shop Scheduling Problem (JSSP)

- The JSSP is an $\mathcal{NP}$-hard[3] [4] scheduling task.
- A JSSP instance defined by n jobs with operations that need to be processed by each of the m machines in a job-specific order.
- Each such operation of each job needs a specific time.
- The goal is to find the assignment of jobs to machines with the smallest possible makespan, i.e., schedule that finishes fastest.

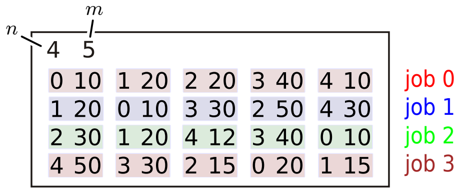
Encoding and Solutions

- We encode solutions as permutations with repetitions, i.e., where each of the n job IDs appears m times in a linear string[5–7].

Encoding and Solutions

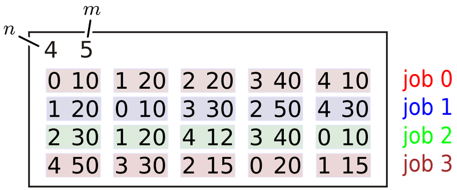
- We encode solutions as permutations with repetitions, i.e., where each of the n job IDs appears m times in a linear string[5–7].
- The strings are processed from front to end to obtain a Gantt chart.

Encoding and Solutions



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

Encoding and Solutions



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

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)

Encoding and Solutions

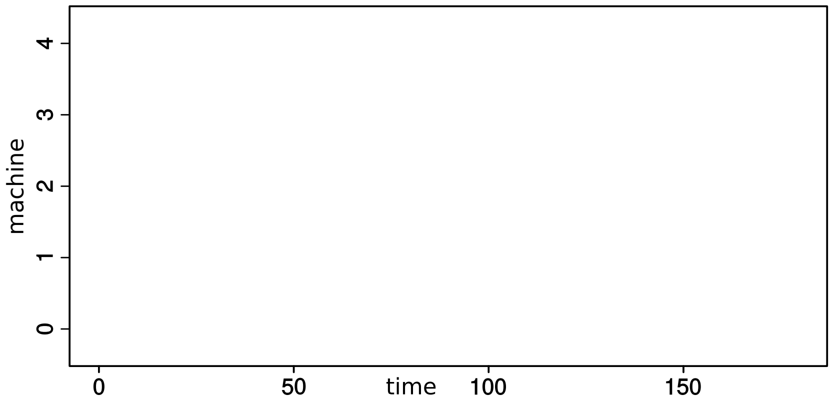
n m

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

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)



Encoding and Solutions

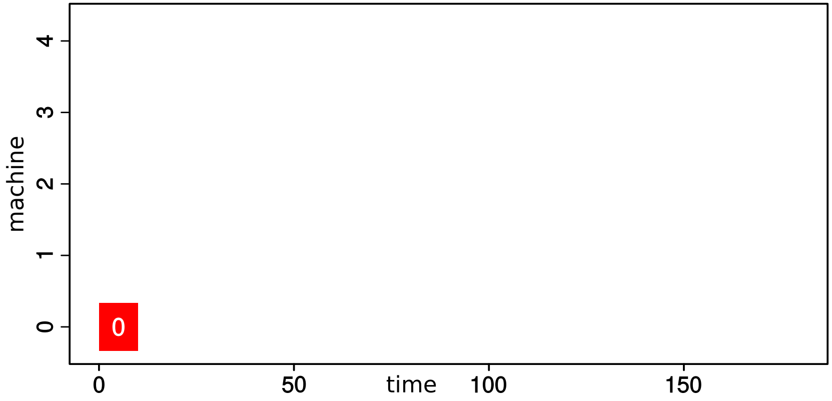
n m

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

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)



Encoding and Solutions

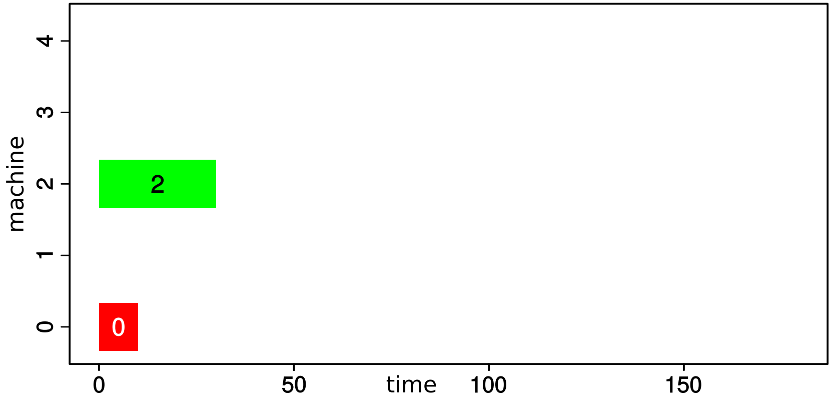
n m

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

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)



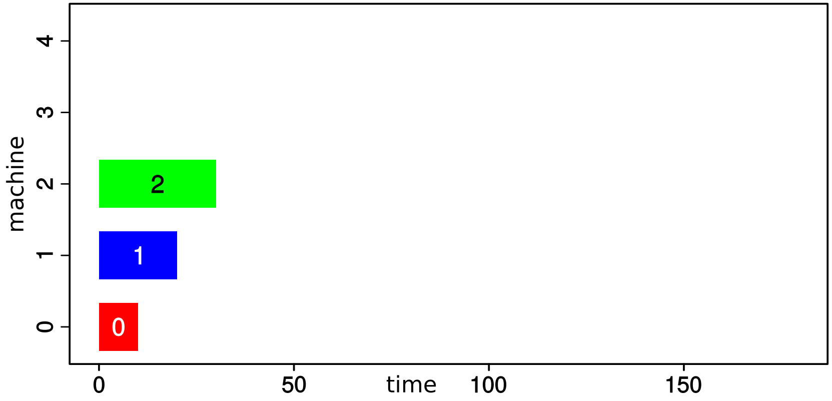
Encoding and Solutions

n m

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

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)

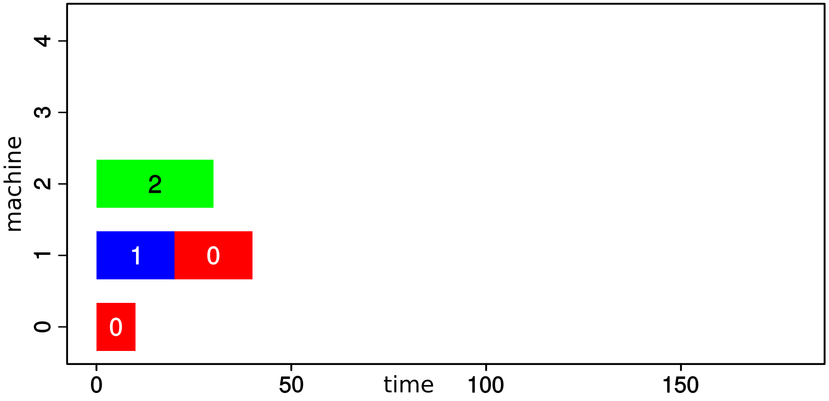


Encoding and Solutions

```
      m
n   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
```

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)



Encoding and Solutions

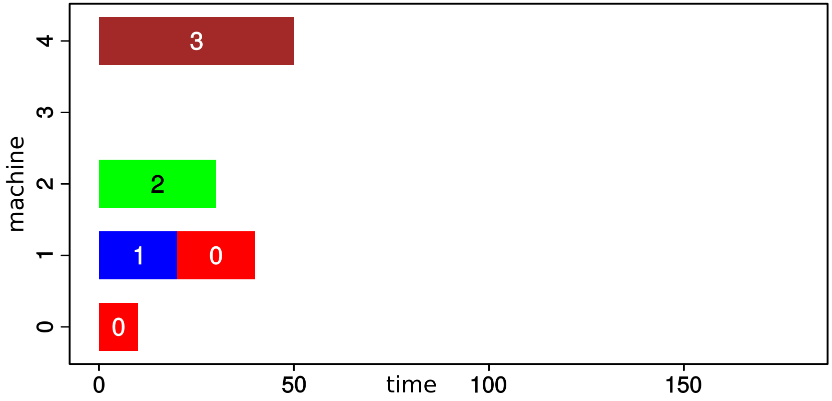
n m

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

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)



Encoding and Solutions

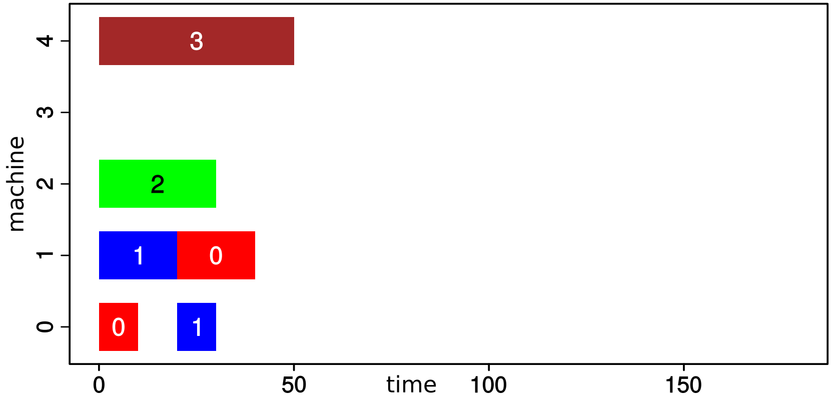
n m

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

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)



Encoding and Solutions

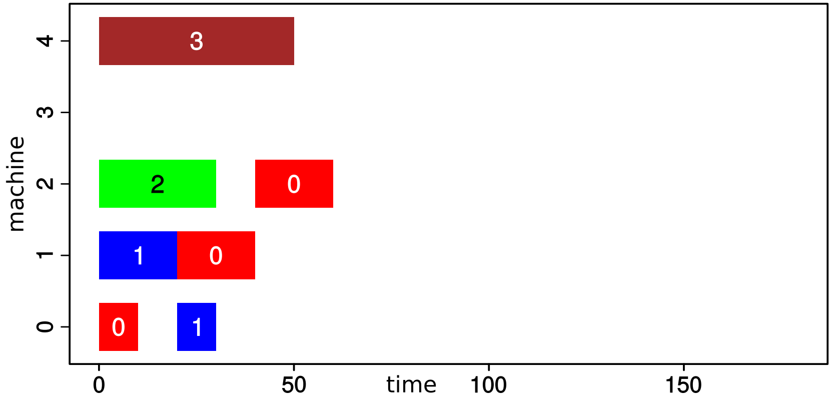
n m

4 5

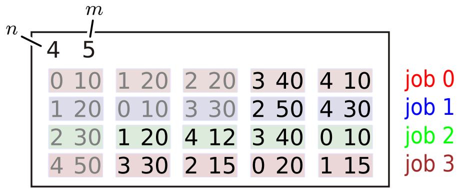
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

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)



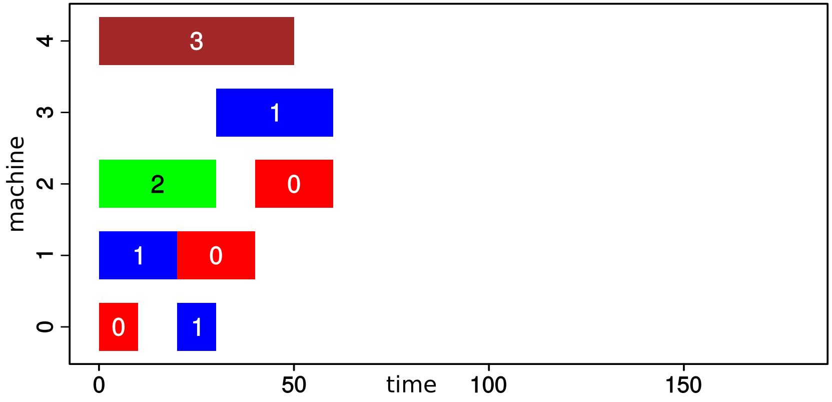
Encoding and Solutions



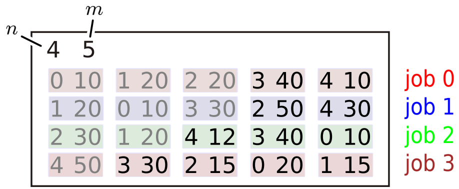
4	5				
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

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)



Encoding and Solutions

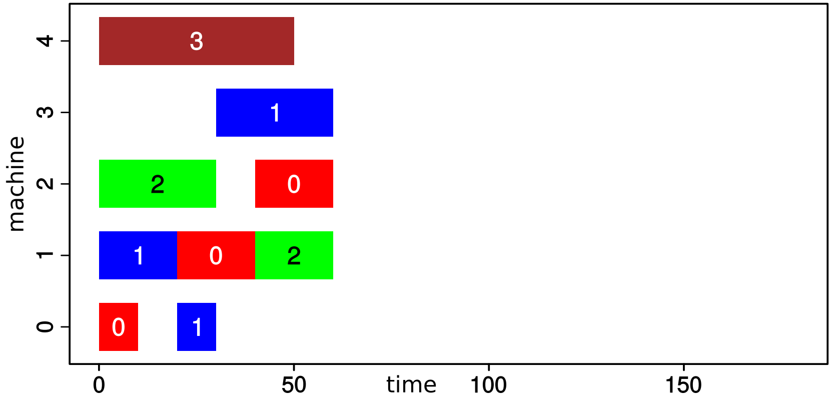


4 5

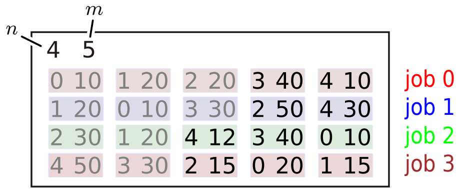
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

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)



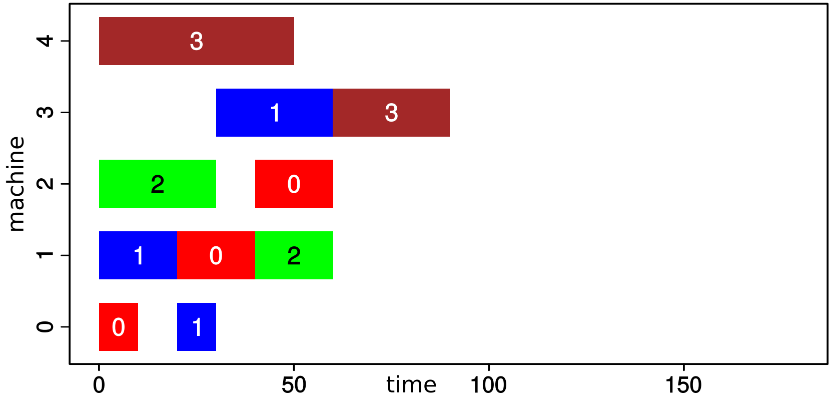
Encoding and Solutions



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

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)



Encoding and Solutions

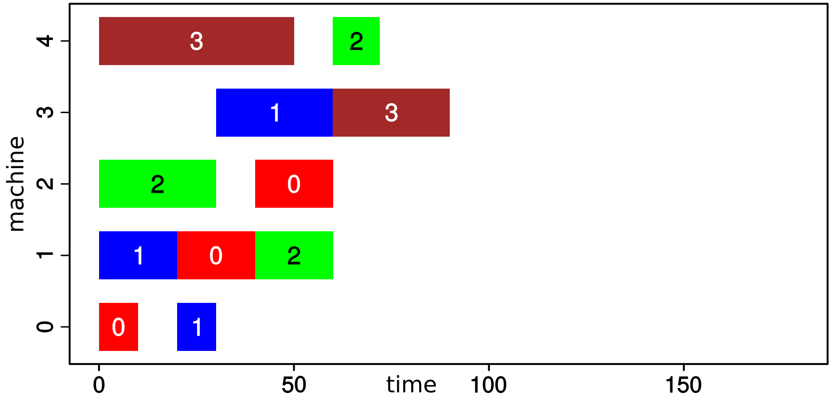
n m

4 5

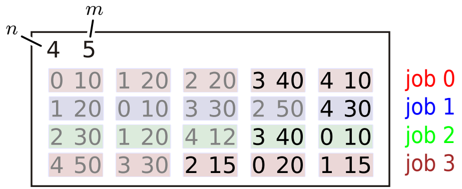
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

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)



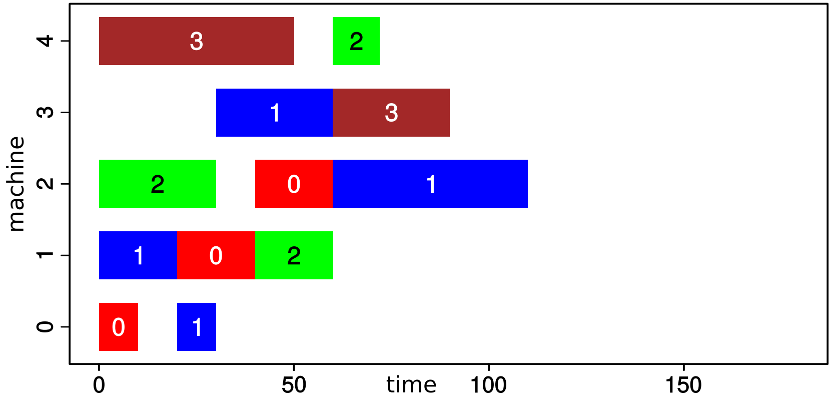
Encoding and Solutions



```
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
```

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)



Encoding and Solutions

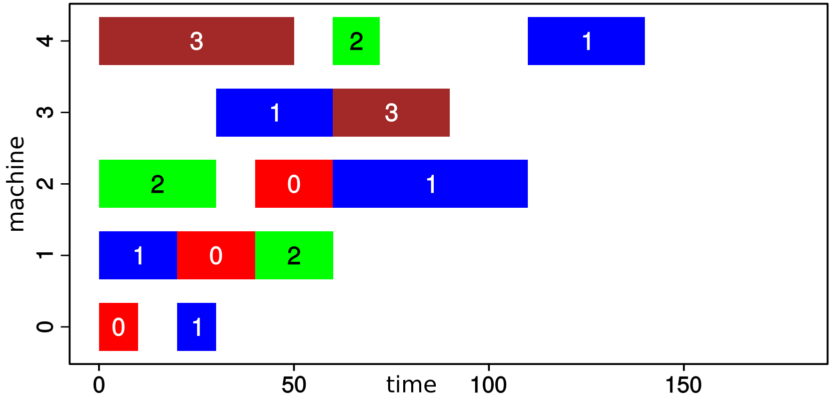
n m

4 5

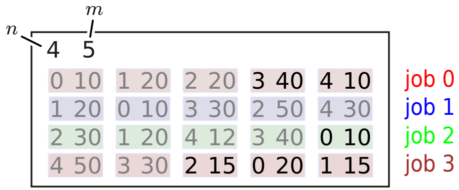
0 10	1 20	2 20	3 40	4 10	job 0
1 20	0 10	3 30	2 50	4 30	job 1
2 30	1 20	4 12	3 40	0 10	job 2
4 50	3 30	2 15	0 20	1 15	job 3

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)



Encoding and Solutions

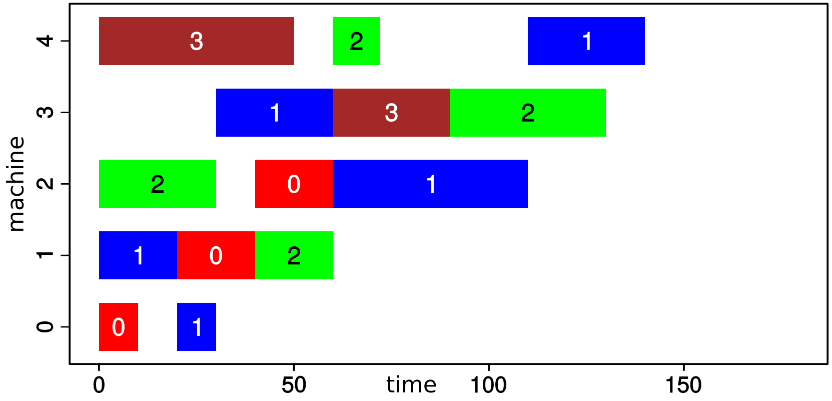


4 5

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

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)

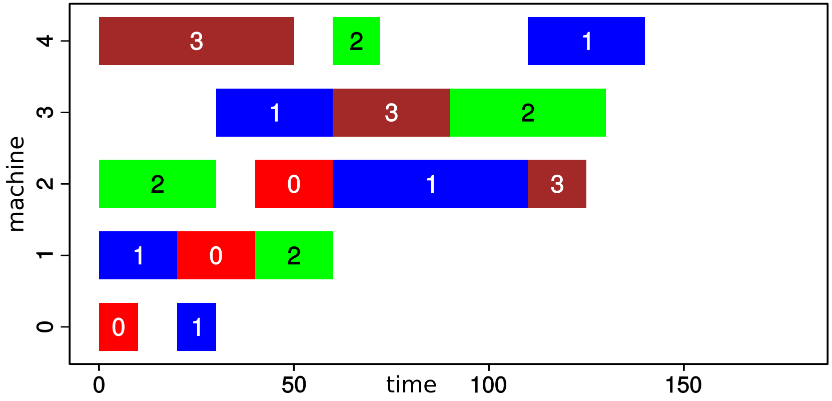


Encoding and Solutions

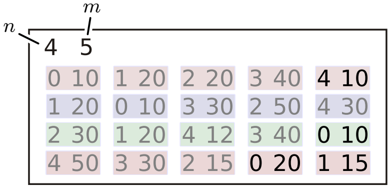
```
       m
 n
   4   5
   0 10   1 20   2 20   3 40   4 10    job 0
   1 20   0 10   3 30   2 50   4 30    job 1
   2 30   1 20   4 12   3 40   0 10    job 2
   4 50   3 30   2 15   0 20   1 15    job 3
```

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)



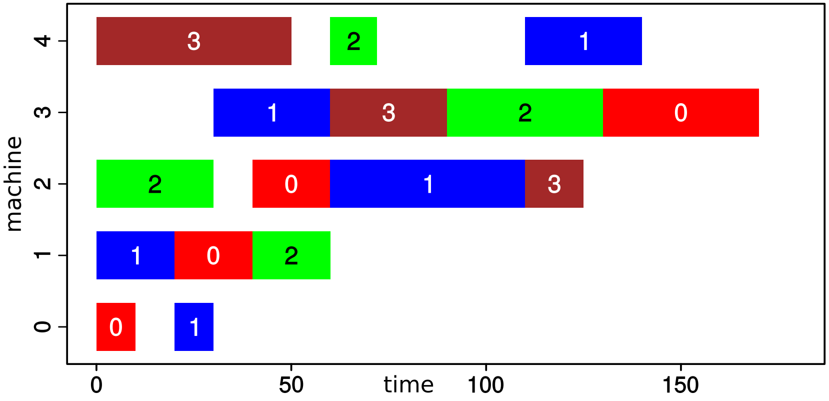
Encoding and Solutions



```
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
```

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)



Encoding and Solutions

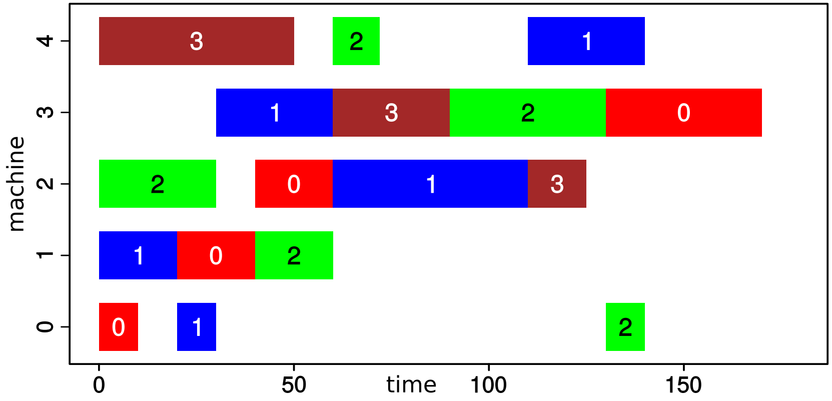
n m

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

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)



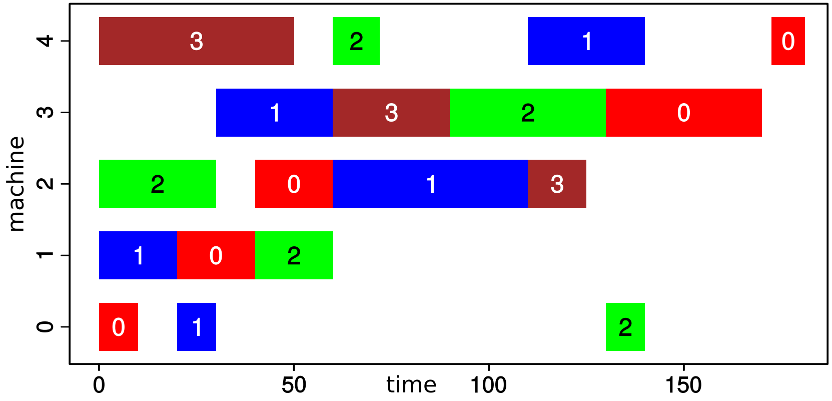
Encoding and Solutions

n m

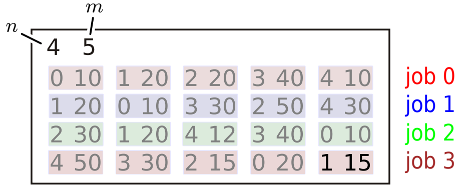
```
4  5
0 10  1 20  2 20  3 40  4 10   job 0
1 20  0 10  3 30  2 50  4 30   job 1
2 30  1 20  4 12  3 40  0 10   job 2
4 50  3 30  2 15  0 20  1 15   job 3
```

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)



Encoding and Solutions

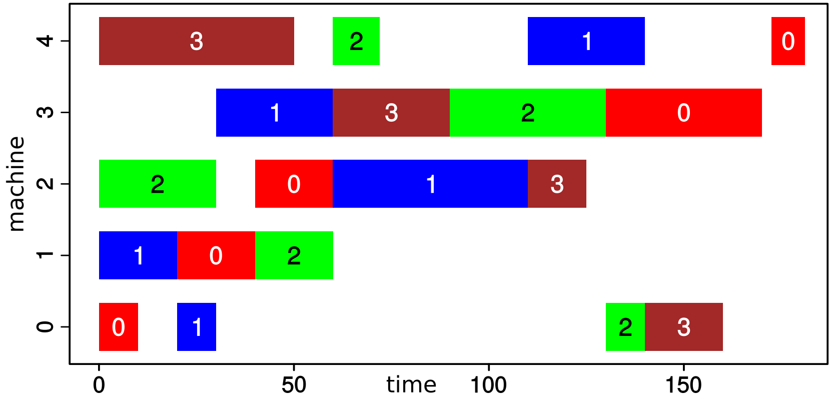


4 5

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

example point in the search space:
permutation with repetitions:

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

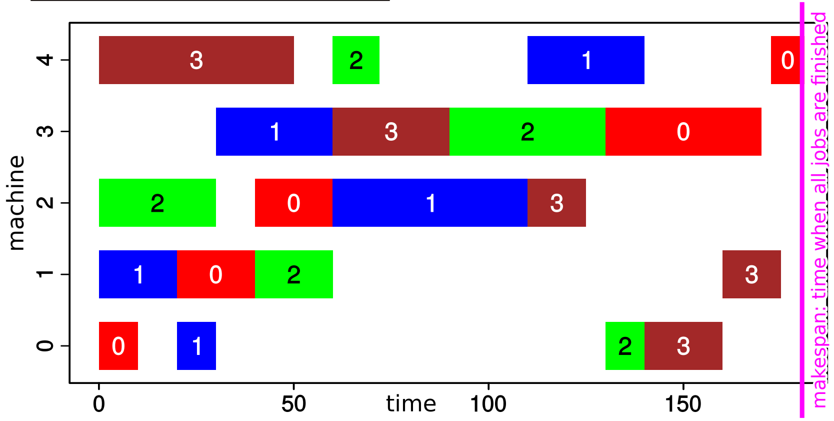


Encoding and Solutions

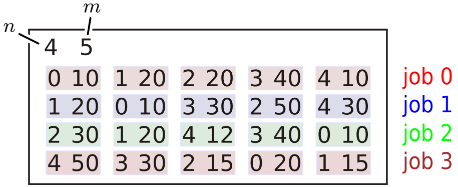
```
     m
  n
   4   5
   0 10   1 20   2 20   3 40   4 10    job 0
   1 20   0 10   3 30   2 50   4 30    job 1
   2 30   1 20   4 12   3 40   0 10    job 2
   4 50   3 30   2 15   0 20   1 15    job 3
```

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)



Encoding and Solutions

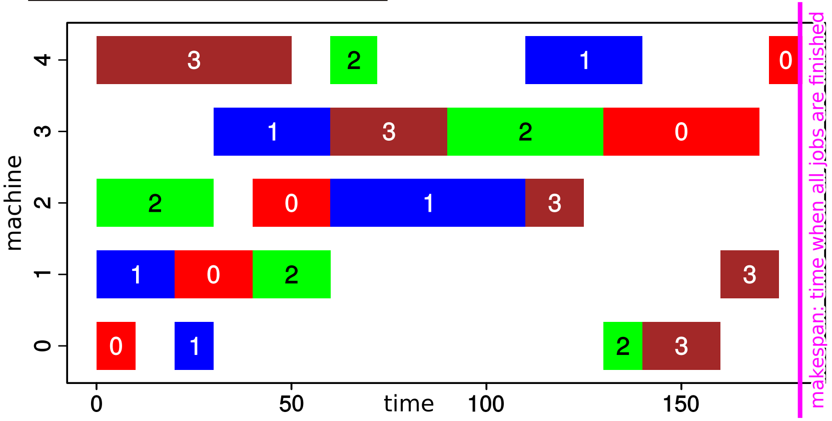


4 5

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

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)



Encoding and Solutions

- We encode solutions as permutations with repetitions, i.e., where each of the n job IDs appears m times in a linear string[5–7].
- The strings are processed from front to end to obtain a Gantt chart.

Encoding and Solutions

- We encode solutions as permutations with repetitions, i.e., where each of the n job IDs appears m times in a linear string[5–7].
- The strings are processed from front to end to obtain a Gantt chart.
- Unary search operator $move$ picks two random indices in the string with different job IDs and swaps these IDs.

Encoding and Solutions

- We encode solutions as permutations with repetitions, i.e., where each of the n job IDs appears m times in a linear string[5–7].
- The strings are processed from front to end to obtain a Gantt chart.
- Unary search operator $move$ picks two random indices in the string with different job IDs and swaps these IDs.
- All of this is fairly standard ($\geq$ 27 year old stuff).

Frequency Fitness Assignment (FFA)

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution, remember best-so-far solution.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution, remember best-so-far solution, and iteratively create modified copies of it using $move$.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution, remember best-so-far solution, and iteratively create modified copies of it using $move$.
- Most fundamental concept in optimization: Better solutions are preferred.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution, remember best-so-far solution, and iteratively create modified copies of it using $move$.
- Most fundamental concept in optimization: Better solutions are preferred.
- Some techniques like Simulated Annealing, Tabu Search, or GAs with Sharing and Niching have methods to prevent premature convergence by sometimes accepting worse solutions.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution, remember best-so-far solution, and iteratively create modified copies of it using $move$.
- Most fundamental concept in optimization: Better solutions are preferred.
- Some techniques like Simulated Annealing, Tabu Search, or GAs with Sharing and Niching have methods to prevent premature convergence by sometimes accepting worse solutions ... but in the core, they almost all of the time apply this bias towards better solutions.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution, remember best-so-far solution, and iteratively create modified copies of it using $move$.
- Most fundamental concept in optimization: Better solutions are preferred.
- Some techniques like Simulated Annealing, Tabu Search, or GAs with Sharing and Niching have methods to prevent premature convergence by sometimes accepting worse solutions ... but in the core, they almost all of the time apply this bias towards better solutions.
- Only random sampling, random walks, and exhaustive enumeration are free of this bias.

Solving Optimization Problems

- Input: objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ that maps the elements of a solution space $\mathcal{X}$ to a subset $\mathcal{Y} \subseteq \mathcal{R}$ of the real numbers $\mathcal{R}$.
- How do we find good solution?
- Start at a random solution, remember best-so-far solution, and iteratively create modified copies of it using $move$.
- Most fundamental concept in optimization: Better solutions are preferred.
- Some techniques like Simulated Annealing, Tabu Search, or GAs with Sharing and Niching have methods to prevent premature convergence by sometimes accepting worse solutions . . . but in the core, they almost all of the time apply this bias towards better solutions.
- Only random sampling, random walks, and exhaustive enumeration are free of this bias. . . and they are not good optimization methods.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.
- Idea: The fitness $H[y]$ corresponding to an objective value $y \in \mathcal{Y}$ is the number of times that y has been seen during the search so far.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.
- Idea: The fitness $H[y]$ corresponding to an objective value $y \in \mathcal{Y}$ is the number of times that y has been seen during the search so far.
- This mapping is dynamic, as the encounter frequencies obviously increase over time.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.
- Idea: The fitness $H[y]$ corresponding to an objective value $y \in \mathcal{Y}$ is the number of times that y has been seen during the search so far.
- This mapping is dynamic, as the encounter frequencies obviously increase over time.
- The frequency fitness only depends on the identity of the objective values.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.
- Idea: The fitness $H[y]$ corresponding to an objective value $y \in \mathcal{Y}$ is the number of times that y has been seen during the search so far.
- This mapping is dynamic, as the encounter frequencies obviously increase over time.
- The frequency fitness only depends on the identity of the objective values.
- We are not looking for better solutions anymore.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.
- Idea: The fitness $H[y]$ corresponding to an objective value $y \in \mathcal{Y}$ is the number of times that y has been seen during the search so far.
- This mapping is dynamic, as the encounter frequencies obviously increase over time.
- The frequency fitness only depends on the identity of the objective values.
- We are not looking for better solutions anymore.
- We are looking for solutions with harder-to-find objective values.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.
- Idea: The fitness $H[y]$ corresponding to an objective value $y \in \mathcal{Y}$ is the number of times that y has been seen during the search so far.
- This mapping is dynamic, as the encounter frequencies obviously increase over time.
- The frequency fitness only depends on the identity of the objective values.
- We are not looking for better solutions anymore.
- We are looking for solutions with harder-to-find objective values.
- $\Longrightarrow$ Algorithms basing all decisions on FFA are not biased towards better solutions.

Frequency Fitness Assignment (FFA)

- FFA is an algorithm module that can be plugged into arbitrary optimization methods.
- It is a fitness assignment process that translates objective values $y \in \mathcal{Y}$ to fitness values $H[y]$ that are used in comparisons.
- Idea: The fitness $H[y]$ corresponding to an objective value $y \in \mathcal{Y}$ is the number of times that y has been seen during the search so far.
- This mapping is dynamic, as the encounter frequencies obviously increase over time.
- The frequency fitness only depends on the identity of the objective values.
- We are not looking for better solutions anymore.
- We are looking for solutions with harder-to-find objective values.
- $\Longrightarrow$ Algorithms basing all decisions on FFA are not biased towards better solutions.
- Can optimization without bias for better solutions even work?

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)
```

(a) (1+1)-EA

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$;







```

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
```

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
```

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$;




```

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
**proc** (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    **while** $\neg$ terminate **do**
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
```

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
 1: proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)
 2:
 3:     randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
 4:
 5:     while $\neg$ terminate do
 6:         $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
 7:
 8:
 9:         if $y_n \leq y_c$ then
10:
```

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$;

```

(a) (1+1)-EA

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

```

(a) (1+1)-EA

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
**proc** (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    **while** $\neg$ terminate **do**
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        **if** $y_n \leq y_c$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    **return** $(x_c, y_c)$;
```

(a) (1+1)-EA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
1:  proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)
2:
3:      randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
4:
5:      while $\neg$ terminate do
6:          $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
7:
8:
9:          if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
10:     return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
1:  proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
2:      $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
3:      randomly sample $x_c$ from $\mathcal{X}$;
4:
5:
6:
7:
8:
9:
10:
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
**proc** (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    **while** $\neg$ terminate **do**
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        **if** $y_n \leq y_c$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    **return** $(x_c, y_c)$;
```

(a) (1+1)-EA

```
**proc** (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while ¬ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while ¬ terminate do
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;




```

(b) (1+1)-FEA

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$;


```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;


```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then

```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$;

```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
**proc** (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    **while** $\neg$ terminate **do**
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        **if** $y_n \leq y_c$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    **return** $(x_c, y_c)$;
```

(a) (1+1)-EA

```
**proc** (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    **while** $\neg$ terminate **do**
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        **if** $H[y_n] \leq H[y_c]$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;

```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
**proc** (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    **while** $\neg$ terminate **do**
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        **if** $y_n \leq y_c$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    **return** $(x_c, y_c)$;
```

(a) (1+1)-EA

```
**proc** (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    **while** $\neg$ terminate **do**
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        **if** $H[y_n] \leq H[y_c]$ **then** $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    **return** $(x_b, y_b)$;
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while ¬ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return ($x_c, y_c$);
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$;
    while ¬ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return ($x_b, y_b$);
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while ¬ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
    while ¬ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;

        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_b, y_b)$;
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return ($x_c, y_c$);
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
        if $y_n < y_b$ then
        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return ($x_b, y_b$);
```

(b) (1+1)-FEA

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
        if $y_n < y_b$ then $x_b \leftarrow x_n$;
        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_b, y_b)$;
```

(b) (1+1)-FEA

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
        if $y_n < y_b$ then $x_b \leftarrow x_n$; $y_b \leftarrow y_n$;
        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_b, y_b)$;
```

(b) (1+1)-FEA

(1+1)-EA **and** (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
        if $y_n < y_b$ then $x_b \leftarrow x_n$; $y_b \leftarrow y_n$;
        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_b, y_b)$;
```

(b) (1+1)-FEA

- Search is driven entirely by frequency H.

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)

    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;

    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;


        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
        if $y_n < y_b$ then $x_b \leftarrow x_n$; $y_b \leftarrow y_n$;
        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_b, y_b)$;
```

(b) (1+1)-FEA

- Search is driven entirely by frequency H.
- Whether a solution is better or worse plays no role in the algorithm's decisions.

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
        if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
    $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
    randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
    $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
    while $\neg$ terminate do
        $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
        if $y_n < y_b$ then $x_b \leftarrow x_n$; $y_b \leftarrow y_n$;
        $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
        if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
    return $(x_b, y_b)$;
```

(b) (1+1)-FEA

- Search is driven entirely by frequency H.
- Whether a solution is better or worse plays no role in the algorithm's decisions.
- Can (1+1)-FEA even work?

(1+1)-EA and (1+1)-FEA

- We plug FFA into the simplest possible evolutionary algorithm, the (1+1)-EA and we get the (1+1)-FEA.

```
 1: proc (1+1)-EA($f : \mathcal{X} \mapsto 0..UB$)
 2:
 3:     randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
 4:
 5:     while ¬ terminate do
 6:         $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
 7:
 8:
 9:         if $y_n \leq y_c$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
10:     return $(x_c, y_c)$;
```

(a) (1+1)-EA

```
 1: proc (1+1)-FEA($f : \mathcal{X} \mapsto 0..UB$)
 2:     $H[0..UB] \leftarrow (0, 0, \cdots, 0)$;
 3:     randomly sample $x_c$ from $\mathcal{X}$; $y_c \leftarrow f(x_c)$;
 4:     $x_b \leftarrow x_c$; $y_b \leftarrow y_c$;
 5:     while ¬ terminate do
 6:         $x_n \leftarrow move(x_c)$; $y_n \leftarrow f(x_n)$;
 7:         if $y_n < y_b$ then $x_b \leftarrow x_n$; $y_b \leftarrow y_n$;
 8:         $H[y_c] \leftarrow H[y_c] + 1$; $H[y_n] \leftarrow H[y_n] + 1$;
 9:         if $H[y_n] \leq H[y_c]$ then $x_c \leftarrow x_n$; $y_c \leftarrow y_n$;
10:     return $(x_b, y_b)$;
```

(b) (1+1)-FEA

- Search is driven entirely by frequency H.
- Whether a solution is better or worse plays no role in the algorithm's decisions.
- Can (1+1)-FEA even work on a hard problem like the JSSP?

Experiment and Results

Experiment

- We use the 8 most common sets: `abz`[8], `dmu`[9], `ft`[10], `la`[11], `orb`[12], `swv`[13], `ta`[14], and `yn`[15].

Experiment

- We use the 8 most common sets: `abz`[8], `dmu`[9], `ft`[10], `la`[11], `orb`[12], `swv`[13], `ta`[14], and `yn`[15].
- We do 5 runs for each algorithm on each of these 242 instances.

Experiment

- We use the 8 most common sets: `abz`[8], `dmu`[9], `ft`[10], `la`[11], `orb`[12], `swv`[13], `ta`[14], and `yn`[15].
- We do 5 runs for each algorithm on each of these 242 instances.
- Runtime limit: 2^{30} FEs ($\approx 10^9$ FEs)

Results

inst	EA			FEA			FEA vs. EA		
	best	*mean*	*conv*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
`abz`	1	0	5	5	5	0	-0.3%	-1.2%	22.0
`dmu`	65	60	79	20	23	1	3.8%	3.4%	4.0
`ft`	2	1	2	3	3	1	-0.2%	-1.3%	5.0
`la`	32	23	38	35	37	2	-0.1%	-0.5%	32.0
`orb`	2	0	8	10	10	2	-1.5%	-3.6%	13.0
`swv`	12	10	19	13	15	1	0.1%	-0.6%	8.6
`ta`	59	52	80	29	33	0	1.1%	1.0%	20.0
`yn`	1	0	4	3	4	0	-0.5%	-0.6%	4.0

best (*mean*): number of instances algorithm reached the best (best average) solution

conv: number of instances algorithm reached end solution (stopped improving) fastest

Results

inst	EA			FEA			FEA vs. EA		
	best	*mean*	*conv*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
`abz`	1	0	5	5	5	0	-0.3%	-1.2%	22.0
`dmu`	65	60	79	20	23	1	3.8%	3.4%	4.0
`ft`	2	1	2	3	3	1	-0.2%	-1.3%	5.0
`la`	32	23	38	35	37	2	-0.1%	-0.5%	32.0
`orb`	2	0	8	10	10	2	-1.5%	-3.6%	13.0
`swv`	12	10	19	13	15	1	0.1%	-0.6%	8.6
`ta`	59	52	80	29	33	0	1.1%	1.0%	20.0
`yn`	1	0	4	3	4	0	-0.5%	-0.6%	4.0

best (*mean*): number of instances algorithm reached the best (best average) solution

conv: number of instances algorithm reached end solution (stopped improving) fastest

- (1+1)-FEA reaches best mean results in 6/8 benchmark sets and best results in 5/8.

Results

inst	EA			FEA			FEA vs. EA		
	best	*mean*	*conv*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
abz	1	0	5	5	5	0	-0.3%	-1.2%	22.0
dmu	65	60	79	20	23	1	3.8%	3.4%	4.0
ft	2	1	2	3	3	1	-0.2%	-1.3%	5.0
la	32	23	38	35	37	2	-0.1%	-0.5%	32.0
orb	2	0	8	10	10	2	-1.5%	-3.6%	13.0
swv	12	10	19	13	15	1	0.1%	-0.6%	8.6
ta	59	52	80	29	33	0	1.1%	1.0%	20.0
yn	1	0	4	3	4	0	-0.5%	-0.6%	4.0

best (*mean*): number of instances algorithm reached the best (best average) solution

conv: number of instances algorithm reached end solution (stopped improving) fastest

- (1+1)-FEA reaches best mean results in 6/8 benchmark sets and best results in 5/8.
- (1+1)-FEA always converge slower.

Results

inst	EA			FEA			FEA vs. EA		
	best	*mean*	*conv*	*best*	*mean*	*conv*	*best*	*mean*	*conv*
abz	1	0	5	5	5	0	-0.3%	-1.2%	22.0
dmu	65	60	79	20	23	1	3.8%	3.4%	4.0
ft	2	1	2	3	3	1	-0.2%	-1.3%	5.0
la	32	23	38	35	37	2	-0.1%	-0.5%	32.0
orb	2	0	8	10	10	2	-1.5%	-3.6%	13.0
swv	12	10	19	13	15	1	0.1%	-0.6%	8.6
ta	59	52	80	29	33	0	1.1%	1.0%	20.0
yn	1	0	4	3	4	0	-0.5%	-0.6%	4.0

best (*mean*): number of instances algorithm reached the best (best average) solution

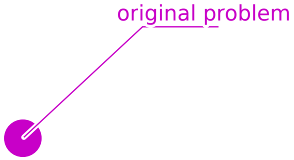
conv: number of instances algorithm reached end solution (stopped improving) fastest

- (1+1)-FEA is at least as good and sometimes better than *all* mean and best results in [16] (2010), aLSGA [17] (2015), the EAS in [18] (2013), all GAs in [19] (2014), the GWO in [20] (2018), SAFA [21] (2018), HBFO[22] (2012), and all algorithms in[23] (2018).

Invariances

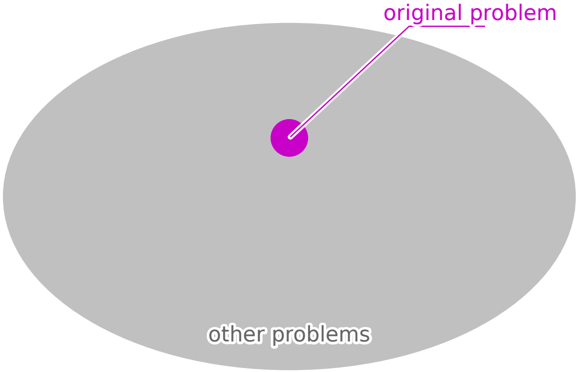
How should a good optimization algorithm behave?

- It should perform well.



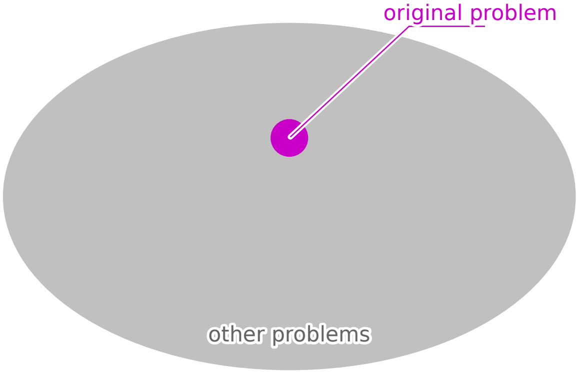
How should a good optimization algorithm behave?

- It should perform well.
- The performance observed on some problems should carry over to other problems.



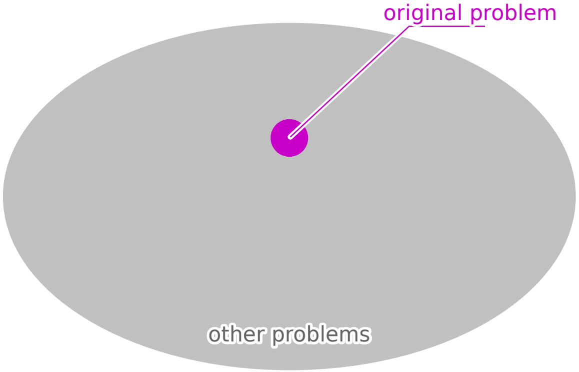
How should a good optimization algorithm behave?

- It should perform well on benchmarks.
- The performance observed on some (benchmark) problems should carry over to other (real-world) problems.



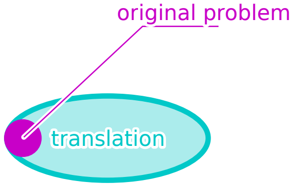
How should a good optimization algorithm behave?

- It should perform well on benchmarks.
- The performance observed on some (benchmark) problems should carry over to other (real-world) problems.
- The algorithm should be invariant under transformations of the original problem.



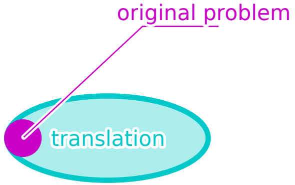
How should a good optimization algorithm behave?

- The algorithm should behave the same if we add an offset to the objective values.



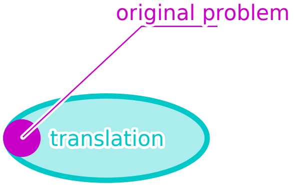
How should a good optimization algorithm behave?

- The algorithm should behave the same if we add an offset to the objective values.
- Then, we cannot use proportions of objective values in the decisions.



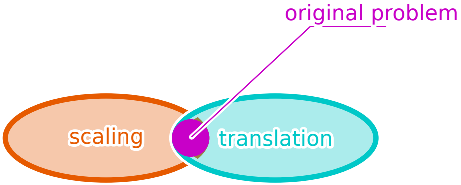
How should a good optimization algorithm behave?

- The algorithm should behave the same if we add an offset to the objective values.
- Then, we cannot use proportions of objective values in the decisions.
- The Genetic Algorithm with Roulette Wheel Selection is not translation invariant.



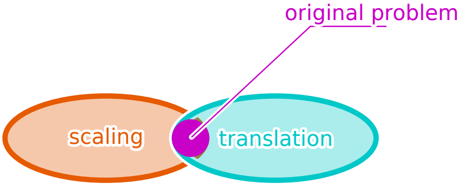
How should a good optimization algorithm behave?

- The algorithm should behave the same if we multiply the objective values by a positive number.



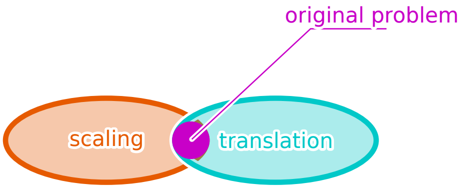
How should a good optimization algorithm behave?

- The algorithm should behave the same if we multiply the objective values by a positive number.
- Then, we cannot use absolute differences of objective values in the decisions.



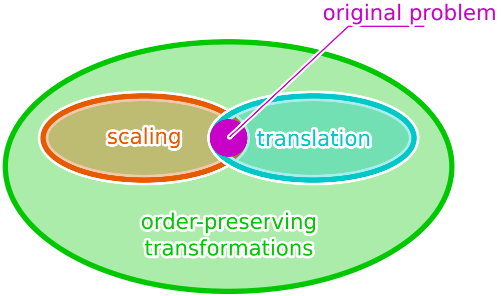
How should a good optimization algorithm behave?

- The algorithm should behave the same if we multiply the objective values by a positive number.
- Then, we cannot use absolute differences of objective values in the decisions.
- Simulated Annealing is not scale invariant.



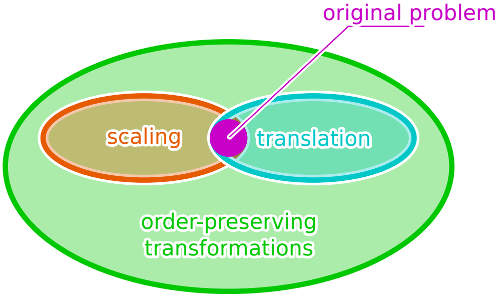
How should a good optimization algorithm behave?

- The algorithm should behave the same on problems where the order of the objective values is the same.



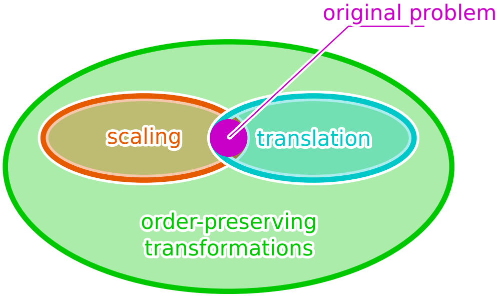
How should a good optimization algorithm behave?

- The algorithm should behave the same on problems where the order of the objective values is the same.
- Then, we can only compare whether solutions are better, worse, or as same as good.



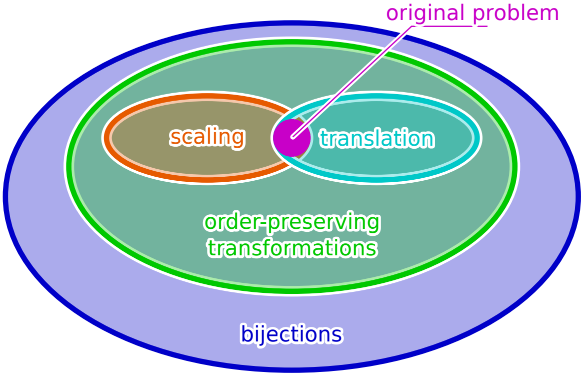
How should a good optimization algorithm behave?

- The algorithm should behave the same on problems where the order of the objective values is the same.
- Then, we can only compare whether solutions are better, worse, or as same as good.
- The (1+1)-EA is invariant under order-preserving transformations.



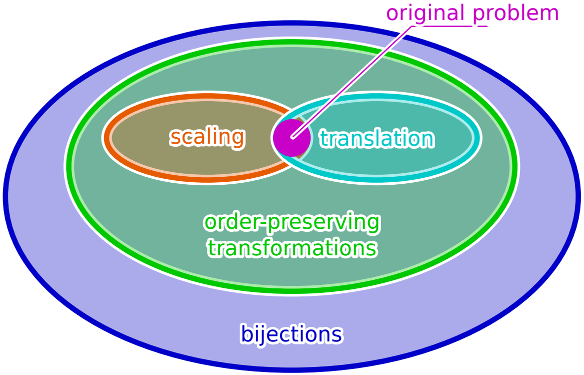
How should a good optimization algorithm behave?

- Is it possible for algorithms to behave the same on all bijective transformations of the objective values?



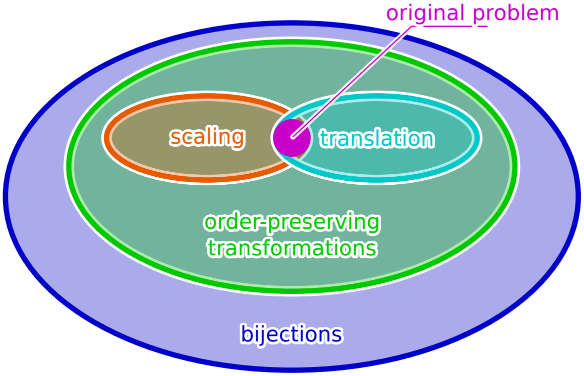
How should a good optimization algorithm behave?

- Is it possible for algorithms to behave the same on all bijective transformations of the objective values?
- Then, we can only compare if two solutions have same objective value and nothing else.



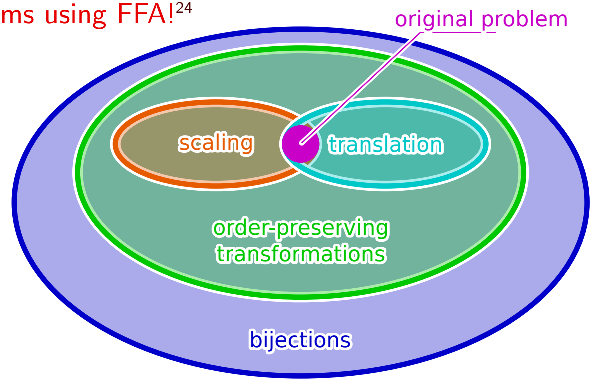
How should a good optimization algorithm behave?

- Is it possible for algorithms to behave the same on all bijective transformations of the objective values?
- Then, we can only compare if two solutions have same objective value and nothing else.
- The only algorithms with this feature are random walks, random sampling, and exhaustive enumeration.



How should a good optimization algorithm behave?

- Is it possible for algorithms to behave the same on all bijective transformations of the objective values?
- Then, we can only compare if two solutions have same objective value and nothing else.
- The only algorithms with this feature are random walks, random sampling, and exhaustive enumeration.
- . . . and all algorithms using FFA![24]



How should a good optimization algorithm behave?

- Is it possible for algorithms to behave the same on all bijective transformations of the objective values?
- Then, we can only compare if two solutions have same objective value and nothing else.
- The only algorithms with this feature are random walks, random sampling, and exhaustive enumeration.
- . . . and all algorithms using FFA![24]
- To give you a taste: encryption is a bijection, too. . .

Conclusions

Conclusions

- We have plugged FFA into the simplest possible EA and applied it to the JSSP.

Conclusions

- We have plugged FFA into the simplest possible EA and applied it to the JSSP.
- (1+1)-FEA is an algorithm that does not prefer better solutions over worse ones.

Conclusions

- We have plugged FFA into the simplest possible EA and applied it to the JSSP.
- (1+1)-FEA is an algorithm that does not prefer better solutions over worse ones.
- It performs surprisingly well on this $\mathcal{NP}$-hard optimization problem.

Conclusions

- We have plugged FFA into the simplest possible EA and applied it to the JSSP.
- (1+1)-FEA is an algorithm that does not prefer better solutions over worse ones.
- It performs surprisingly well on this $\mathcal{NP}$-hard optimization problem.
- On Max-Sat, (1+1)-FEA is very significantly faster than (1+1)-EA[24].

Conclusions

- We have plugged FFA into the simplest possible EA and applied it to the JSSP.
- (1+1)-FEA is an algorithm that does not prefer better solutions over worse ones.
- It performs surprisingly well on this $\mathcal{NP}$-hard optimization problem.
- On Max-Sat, (1+1)-FEA is very significantly faster than (1+1)-EA[24].
- So there are now two classical $\mathcal{NP}$-hard problems where optimization without bias for good solutions works!

Conclusions

- We have plugged FFA into the simplest possible EA and applied it to the JSSP.
- (1+1)-FEA is an algorithm that does not prefer better solutions over worse ones.
- It performs surprisingly well on this $\mathcal{NP}$-hard optimization problem.
- On Max-Sat, (1+1)-FEA is very significantly faster than (1+1)-EA[24].
- So there are now two classical $\mathcal{NP}$-hard problems where optimization without bias for good solutions works!
- Interesting: All algorithms using FFA are invariant under all bijections of the objective function value.

谢谢

Thank you

References I

1. Thomas Weise. *An Introduction to Optimization Algorithms*. Institute of Applied Optimization (IAO) [应用优化研究所] of the School of Artificial Intelligence and Big Data [人工智能与大数据学院] of Hefei University [合肥学院], Hefei [合肥市], Anhui [安徽省], China [中国], 2018–2020. URL http://thomasweise.github.io/aitoa/.
2. Thomas Weise. *Global Optimization Algorithms – Theory and Application*. it-weise.de (self-published), Germany, 2009. URL http://www.it-weise.de/projects/book.pdf.
3. Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In Stephen C. Graves, Alexander Hendrik George Rinnooy Kan, and Paul H. Zipkin, editors, *Handbook of Operations Research and Management Science*, volume IV: Production Planning and Inventory, chapter 9, pages 445–522. North-Holland Scientific Publishers Ltd., Amsterdam, The Netherlands, 1993. doi:10.1016/S0927-0507(05)80189-6.
4. Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. In Ding-Zhu Du and Panos M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 1493–1641. Springer-Verlag US, Boston, MA, USA, 1998. ISBN 978-1-4613-7987-4. doi:10.1007/978-1-4613-0303-9_25. also pages 21–169 in volume 3/3 by Kluwer Academic Publishers.
5. Mitsuo Gen, Yasuhiro Tsujimura, and Erika Kubota. Solving job-shop scheduling problems by genetic algorithm. In *Humans, Information and Technology: Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics, October 2–5, 1994, San Antonio, TX, USA*, volume 2. IEEE, 1994. ISBN 0-7803-2129-4. doi:10.1109/ICSMC.1994.400072. URL http://read.pudn.com/downloads151/doc/658565/00400072.pdf.
6. Christian Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum (OR Spectrum)*, 17:87–92, June 1995. doi:10.1007/BF01719250. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.7392&type=pdf.
7. Guoyong Shi, Hitoshi Iima, and Nobuo Sannomiya. New encoding scheme for solving job shop problems by genetic algorithm. In *Proceedings of the 35th IEEE Conference on Decision and Control (CDC'96), December 11–13, 1996, Kobe, Japan*, volume 4, pages 4395–4400. IEEE, 1997. ISBN 0-7803-3590-2. doi:10.1109/CDC.1996.577484.
8. Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988. doi:10.1287/mnsc.34.3.391.
9. Ebru Demirkol, Sanjay V. Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research (EJOR)*, 109(1):137–141, August 1998. doi:10.1016/S0377-2217(97)00019-2.
10. Henry Fisher and Gerald L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In John F. Muth and Gerald L. Thompson, editors, *Industrial Scheduling*, chapter 3.2, pages 225–251. Prentice-Hall, Englewood Cliffs, NJ, USA, 1963.

References II

11. Stephen R. Lawrence. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*. PhD thesis, Graduate School of Industrial Administration (GSIA), Carnegie-Mellon University, Pittsburgh, PA, USA, 1984.
12. David Lee Applegate and William John Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, May 1991. doi:10.1287/ijoc.3.2.149. the JSSP instances used were generated in Bonn in 1986.
13. Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992. doi:10.1287/mnsc.38.10.1495.
14. Éric D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research (EJOR)*, 64(2): 278–285, January 1993. doi:10.1016/0377-2217(93)90182-M.
15. Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop instances. In Reinhard Männer and Bernard Manderick, editors, *Proceedings of Parallel Problem Solving from Nature 2 (PPSN II), September 28–30, 1992, Brussels, Belgium*, pages 281–290, Amsterdam, The Netherlands, 1992. Elsevier.
16. Tamer F. Abdelmaguid. Representations in genetic algorithm for the job shop scheduling problem: A computational study. *Journal of Software Engineering and Applications (JSEA)*, 3(12):1155–1162, December 2010. doi:10.4236/jsea.2010.312135. URL `http://www.scirp.org/journal/paperinformation.aspx?paperid=3561`.
17. Leila Asadzadeh. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Computers & Industrial Engineering*, 85:376–383, July 2015. doi:10.1016/j.cie.2015.04.006.
18. Edson Flórez, Wilfredo Gómez, and Lola Bautista. An ant colony optimization algorithm for job shop scheduling problem. Computing Research Repository (CoRR) abs/1309.5110, arxiv, 2013. URL `https://arxiv.org/pdf/1309.5110.pdf`.
19. Vedavyasrao Jorapur, V. S. Puranik, A. S. Deshpande, and M. R. Sharma. Comparative study of different representations in genetic algorithms for job shop scheduling problem. *Journal of Software Engineering and Applications (JSEA)*, 7(7): 571–580, June 2014. doi:10.4236/jsea.2014.77053. URL `http://www.scirp.org/journal/paperinformation.aspx?paperid=46670`.
20. Tianhua Jiang and Chao Zhang. Application of grey wolf optimization for solving combinatorial problems: Job shop and flexible job shop scheduling cases. *IEEE Access*, 6:26231–26240, May 2018. doi:10.1109/ACCESS.2018.2833552. URL `http://ieeexplore.ieee.org/document/8355479`.
21. Joss Miller-Todd, Kathleen Steinhöfel, and Patrick Veenstra. Firefly-inspired algorithm for job shop scheduling. In Hans-Joachim Böckenhauer, Dennis Komm, and Walter Unger, editors, *Adventures Between Lower Bounds and Higher Altitudes – Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, pages 423–433. Springer, 2018. doi:10.1007/978-3-319-98355-4_24.

References III

22. S. Narendhar and T. Amudha. A hybrid bacterial foraging algorithm for solving job shop scheduling problems. *Intl. Journal of Programming Languages and Applications*, 2(4):1–11, 2012. doi:10.5121/ijpla.2012.2401.
23. Shao-Juan Wang, Chun-Wei Tsai, and Ming-Chao Chiang. 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, November 5–8, 2018, Leuven, Belgium*, pages 119–126. Elsevier, 2018. doi:10.1016/j.procs.2018.10.157.
24. Thomas Weise, Zhize Wu, Xinlu Li, and Yan Chen. Frequency fitness assignment: Making optimization algorithms invariant under bijective transformations of the objective function value. *IEEE Transactions on Evolutionary Computation*, 25:307–319, April 2021. doi:10.1109/TEVC.2020.3032090.