Computationally Hard Problems

Randomized Search Heuristics – Introduction

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2020

A Typical Evolutionary Algorithm

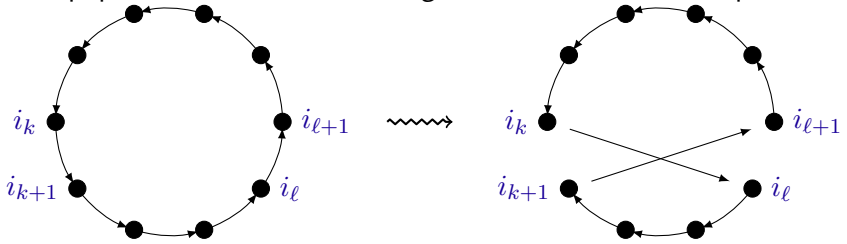
Initialize population P_0 of size μ. Set $t \leftarrow 0$.
while stopping criterion not fulfilled **do**
  for $i \leftarrow 1, \ldots, \mu$ **do**
    Choose two individuals x and y from P_t by applying some selection operator.
    Create z by applying some crossover operator to x and y.
    Create z' by applying some mutation operator to z.
    Add z' to P_{t+1}. (assumption: P_{t+1} initially empty)
  end for
  $t \leftarrow t + 1$.
end while

The scheme is typical but does not cover all variants of EAs. E. g., varying population size is not allowed. Also often mutation and crossover are not necessarily applied in every step but only depending a the so-called mutation and crossover probability.

Operators for the TSP: Mutation

Mutation of a tour $(i_1, \ldots, i_n) \in \Pi_n$.

- Swap mutation usually does not give good results on practical TSP instances. For example, it does not exploit symmetry in the distances ($d(i,j) \approx d(j,i)$).
- Better: Jump: Pick two indices $k < \ell$ and let element at pos. k jump to pos. ℓ, shift intermediate values to the left: $(1,3,4,6,2,5) \mapsto (1,4,6,2,3,5)$ for $k = 2$, $\ell = 5$
- More popular: 2-OPT: Pick two *edges* and connect their endpoints "crossing over"



Operators for the TSP: Crossover

Order Crossover (OX): Choose $k, \ell \in \{1, \ldots, n\}$, where $k < \ell$, u. a. r. Take $(x_k, \ldots, x_\ell)$ from the first parent. Remaining positions, starting from $\ell + 1$, filled consecutively from second parent (also from $\ell + 1$), skipping any occurrences of elements from $x_k, \ldots, x_\ell$.

Example: parents

$$x = (1, 2, 3, 5, 4, 6, 7, 8, 9)$$

and

$$y = (4, 5, 2, 1, 8, 7, 6, 9, 3)$$

and $k = 4$, $\ell = 7$. Then the child z is

$$(2, 1, 8, \mathbf{5}, \mathbf{4}, \mathbf{6}, \mathbf{7}, 9, 3).$$

Second child can be created by swapping roles of x and y.

Operators for the TSP: Crossover

Partially Matched Crossover (PMX) Starts out as OX (take $(x_k, \ldots, x_\ell)$ from first parent). Remaining elements added from second parent by preserving absolute positions as far as possible. Elements j from the second parent that cannot be put at the same position are filled with the elements that stand at the same position in y as j in x (repeating the procedure until a valid element is found).

Example again: parents

$$x = (1, 2, 3, 5, 4, 6, 7, 8, 9)$$

and

$$y = (4, 5, 2, 1, 8, 7, 6, 9, 3)$$

and $k = 4$, $\ell = 7$.

Child starts out as

$$(*, *, *, \mathbf{5}, \mathbf{4}, \mathbf{6}, \mathbf{7}, *, *).$$

PMX continued: All elements but 4 and 5 can be taken from y and put at the same position. Intermediate result:

$$(*, *, 2, \mathbf{5}, \mathbf{4}, \mathbf{6}, \mathbf{7}, 9, 3)$$

The element 4 from the first position of y shows up at x_5. Hence, the first element of z becomes $y_5 = 8$. As the element 5 is in x_4, the second position becomes $y_4 = 1$. The final result is

$$(8, 1, 2, \mathbf{5}, \mathbf{4}, \mathbf{6}, \mathbf{7}, 9, 3).$$

Operators for the TSP: Crossover

Cycle Crossover (CX): tries to maintain many absolute positions of the first parent by identifying a "cycle" through the parents. Consider the two parents to define a mapping f on the indices: $f(x_i) := y_i$ for $i \in \{1, \ldots, n\}$.

Start by copying x_1 to z_1. Then the element of x that equals $y_1 = f(x_1)$, say x_j, is copied to z at the same position j. Proceed by finding the element of x equaling $f(x_j)$ etc. until we hit an element that has already been copied to z. The remaining positions are filled with the missing elements according to the order in y.

Operators for the TSP: Crossover

CX example: again parents

$$x = (1, 2, 3, 5, 4, 6, 7, 8, 9)$$

and

$$y = (4, 5, 2, 1, 8, 7, 6, 9, 3),$$

Child starts out as $(1, *, \ldots, *)$.

As $y_1 = 4$, next element added is the 4 at position 5: $(1, *, *, *, 4, *, *, *, *)$.

Now $y_5 = 8$, hence, 8 is added at position 8.

Proceeding in this way, $(1, 2, 3, 5, 4, *, *, 8, 9)$ is obtained.

As 5 maps to 1, which is already contained in y, fill the remaining two positions with entries from y. Finally, $z = (\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{5}, \mathbf{4}, 7, 6, \mathbf{8}, \mathbf{9})$.

Operators for the TSP: Crossover

Partition Crossover (PX) and Generalized Partition Crossover (GPX): the most advanced and often most successful class of operator. New idea: consider the permutations as tours, focus on edges instead of cities.

Structural properties:

- preserves the relative order in which cities are visited from one of the parents
- preserves the absolute positions of the cities with respect to one of the Hamilton cycles, given a fixed starting city.

$\rightarrow$ Combines the aims of OX and PMX.

(Without proof) PX can be implemented to run in time $O(n)$.

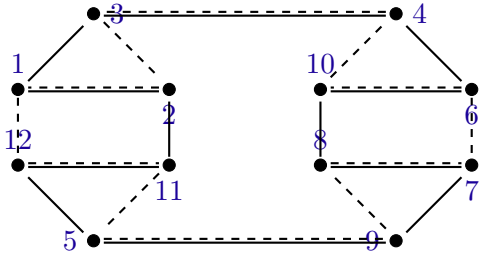
Operators for the TSP: Crossover

Step 1: take the union of the tours given by the parents. Find edges common to both tours.

$$x = (5, 12, 11, 2, 1, 3, 4, 6, 10, 8, 7, 9)$$

and

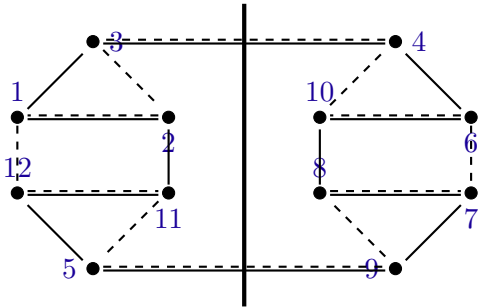
$$y = (4, 10, 6, 7, 8, 9, 5, 11, 12, 1, 2, 3)$$



There are several common edges: $\{3, 4\}$, $\{5, 9\}$, $\{1, 2\}$, $\{11, 12\}$, ..., ...

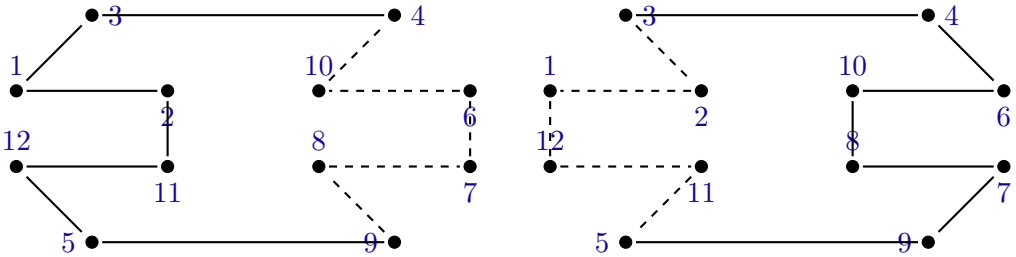
Operators for the TSP: PX continued

Step 2: Find a cut through exactly two common edges. These edges are called cut edges. (If impossible: crossover not feasible, return parents.)



Operators for the TSP: PX continued

Step 3: Given the two edges e and e' from the cut, start out at e, follow the tour from the first parent up to e' and then the tour from the second one. Other child is created in the opposite way.

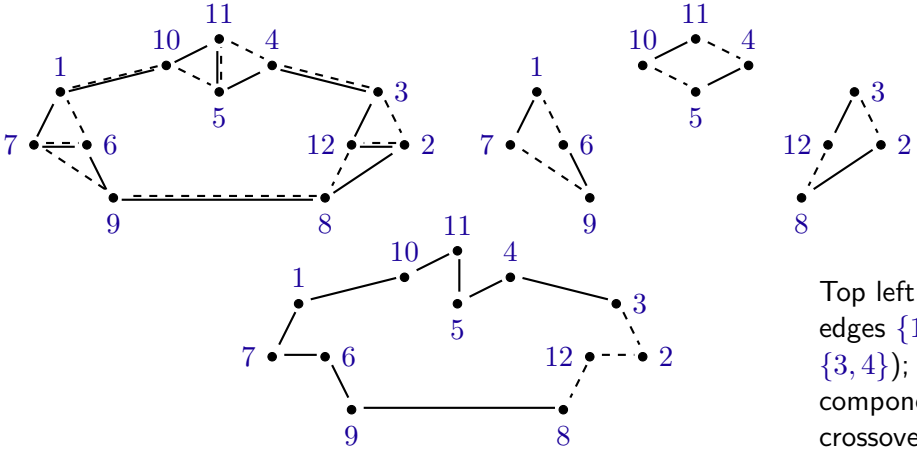


Operators for the TSP: GPX

GPX generalizes PX by considering possibly more than one cut:

- Take out all common edges: the graph falls apart into connected components.
- Determine for each component the shortest path through the component, again based on all edges from the two parents.
- Concatenation of these tours then gives rise to an offspring.

Operators for the TSP: GPX



Top left: union graph (cut edges $\{1, 10\}$, $\{8, 9\}$ and $\{3, 4\}$); top right: components; bottom: crossover result

Demonstrations

- Mutation-only (1+1) EA. 2-OPT as only operator (Bachelor project from 2017)
- GA using populations and GPX as crossover operator (Master project by J. P. Träff, 2013)

Computationally Hard Problems

Analysis of Randomized Search Heuristics – Foundations

Carsten Witt

Institut for Matematik og Computer Science
Danmarks Tekniske Universitet

Fall 2020

Two Very Simple Search Heuristics

Search space $\{0,1\}^n$ 

, "population" size 1, "offspring population" size 1, selection: "take the better"

Aim: maximize $f\colon \{0,1\}^n \to \mathbb{R}$

(1+1) Evolutionary Algorithm ((1+1) EA)

1. $t := 0$. Choose $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ uniformly at random.
2. $y := x$
3. Independently for each bit in y: flip it with probability $\frac{1}{n}$ (mutation).
4. If $f(y) \geq f(x)$ Then $x := y$ (selection).
5. $t := t + 1$. Continue at line 2.

Two Very Simple Search Heuristics

Search space $\{0,1\}^n$ 

, "population" size 1, "offspring population" size 1, selection: "take the better"

Aim: maximize $f\colon \{0,1\}^n \to \mathbb{R}$

Randomized Local Search (RLS)

1. $t := 0$. Choose $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ uniformly at random.
2. $y := x$
3. Choose one bit in y uniformly and flip it. (mutation).
4. If $f(y) \geq f(x)$ Then $x := y$ (selection).
5. $t := t + 1$. Continue at line 2.

Extremely simple (good for analysis) and surprisingly efficient.

Focus: smallest t ("runtime") to reach optimal solution

Framework for Analysis

Given

- randomized search heuristic A
- fitness function f

study no. T of f-evaluations (black-box) until A finds optimum.

T is random variable

- ideally study whole distribution $\Pr(T \leq t)$
- less ambitious: expectation $E(T)$
- even less ambitious (but feasible): bounds on $E(T)$

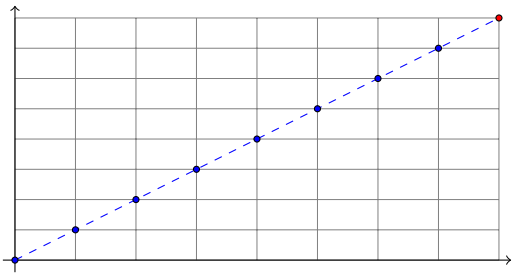
For (1+1) EA: T equals no. of iterations until optimum found.
Call this runtime/optimization time of (1+1) EA on f.

What problems/fitness functions to study?

Example Problems/Toy Problems

Most famous example problem: $\textsc{OneMax}(x_1, \ldots, x_n) = x_1 + \cdots + x_n$
(the heuristic does not know it is working on it)

$\textsc{OneMax}(x)$



Example Problems/Toy Problems

Most famous example problem: $\textsc{OneMax}(x_1, \ldots, x_n) = x_1 + \cdots + x_n$
(the heuristic does not know it is working on it)

Why should we care about such example problems?

- support analysis, help to develop analytical tools
- are easy to understand, are clearly structured
- make important aspects visible
- act as counterexamples
- help to discover general properties
- are important tools for further analysis $\rightarrow \mathcal{NP}$-complete problems
- positive results on easy examples make us trust the algorithm

A First Attempt

Theorem (General Upper Bound)

The expected optimization time of the (1+1) EA on an arbitrary function is $O(n^n)$.

Proof:

- Wait for current bitstring to mutate to optimum.
- At most n bits need to flip: probability at least $1/n^n$.
- Waiting time argument (Lemma A2 in lecture notes). □

General upper bound for RLS does not exist (∞).

Upper Bound on OneMax

Theorem

The expected optimization time of RLS and the (1+1) EA on OneMax *is* $O(n \log n)$.

Proof for (1+1) EA:

- Consider $\phi \in \{0, \ldots, n\}$: current no. one-bits
- Divide run: phase i starts when $\phi = i$ and ends when ϕ increases
- Sufficient for increase: flip a zero-bit, do not flip rest
- $\Pr(\text{increase } \phi \mid \phi = i) \geq \binom{n-i}{1} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{n-i}{en}$
- $E(\text{length of phase } i) \leq \frac{en}{n-i}$
- Expected duration of all phases $\leq \sum_{i=0}^{n-1} \frac{en}{n-i} = en \sum_{i=1}^{n} \frac{1}{i} = O(n \log n)$ since $\sum_{i=1}^{n} \frac{1}{i} \leq \ln n + 1$.

Upper Bound on OneMax

Theorem

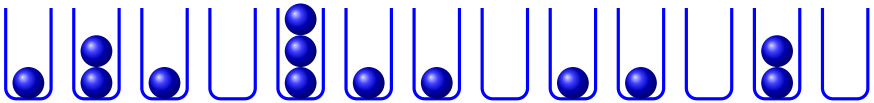
The expected optimization time of RLS and the (1+1) EA on OneMax *is* $O(n \log n)$.

Proof for RLS

- Consider $\phi \in \{0, \ldots, n\}$: current no. one-bits
- Divide run: phase i starts when $\phi = i$ and ends when ϕ increases
- Sufficient for increase: flip a zero-bit
- $\Pr(\text{increase } \phi \mid \phi = i) \geq \binom{n-i}{1} \cdot \frac{1}{n} \geq \frac{n-i}{n}$
- $E(\text{length of phase } i) \leq \frac{n}{n-i}$
- Expected duration of all phases $\leq \sum_{i=0}^{n-1} \frac{n}{n-i} = n \sum_{i=1}^{n} \frac{1}{i} = O(n \log n)$ since $\sum_{i=1}^{n} \frac{1}{i} \leq \ln n + 1$.

Coupon Collector's Problem

Previous reasoning and proof strategy is well known in a combinatorial game.



Scenario: You have n bins. At each time step, you choose a bin uniformly at random and throw a ball in it.

Question: How long does it take until every bin has at least one ball?

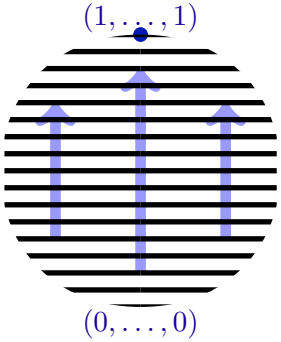
Name: Coupon Collector's Problem (each bin can be considered a "coupon"): there are n types of coupons and in each round you get one coupon uniformly at random with replacement. How long does it take until you have at least coupon of very type?

Known result: Expected time is at least $n \ln n$ and at most $n \ln n + n$.

Note: Arguments for this are in the previous proof (RLS part).

Fitness-Based Partitions

Aim: more general upper-bound method



Definition

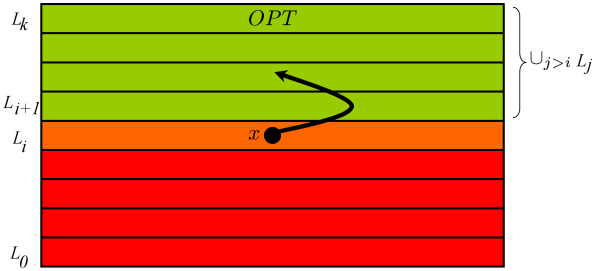
$L_0, L_1, \ldots, L_k \subseteq \{0,1\}^n$ is fitness-based partition iff

1. $\forall i \in \{0,1,\ldots,k\}\colon L_i \neq \emptyset$
2. $\forall i \neq j \in \{0,1,\ldots,k\}\colon L_i \cap L_j = \emptyset$
3. $\bigcup_{i=0}^{k} L_i = \{0,1\}^n$
4. $\forall i < j \in \{0,1,\ldots,k\}\colon \forall x \in L_i\colon\ \forall y \in L_j\colon f(x) < f(y)$
5. $L_k = \{x \in \{0,1\}^n \mid f(x) = \max\{f(x') \mid x' \in \{0,1\}^n\}\}$

Example:

$L_i = \{x \mid \textsc{OneMax}(x) = i\}$

Bounds with Fitness-Based Partitions



Note: for OneMax we had

- $L_i := \{(x_1, \ldots, x_n) \mid x_1 + \cdots + x_n = i\}$
- $s_i \geq \frac{n-i}{en}$.

Bounds with Fitness-Based Partitions

Theorem

Consider (1+1) EA/RLS on $f\colon \{0,1\}^n \to \mathbb{R}$.
Consider f*-based partition* $L_0, L_1, \ldots, L_k$.
Let for all $i \in \{0,1,\ldots,k-1\}$ $s_i := \min_{x \in L_i} \Pr(\textit{mutate } x \textit{ into some } y \in L_{i+1} \cup \cdots \cup L_k)$

Then expected optimization time at most $\leq \sum_{i=0}^{k-1} \frac{1}{s_i}$.

Note: for OneMax we had

- $L_i := \{(x_1, \ldots, x_n) \mid x_1 + \cdots + x_n = i\}$
- $s_i \geq \frac{n-i}{en}$.

Fitness-Based Partitions: Example

Consider $\textsc{BinVal}(x_1, \ldots, x_n) := \sum_{i=1}^{n} 2^{n-i} \cdot x_i$,
another simple function (a so-called separable/linear function).

Theorem

The expected optimization time of the (1+1) EA and RLS on $\textsc{BinVal}$ *is* $O(n^2)$.

Proof idea:

- $L_i := \{x \mid 2^{n-1} + 2^{n-2} + \cdots + 2^{n-i} \leq \textsc{BinVal}(x) < 2^{n-1} + 2^{n-2} + \cdots + 2^{n-i} + 2^{n-i-1}\}$
  (the i most significant bits are 1, bit $i+1$ is 0).
- $s_i \geq \frac{1}{en}$ for all $i < n$
- Optimization time $\leq n \cdot en = O(n^2)$.