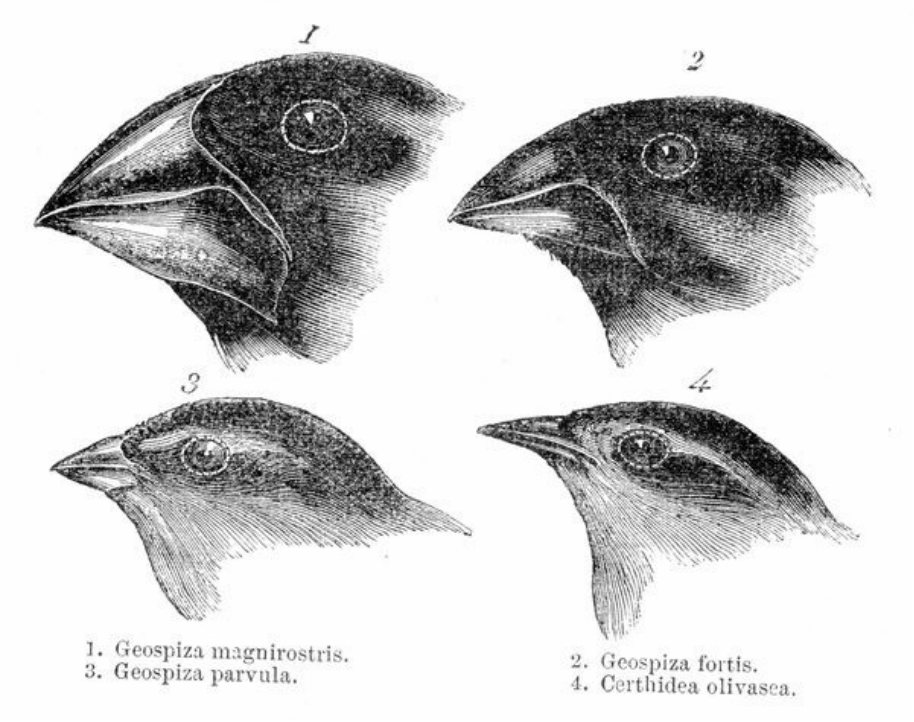
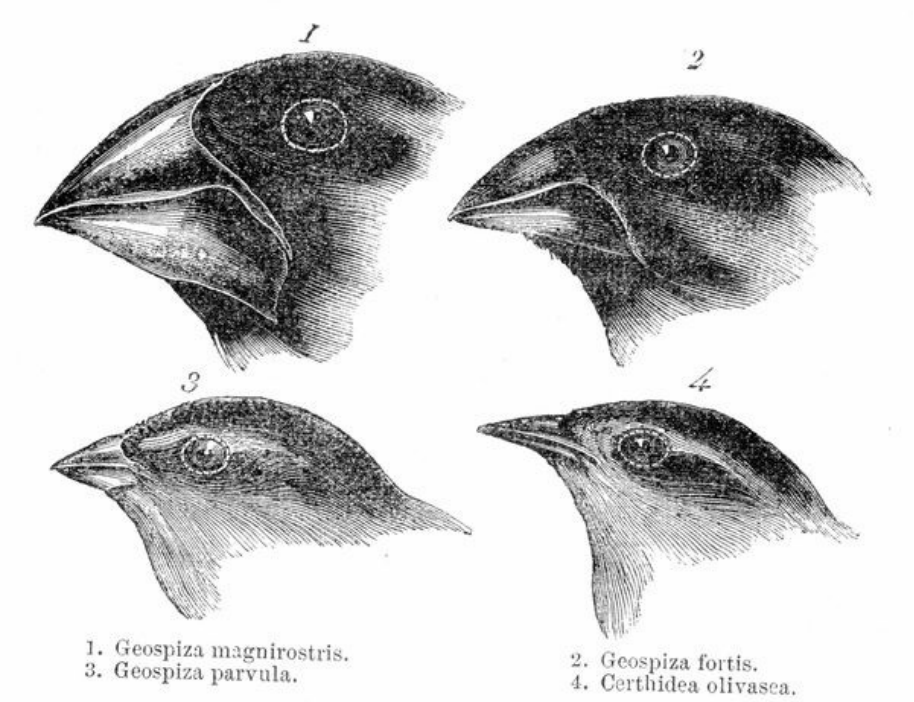
Genetic Algorithms

Brian Busemeyer
Algorithms interest group
08/2016

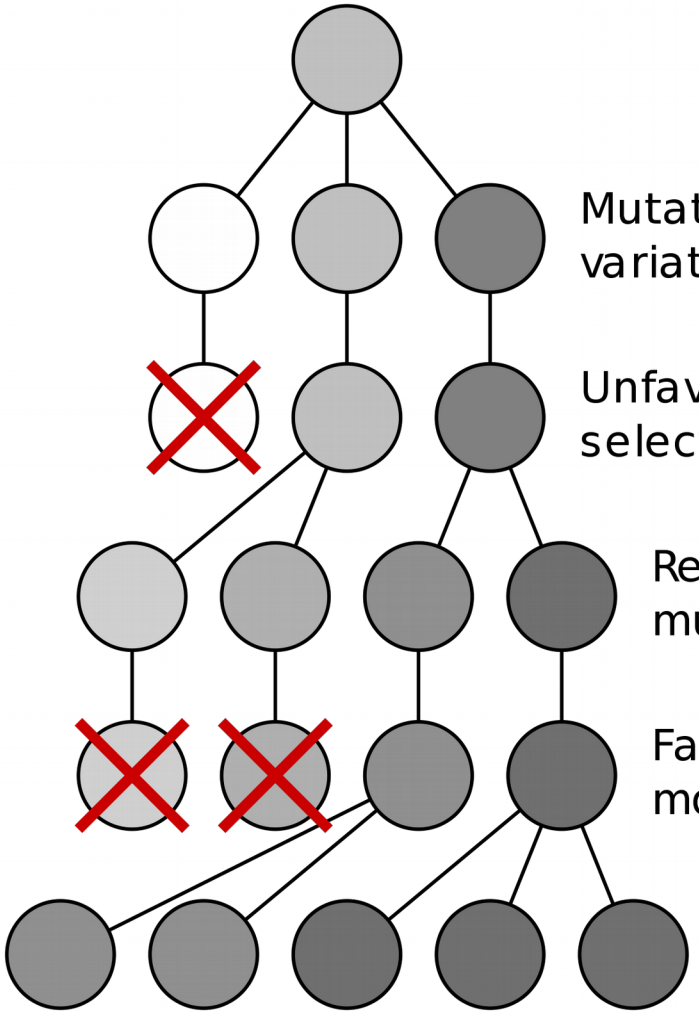


Introduction:

Nature is very good at optimizing over a huge parameter space!



Evolution in a nutshell.



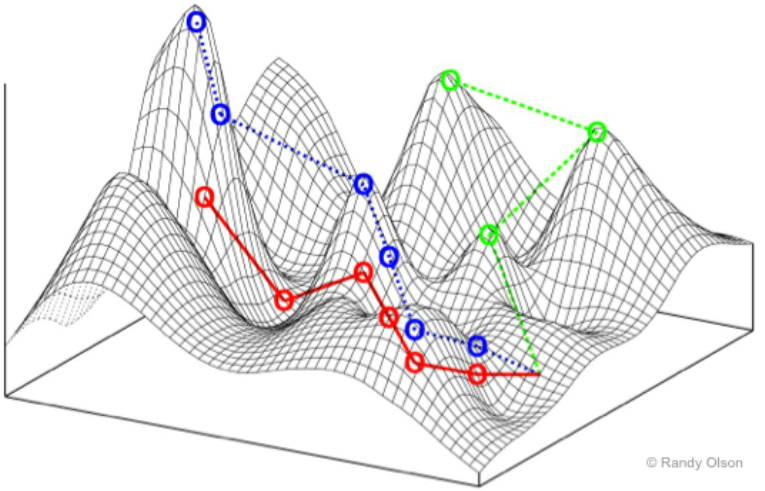
Mutation creates variation

Unfavorable mutations selected against

Reproduction and mutation occur

Favorable mutations more likely to survive

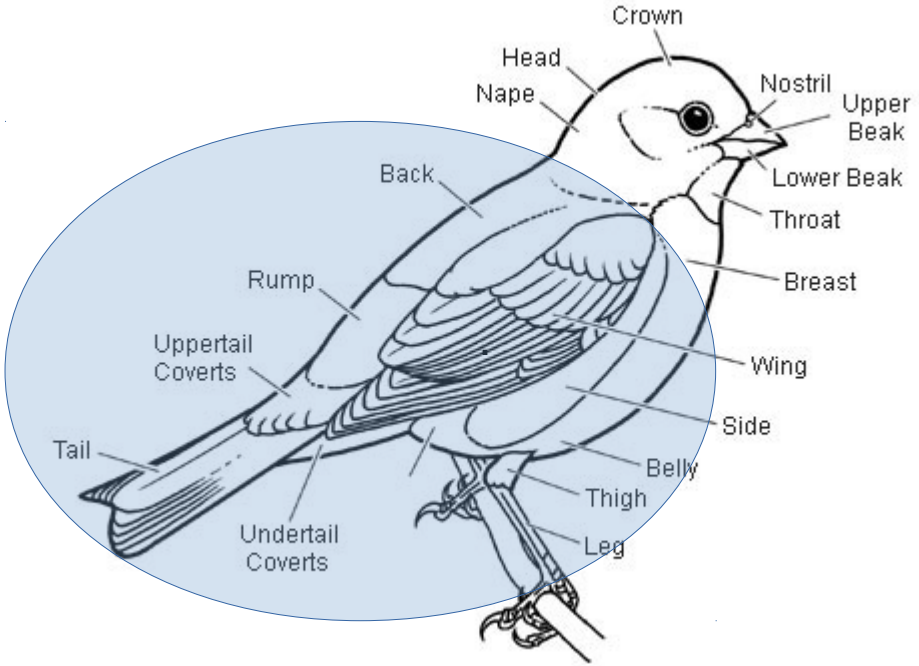
... and reproduce

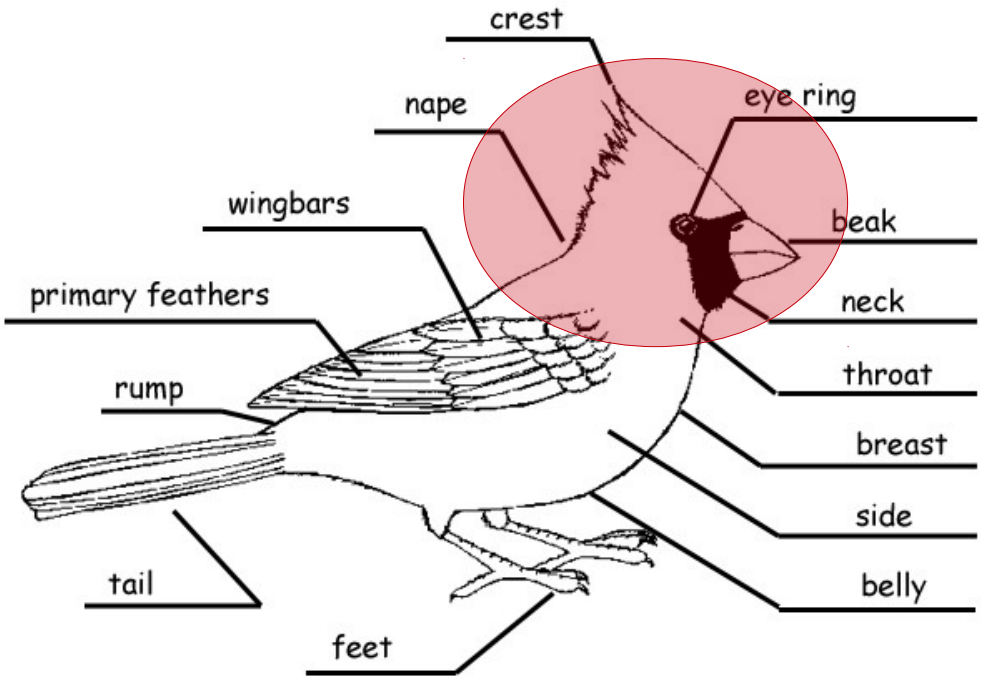


When does it work, and why?

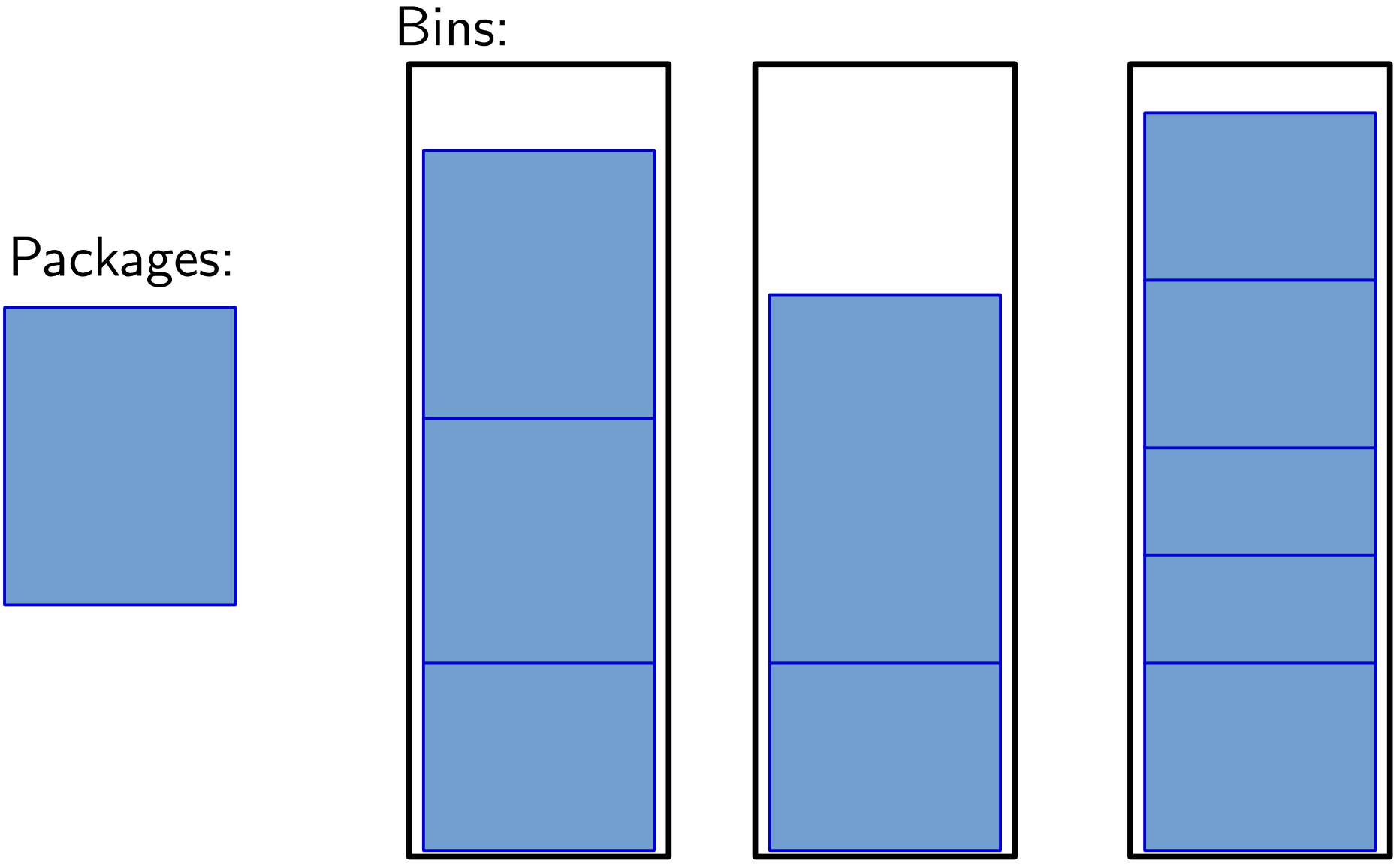
Building-block hypothesis:

- Don't optimize over all combinations
- Identify good "chunks"
- Construct better solutions from good partial solutions.





Example problem: bin packing (NP-hard).



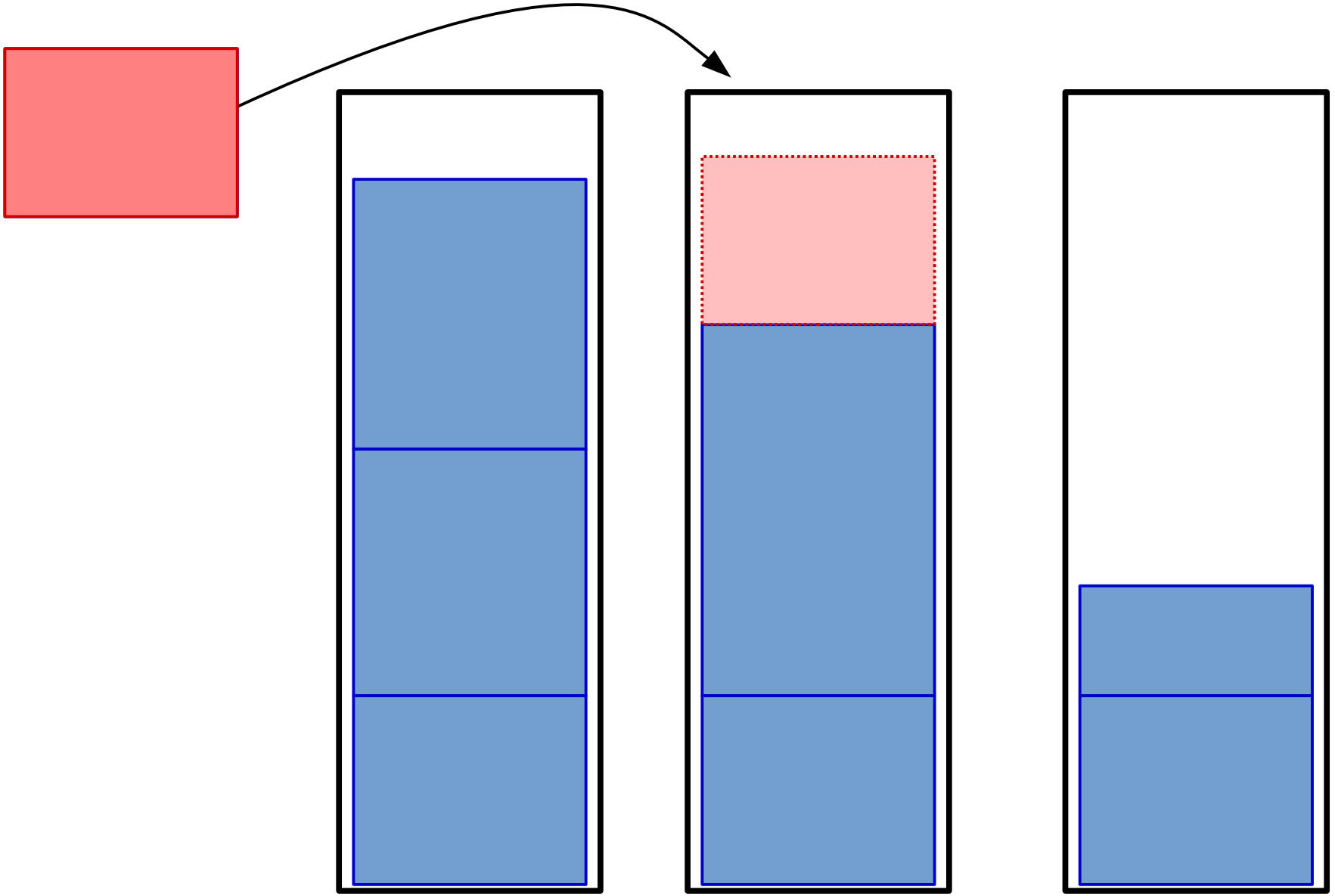
What's the least number of bins required for N packages?

Example problem: bin packing (NP-hard)

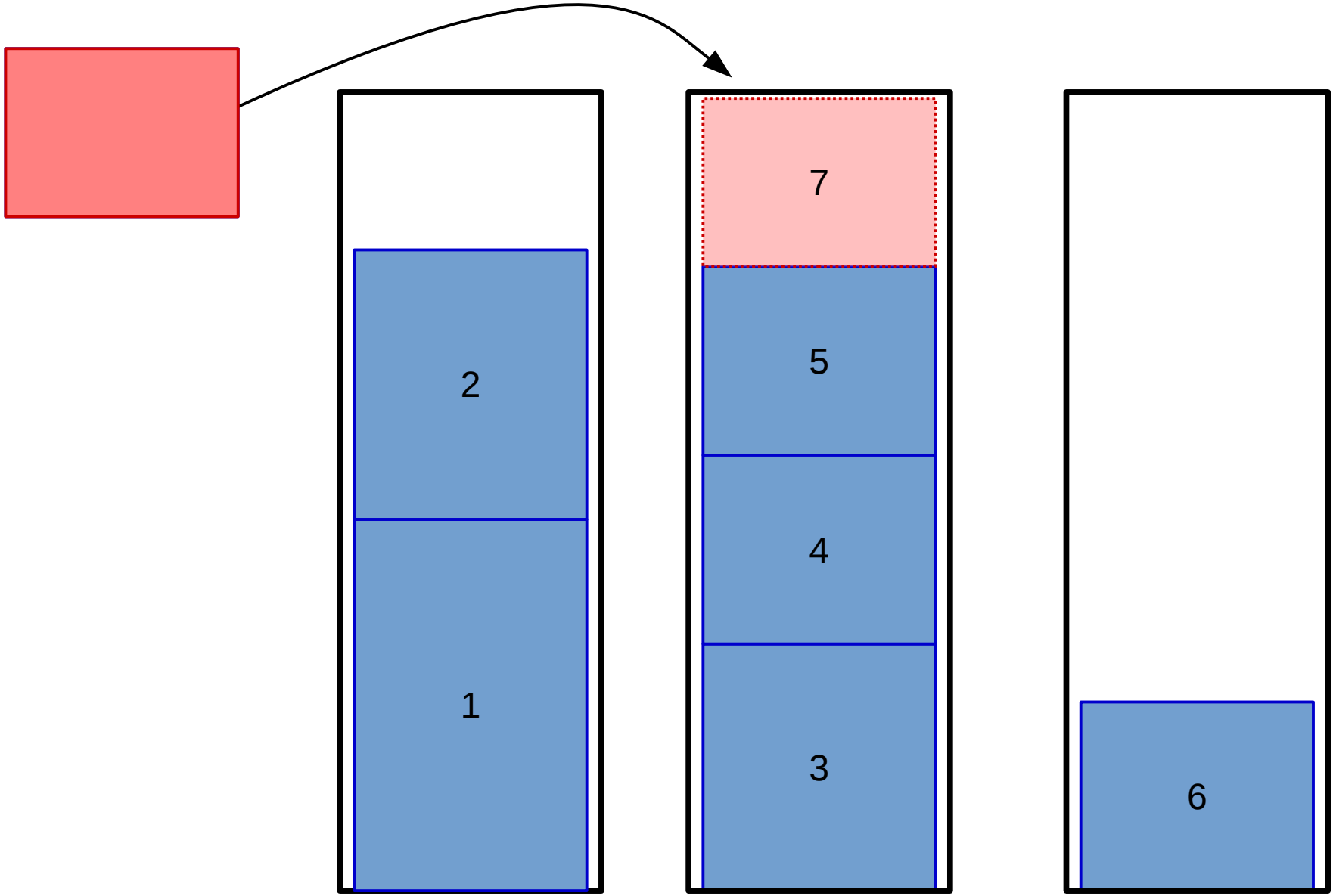
Important details:

- 1-d binpacking (only worry if package fits in 1 direction).
- Bins are all some fixed size $\equiv 1$.
- Package sizes randomly sampled $U(0, M)$, $M < 1$.
- Figure of merit: $V_{\text{packages}}/V_{\text{bins}}$ "filled fraction".

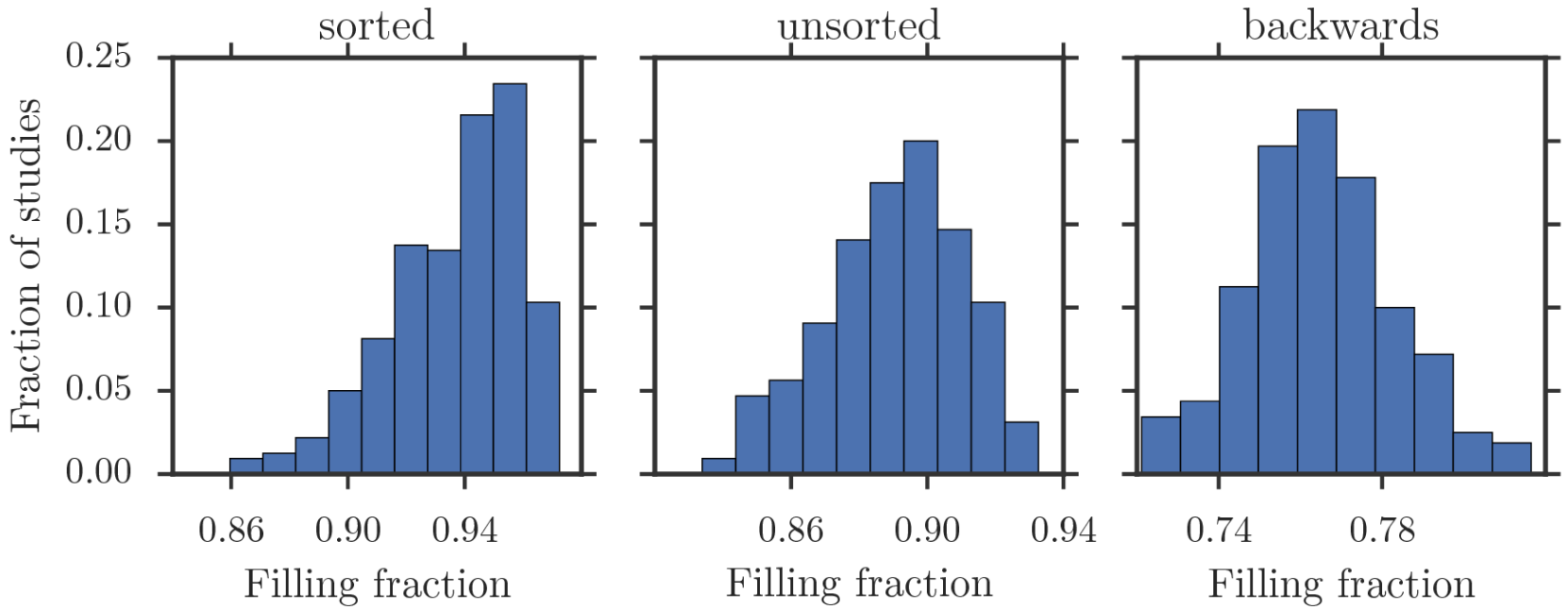
Greedy solution: put it in the first bin it fits!



Improvement: sort, then greedy pack.

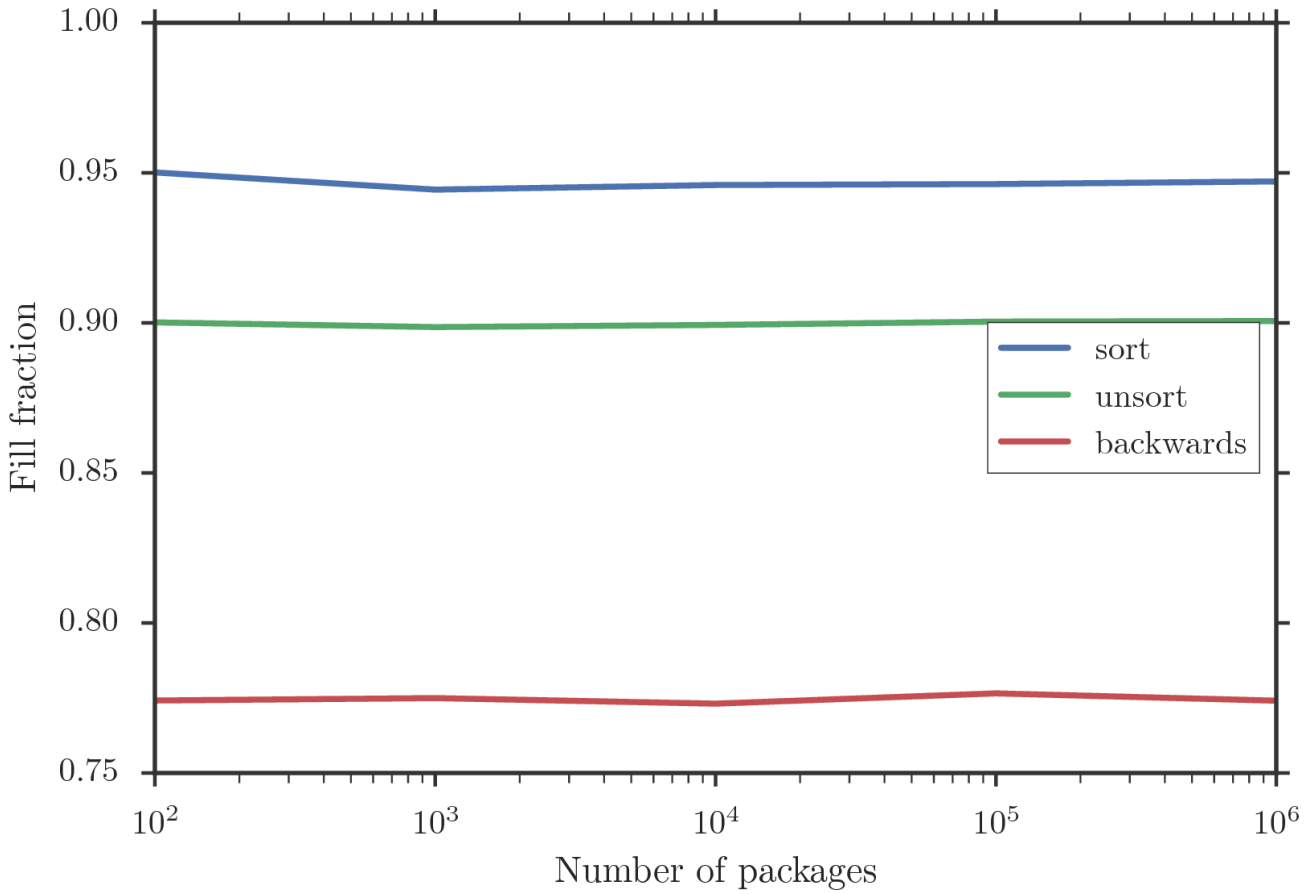


Statistics of greedy solutions.



- Sorting improves performance slightly.
- Sorting backwards hinders performance dramatically.
- Generally, greedy packing leaves roughly 10% unfilled space.

Performance doesn’t depend on size!



Genetic algorithm:

1. Initialize populations.
2. Cycle:
   (a) Measure fitness of all solutions.
   (b) Sample solutions weighted by fitness for breeding.
   (c) Mutate all new solutions.

Genetic algorithm:

1. Initialize populations.
2. Cycle:
   (a) Measure fitness of all solutions.
   (b) Sample solutions weighted by fitness for breeding.
   (c) Mutate all new solutions.

Bin packing implementation:

Greedy solutions, packing in random orders and in order of package size

Genetic algorithm:

1. Initialize populations.
2. Cycle:
   (a) Measure fitness of all solutions.
   (b) Sample solutions weighted by fitness for breeding.
   (c) Mutate all new solutions.

Bin packing implementation:

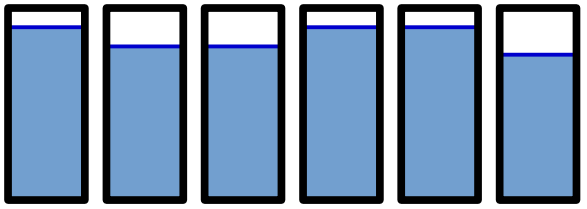
Figure of merit: $\frac{V_{\text{packages}}}{V_{\text{bins}}}$

Genetic algorithm:

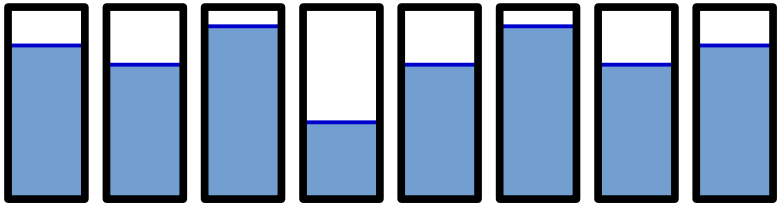
1. Initialize populations.
2. Cycle:
   (a) Measure fitness of all solutions.
   (b) Sample solutions weighted by fitness for breeding.
   (c) Mutate all new solutions.

Bin packing implementation:

Solution 1



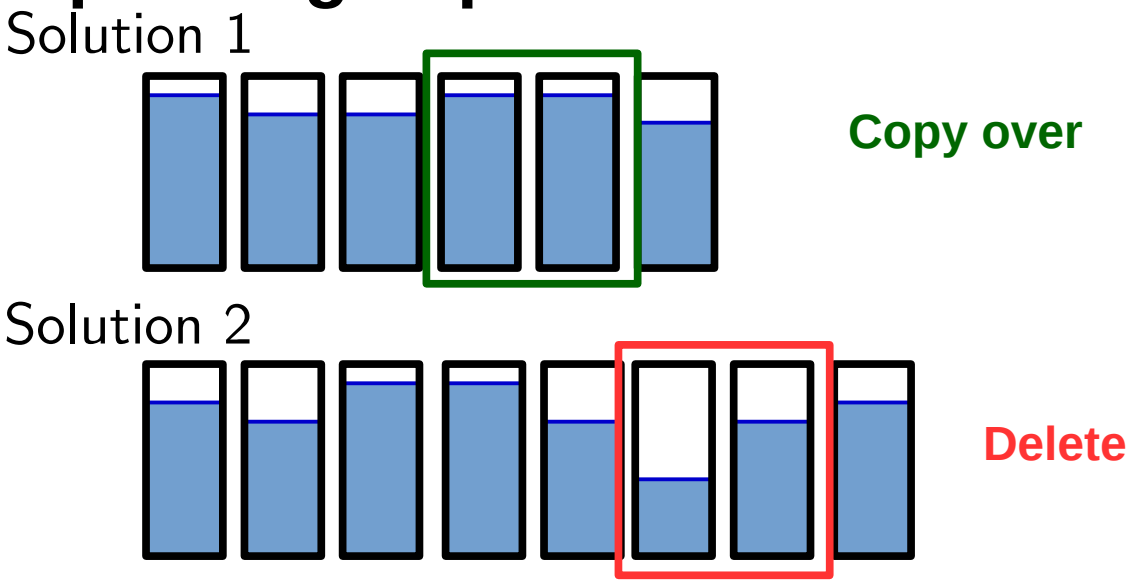
Solution 2



Genetic algorithm:

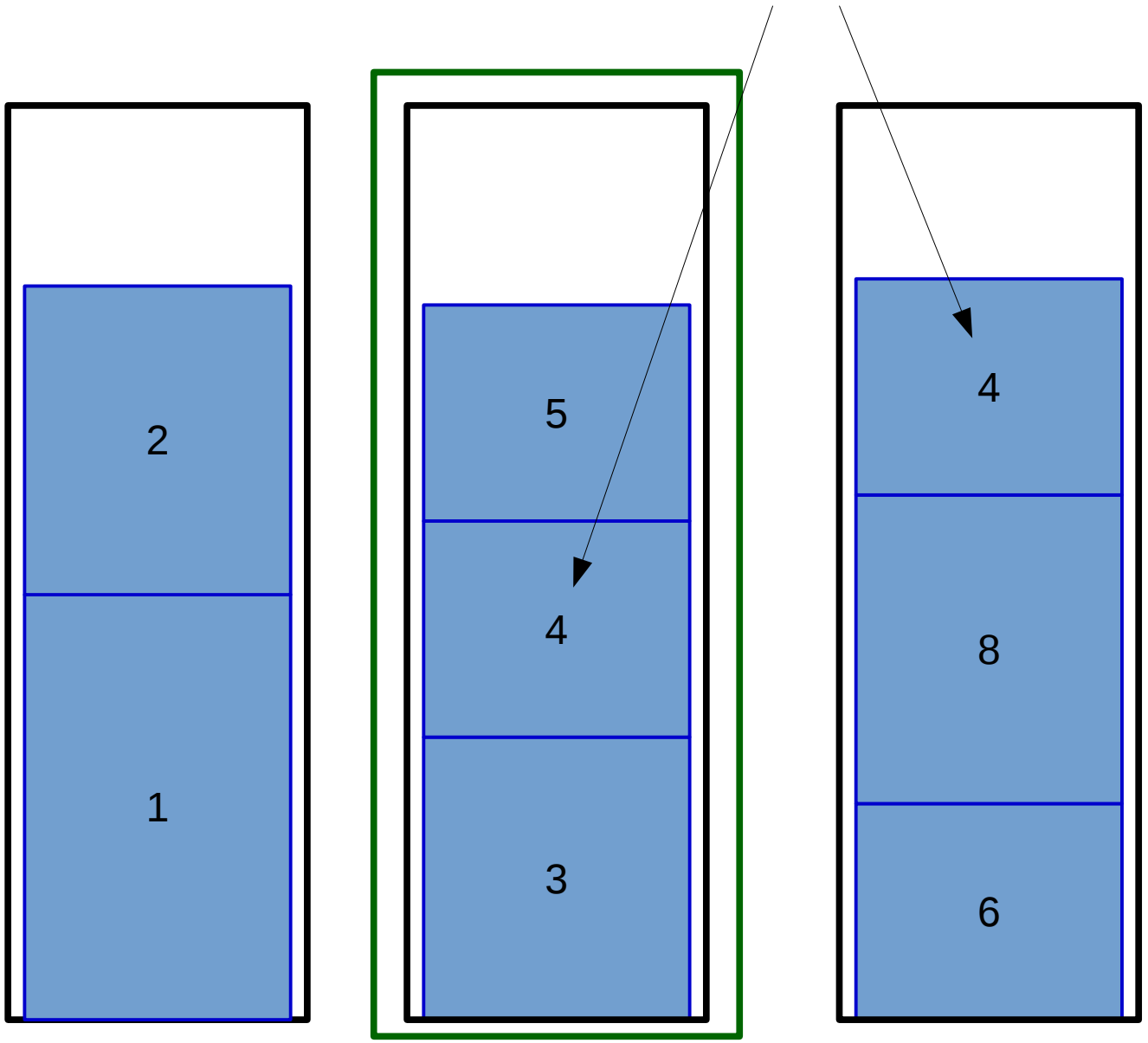
1. Initialize populations.
2. Cycle:
   (a) Measure fitness of all solutions.
   (b) Sample solutions weighted by fitness for breeding.
   (c) Mutate all new solutions.

Bin packing implementation:



Repair bin packing: greedy pack

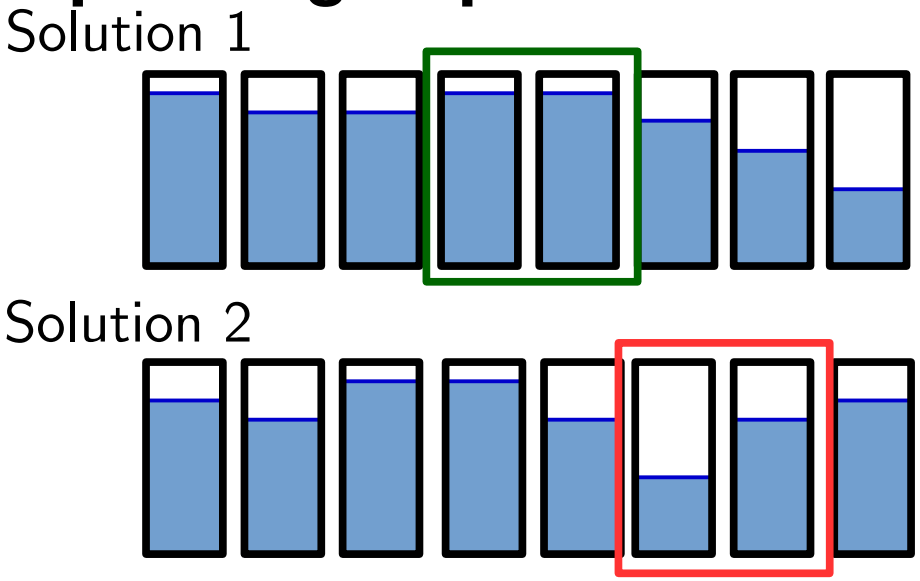
Package 7 not packed!

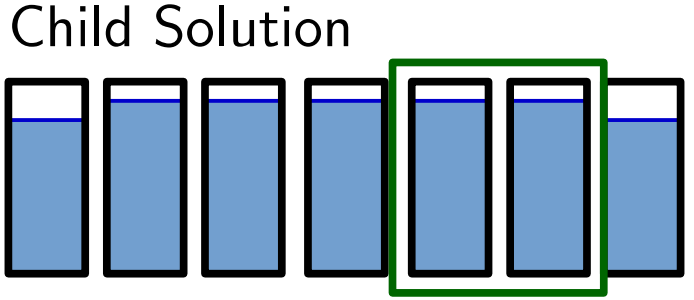


Genetic algorithm:

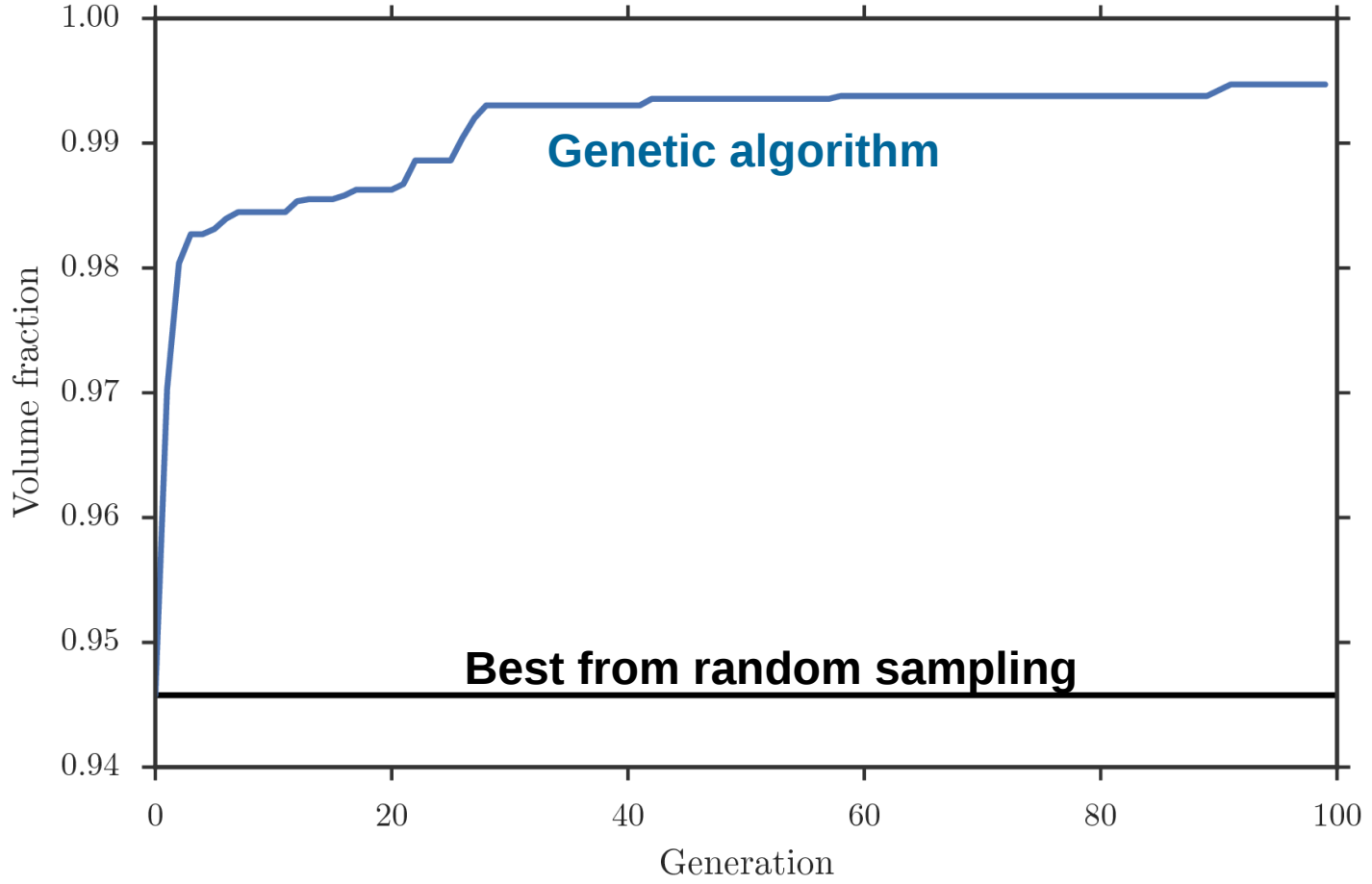
1. Initialize populations.
2. Cycle:
   (a) Measure fitness of all solutions.
   (b) Sample solutions weighted by fitness for breeding.
   (c) Mutate all new solutions.

Bin packing implementation:

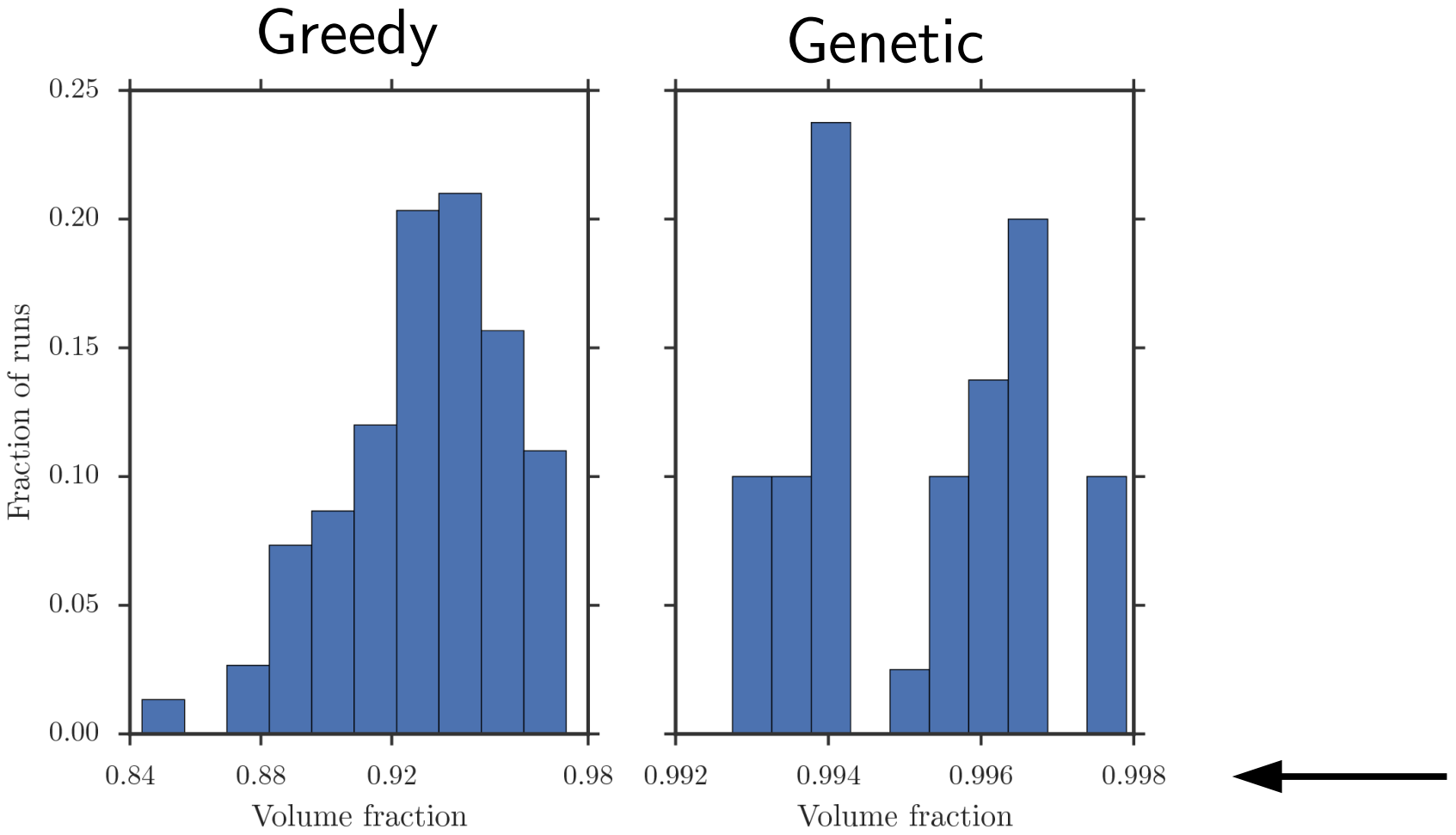




Genetic algorithm demonstration:

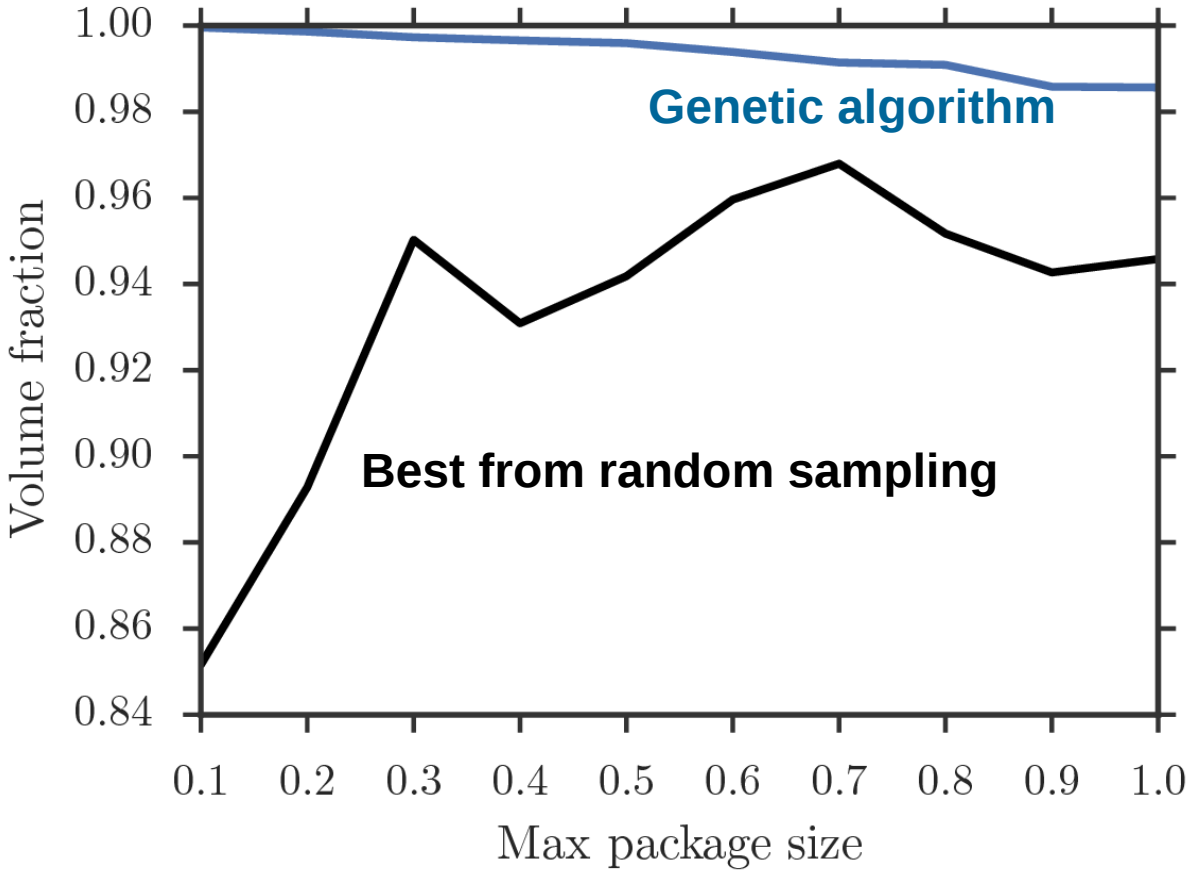


Statistics of genetic algorithm performance.

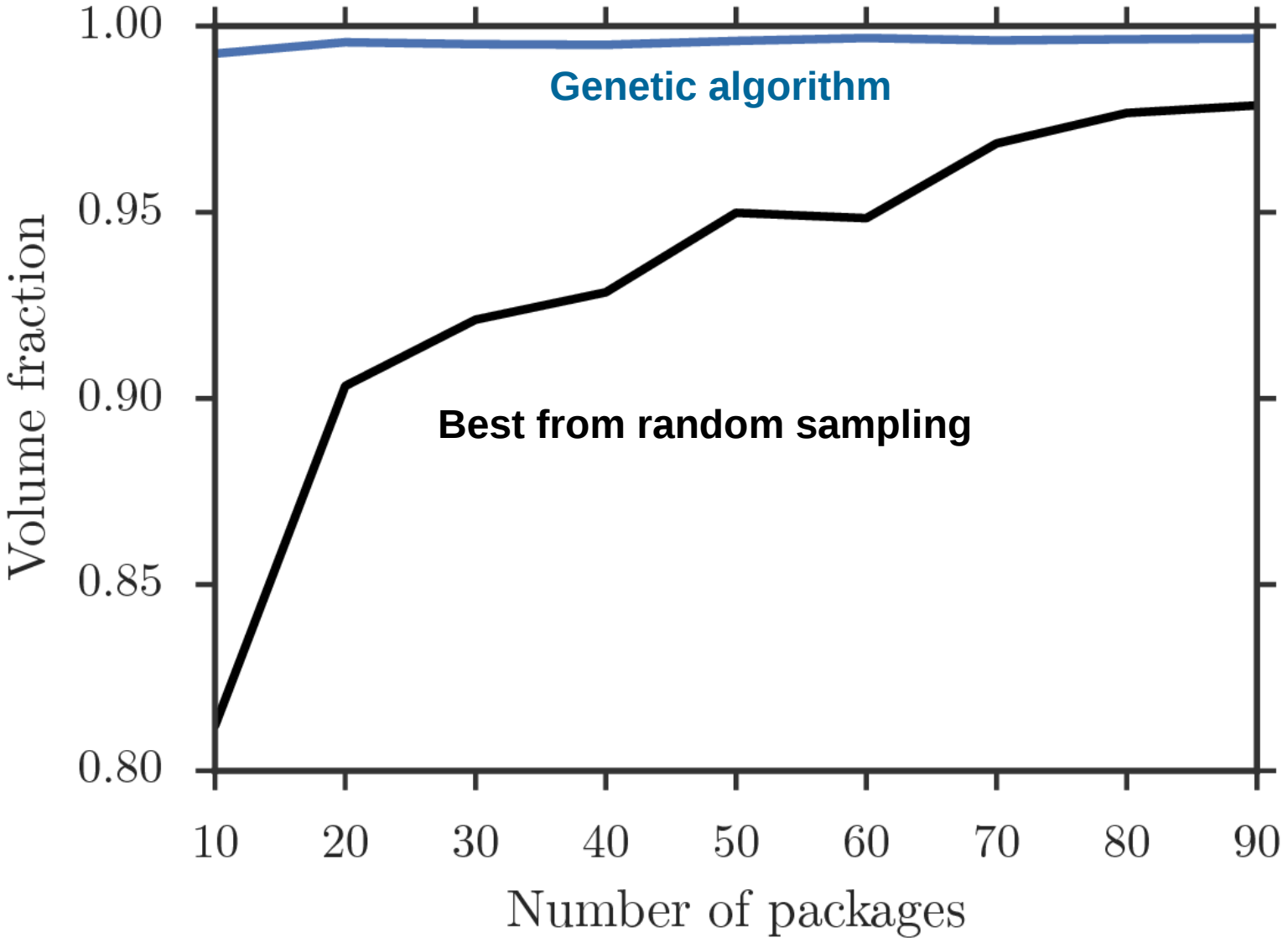


In 80 trials, genetic algorithms fills $> 99\%$ of bins, better than all greedy.

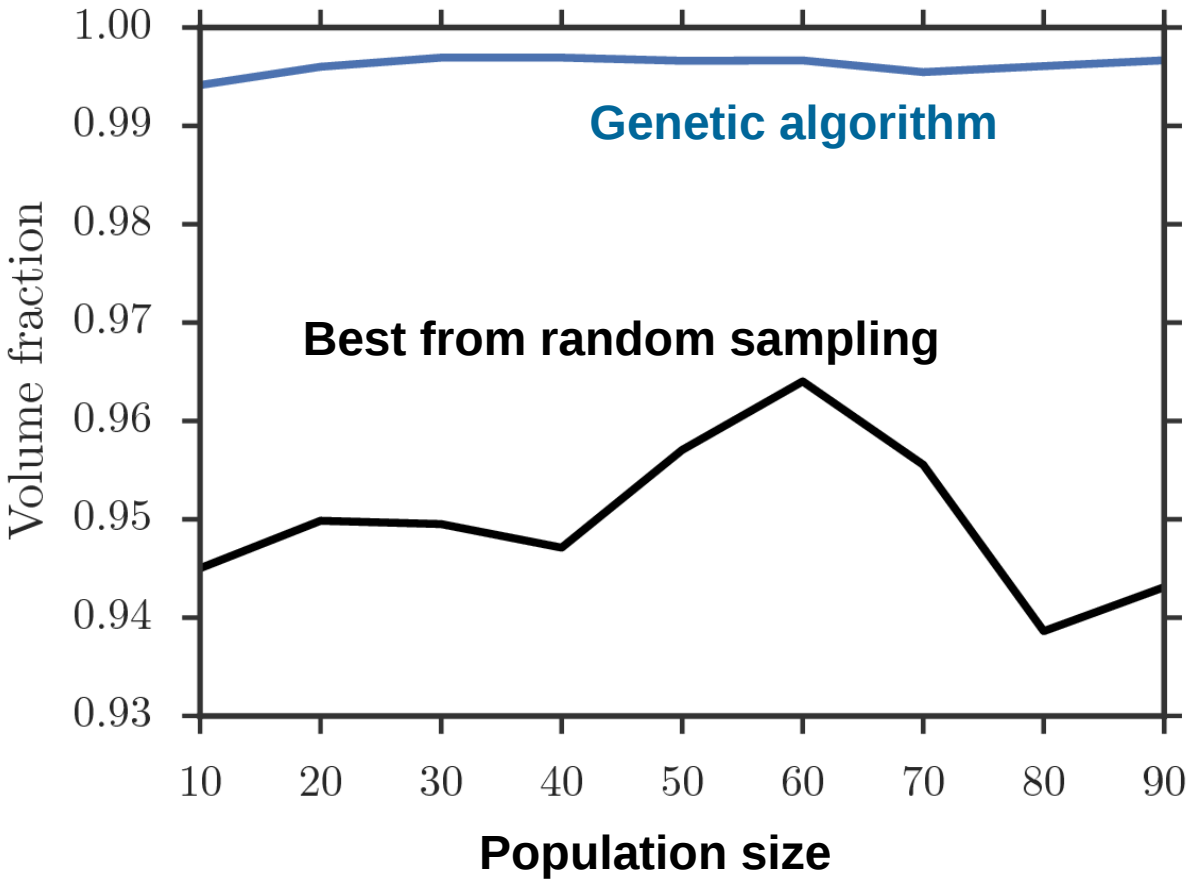
Increasing difficulty: larger packages.



Increasing difficulty: more packages.



Increasing accuracy: larger populations.



Conclusions

Genetic algorithms

- Identify high-fitness partial solutions and attempt to combine.
- Consistent strong improvement over random search.