Parameterized Complexity Analysis of Evolutionary Algorithms

Frank Neumann[1] **Andrew M. Sutton**[2]

[1]Optimisation and Logistics Group
School of Computer Science
University of Adelaide, Australia

[2]Hasso-Plattner-Institut
Universität Potsdam
Potsdam, Germany

GECCO'15 Companion, July 11-15, 2015, Madrid, Spain
ACM 978-1-4503-3488-4/15/07.
http://dx.doi.org/10.1145/2739482.2756562

Introduction

Our task

- Given a **function** $f\colon X \to \mathbb{R}$
- and a set $D \subseteq X$ of **feasible solutions**,
- find $\arg\max_{x \in D} f(x)$.

We are interested in **general purpose algorithms** that can be applied without problem knowledge

Introduction

Why General Purpose Algorithms?

- **Algorithms** are the **heart** of every **nontrivial computer application**.
- For **many problems** we know good or **optimal algorithms**.
  - Sorting
  - Shortest paths
  - Minimum spanning trees
- What about **new or complex problems**?
- Often there are **no good problem specific algorithms**.

Introduction

Points that may rule out problem specific algorithms

- Problems that are **rarely understood**.
- **Quality of solutions** is determined by **simulations**.
- Problems that fall into the **black box scenario**.
- **Not enough resources** such as time, money, knowledge.

General purpose algorithms are often a good choice.

Introduction

General purpose algorithms for optimizing a function $f\colon X \to \mathbb{R}$

1. Choose a **representation** for the elements in X.
2. Fix a **function** to evaluate the quality (might be different from f).
3. Define **operators** that produce new elements.

Evolutionary Algorithms

Evolutionary algorithms are **general purpose algorithms**.

Follow Darwin's principle (**survival of the fittest**).

Work with a set of solutions called **population**.

Parent population produces offspring population by **variation operators** (mutation, crossover).

Select individuals from the parents and children to create a new parent population.

Iterate the process until a **"good solution"** has been found.

Simple Evolutionary Algorithm

(1+1) EA

$x \leftarrow$ an element of $\{0,1\}^n$ uniformly at random.
repeat forever
  Produce y by flipping each bit of x with probability $1/n$.
  if $f(y) \geq f(x)$ **then** $x \leftarrow y$

Theory of Evolutionary Algorithms

Evolutionary algorithms are **successful** for many complex optimization problems.

Rely on **random decisions** $\Rightarrow$ **randomized algorithms**.

Goal: understand **how** and **why** they work.

Study the **computational complexity** of these algorithms on prominent examples.

Runtime analysis

Black box scenario

- Measure the runtime T by the number of fitness evaluations.
- Studies consider time in dependence of the input to reach
  - An optimal solution
  - A good approximation

Rigorous estimates

- Expected number of fitness evaluations $\mathrm{E}(T)$
- Tail bounds, e.g., useful bounds on $\Pr(T \leq g(n))$ where n measures the size of the problem instance

Motivation

We want to tackle the analysis of randomized search heuristics applied to NP-hard problems

- Reducing the base of the exponent
  - Conflict-directed walk (Schöning, 1999) $O(1.334^n)$ for 3-SAT
- Polynomial-time approximation
  - Partition (Witt, 2005)
  - Vertex Cover (Friedrich et al., 2007, Oliveto et al., 2007)
  - Set Cover (Friedrich et al., 2007)
  - Intersection of $p \geq 3$ matroids (Reichel & Skutella, 2010)
- Average-case analysis
  - Partition (Witt, 2005)

Classical view: understand runtime as a function of problem size alone.

Real world problems: inputs are often structured or restricted in some way.

Motivation

Type-checking in ML

- No explicit typing in ML: compiler must infer types/check for consistency: complete for EXPTIME
- Let k be the nesting depth of a type declaration, there is an exact algorithm that solves the problem in $O(2^k n)$
- In most real-world problems: $k \leq 10$

Motivation

Reconfigurable computing

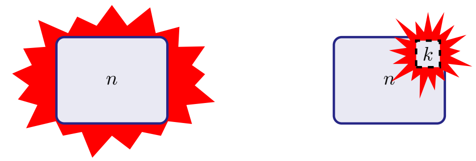
- Given:
  - an $n \times n$ memory array $\mathbf{A}$ with some defective elements
  - k_1 extra rows of spare memory, k_2 extra columns of spare memory
- A defective element can be *repaired* by replacing the row/column that contains it with a spare row/column
- Determine if there is a replacement arrangement that repairs all defective elements in $\mathbf{A}$
- Reduces to: constrained minimum vertex cover in bipartite graphs: NP-complete
- Chen & Kanj (2003): $O(1.26^k n)$ algorithm where $k = k_1 + k_2$
- In the real world, due to hardware constraints, $k \leq 40$

Motivation

Many heuristics are successful in practice because they can take advantage of problem structure...
...want analyses to capture that

Parameterized complexity

Find a hardness parameter k that isolates the source of exponential complexity.



Let L be a language over a finite alphabet Σ.

A parameterization of L is a mapping $\kappa : \Sigma^* \to \mathbb{N}$

Corresponding parameterized problem is given by (L, κ).

For a string $x \in \Sigma^*$, let $k = \kappa(x)$ and $n = |x|$.

An algorithm deciding $x \in L$ in the time bounded by $f(k) \cdot poly(n)$ is called a fixed-parameter tractable (FPT) algorithm for the parameterization κ.

Parameterized complexity for EAs

Monte-Carlo FPT algorithm: in FPT-time, accept with probability at least $1/2$ if $x \in L$, with probability 0 if $x \notin L$.

Definition

An evolutionary algorithm is called *fixed-parameter* tractable (FPT) if it **finds an optimal solution** in expected time $O(f(k) \cdot poly(n))$.

- vertex cover (Kratsch and Neumann, 2013)
- maximum leaf spanning tree (Kratsch et al., 2010)
- MAX-2-SAT (Sutton, Day, Neumann, GECCO 2012)
- Makespan scheduling (Sutton and Neumann, PPSN 2012)
- Euclidean TSP (Nallaperuma et al., CEC 2013)
- Bilevel optimization (Corus, Lehre, and Neumann, GECCO 2013)
- Hypervolume indicator (Bringmann and Friedrich, GECCO 2013)

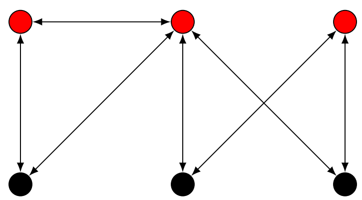
The Minimum Vertex Problem

Friedrich, He, Hebbinghaus, Neumann and Witt (ECJ 2010)
Kratsch and Neumann (Algorithmica 2013)

The Problem

The Vertex Cover Problem:

Given an undirected graph $G = (V, E)$

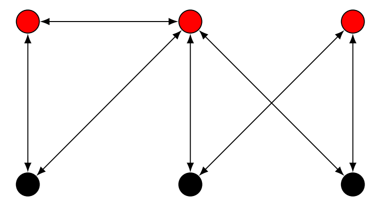


Find a minimum subset of vertices such that each edge is covered at least once.

NP-hard, several 2-approximation algorithms

Simple, single-objective evolutionary algorithms fail!!!

The Problem



Integer Linear Program (ILP)

$$\min \sum_{i=1}^{n} x_i$$
$$\text{s.t. } x_i + x_j \geq 1 \quad \forall \{i,j\} \in E$$
$$x_i \in \{0,1\}$$

Linear Program (LP)

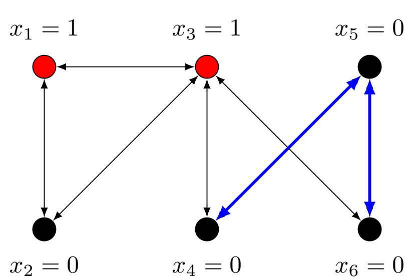
$$\min \sum_{i=1}^{n} x_i$$
$$\text{s.t. } x_i + x_j \geq 1 \quad \forall \{i,j\} \in E$$
$$x_i \in [0,1]$$

Decision problem: is there a set of vertices of size at most k covering all the edges?

Our parameter: value of an optimal solution (OPT)

Evolutionary Algorithm

Representation: bitstrings of length n



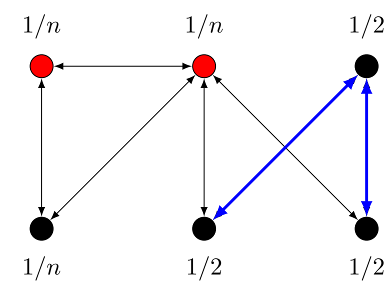
Minimize fitness function:

$f_1(x) = (|x|_1, |U(x)|)$
$f_1(x) = (2, 2)$

$f_2(x) = (|x|_1, LP(x))$
$f_2(x) = (2, 1)$

$U(x) :=$ edges not covered by x
$G(x) := G[U(x)]$
$LP(x) :=$ the value of LP applied to $G[U(x)]$

Evolutionary Algorithm



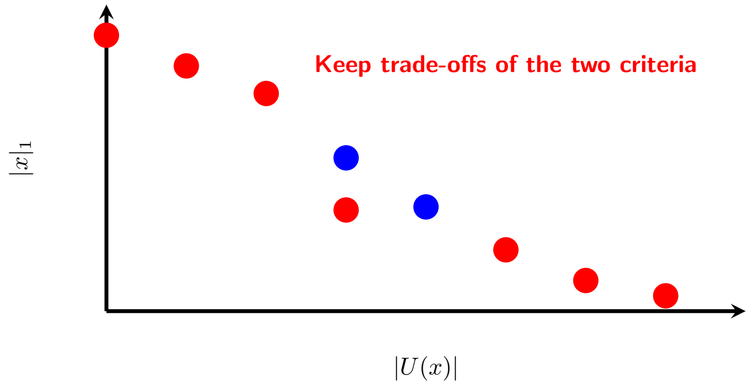
Two mutation operations:

1. Standard bit mutation with probability $1/n$
2. Mutation probability $1/2$ for vertices adjacent to edges of $U(x)$. Otherwise, mutation probability $1/n$.

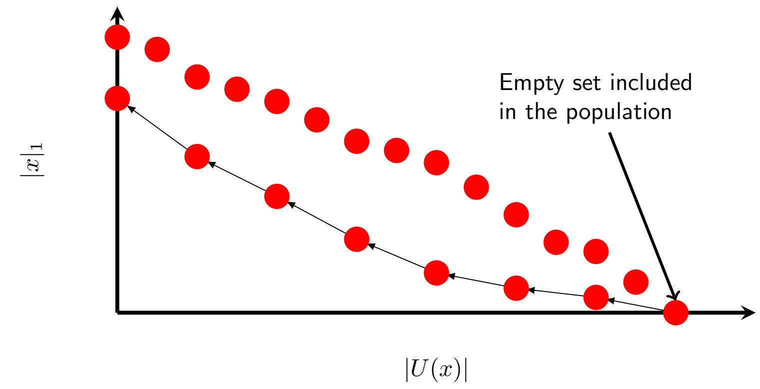
Decide uniformly at random which operator to use in each iteration

Multi-objective approach

Treat the different objectives in the same way

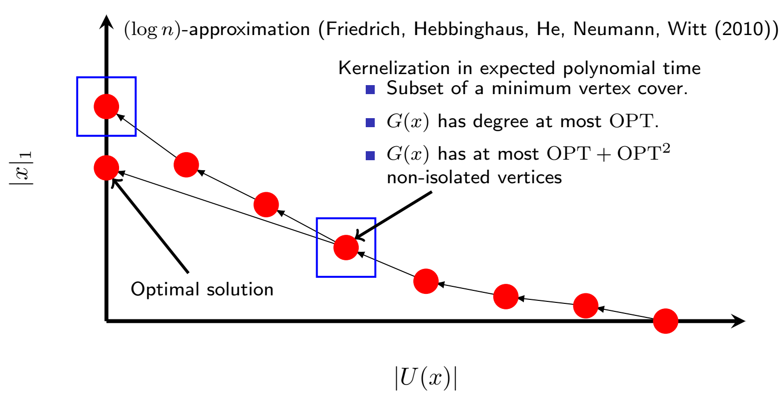


Multi-objective approach



Multi-objective approach

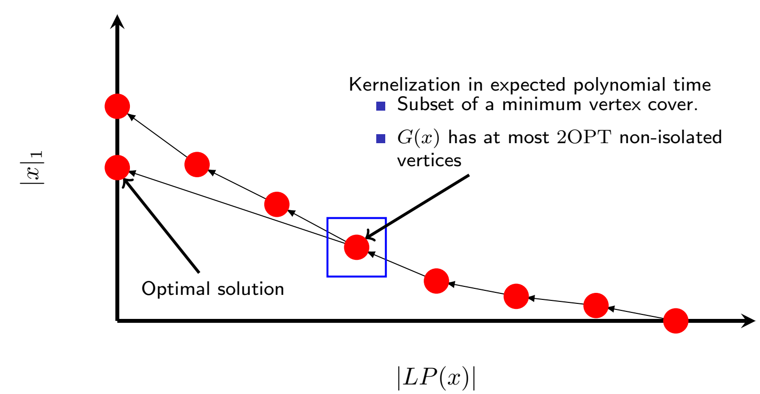
What can we say about these solutions?



Expected time $g(\mathrm{OPT}) \cdot poly(n)$

Fixed-parameter evolutionary algorithm

Multi-objective approach



Expected time $4^{\mathrm{OPT}} \cdot poly(n)$

Fixed-parameter evolutionary algorithm

Linear programming

Combination with linear programming

LP-relaxation is half integral, i.e.,

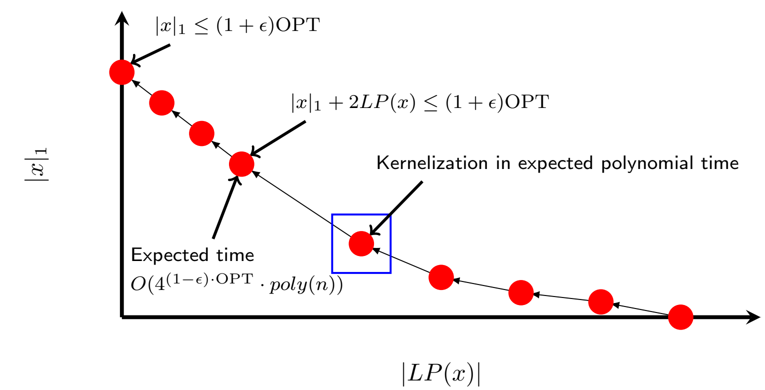
$$x_i \in \{0, 1/2, 1\}, \quad 1 \leq i \leq n.$$

Theorem (Nemhause, Trotter (1975)):
Let x^* be an optimal solution of the LP. Then there is a minimum vertex cover that contains all vertices v_i where $x_i^* = 1$.

Lemma:
All search points x with $LP(x) = LP(0^n) - |x|_1$ are Pareto optimal. They can be extended to minimum vertex cover by selecting additional vertices.

Can we also say something about approximations?

Approximations



Parameterized Complexity Analysis of EAs

Maximum Leaf Spanning Trees

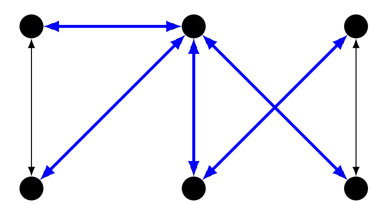
Kratsch, Lehre, Neumann and Oliveto (PPSN 2010)

Parameterized Complexity Analysis of EAs

The Problem

The Maximum Leaf Spanning Tree Problem:

Given an undirected connected graph $G = (V, E)$,



find a spanning tree with a maximum number of leaves.

NP-**hard, different classical FPT-studies.**

Parameterized Complexity Analysis of EAs

Two Evolutionary Algorithms

Algorithm 1: Generic (1+1) EA

Choose a spanning tree T of G uniformly at random
repeat forever
  Produce T' by swapping each edge of T indep. w/ prob. $1/m$
  if T' is a tree and $\ell(T') \geq \ell(T)$ **then** $T \leftarrow T'$

Two Evolutionary Algorithms

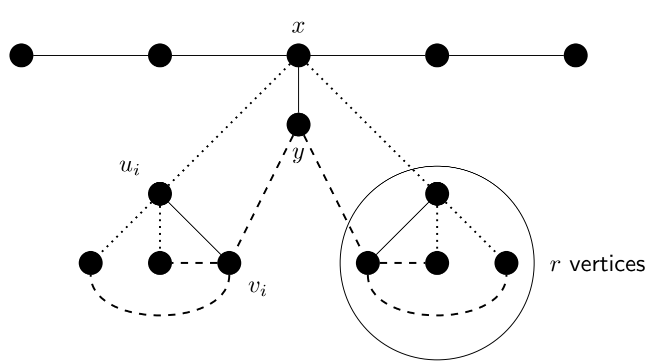
Algorithm 2: Tree-based (1+1) EA

Choose a spanning tree T of G uniformly at random.
repeat forever
  Choose $s \sim \mathrm{Pois}(\lambda = 1)$
  Produce T' by performing sequentially s random edge-exchange operations.
  if $\ell(T') \geq \ell(T)$ **then** $T \leftarrow T'$

A random exchange operation applied to a spanning tree $\widetilde{T}$ chooses an edge $e \in E \setminus \widetilde{T}$ uniformly at random. The edge e is inserted and one randomly chosen edge of the cycle in $\widetilde{T} \cup \{e\}$ is deleted.

Does the mutation operator make the difference between FPT and non-FPT runtime?

Local Optimum



Lower Bounds

Theorem 1.
The expected optimization time of Generic (1+1) EA on G_{loc} is lower bounded by $\left(\frac{m}{c}\right)^{2(r-2)}$ where c is an appropriate constant.

Theorem 2.
The expected optimization time of Tree-Based (1+1) EA on G_{loc} is lower bounded by $\left(\frac{r-2}{c}\right)^{r-2}$ where c is an appropriate constant.

Idea for lower bounds:

- Both algorithms may get stuck in local optimum.
- For the Generic (1+1) EA it is less likely to escape the local optimum as it often flips edges on the path.

Structural insights

Similar to Fellows, Lokshatanov, Misra, Mnich, Rosamond, Saurabh (2009)

Lemma 2.

Any connected graph G on n nodes and with a maximum number of k leaves in any spanning tree has at most $n + 5k^2 - 7k$ edges and at most $10k - 14$ nodes of degree at least three.

Proof idea:

- Let T be a maximum leaf spanning tree with k leaves
- Let P_0 be the set of all leaves and all nodes of degree at least three in T
- Let P be the set of nodes that are of distance at most 2 (w.r.t. T) to any node in P_0 and let Q be the set of remaining nodes.
- **Show:** all nodes of Q have degree 2 in G.
- **Implies:** number of nodes in P is at most $10k - 14$.
- **No node has degree greater than k**, which implies bound on number of edges.

Upper Bound

Theorem 3.

If the maximal number of leaf nodes in any spanning tree of G is k, then Tree-Based (1+1) EA finds an optimal solution in expected time $O(2^{15k^2 \log k})$.

Proof idea:

- We call an edge **distinguished** if it is adjacent to at least one node of degree at least 3 in G.
- **Number of distinguished edges** on any cycle is at most $20k - 28$.
- Total number of edges in G: $m \leq n + 5k^2 - 7k$.
- Probability to introduce a specific non-chosen distinguished edge is at least $1/(m - (n-1)) \geq 1/5k^2$.
- **Show:** length of created cycle is at most $20k$.
- Probability to remove edge of the cycle that does not belong to optimal solution is at least $1/20k$.

Proof of Upper Bound (continued)

- Probability to obtain a specific spanning tree that can be obtained by an edge-swap is at least $1/(20k \cdot 5k^2)$.
- Probability to produce optimal spanning tree, which has distance $r \leq 5k^2$, is at least

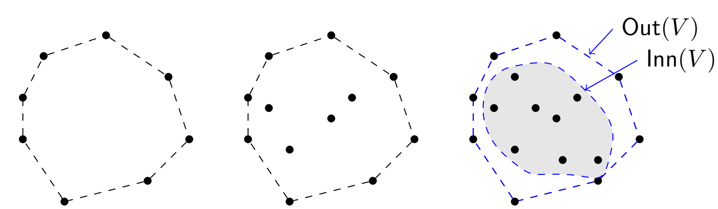
$$r! \cdot \frac{1}{er!}\left(\frac{1}{5k^2} \cdot \frac{1}{20k}\right)^r \geq \frac{1}{e}\left(\frac{1}{100k^3}\right)^{5k^2} \geq \frac{1}{2}\left(\frac{1}{100}\right)^{5k2}\left(\frac{1}{k}\right)^{3 \cdot 5k^2},$$

- Implies that the expected time to get the maximum leaf spanning tree is at most $O(2^{15k^2 \log k})$. □

Euclidean Planar TSP

Given a set V of n points in the plane, find a Hamiltonian cycle of minimum length (NP-hard, Papadimitriou, 1977)

Deĭneko et al. (2006): dynamic programming (simple polygon, k inner points)



TSP parameterization

How does this structure affect evolutionary algorithms?

For n points in the plane with $|\mathrm{Inn}(V)| = k$ interior to the convex hull, what is the runtime of an EA in terms of n and k?

Each tour is represented by a permutation $\pi : V \to V$.

$$v_{\pi(1)} \Rightarrow v_{\pi(2)} \Rightarrow \cdots \Rightarrow v_{\pi(n)} \Rightarrow v_{\pi(1)}$$

Fitness function

$$f(\pi) = \left(\sum_{i=1}^{n-1} d(v_{\pi(i)}, v_{\pi(i+1)}) \right) + d(v_{\pi(n)}, v_{\pi(1)})$$

where $d(u,v)$ is the distance between points u and v.

TSP parameterization

Main structural idea: An optimal tour does not intersect itself.

Lemma

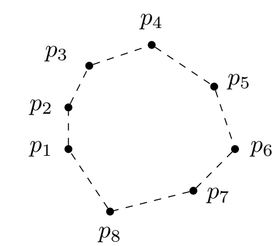
Suppose $\pi^\star$ is a permutation that minimizes f. Then the elements of $\mathrm{Out}(V)$ appear in $\pi^\star$ in the **same** order they appear on the hull.

Definition

We define γ as a linear order on $\mathrm{Out}(V)$

$$\gamma = (p_1, p_2, \ldots, p_{n-k})$$

such that for all $i \in \{1, \ldots, n-k\}$, p_i and p_{i+1} are adjacent on the boundary of the convex hull of V.



For any V, γ can be computed in $O(n \log n)$ time

TSP parameterization

Definition

A permutation π on a subset S of V is γ-**respecting** if and only if, for any $p_i, p_j \in \gamma \cap S$,

$$\pi^{-1}(p_i) < \pi^{-1}(p_j) \implies i < j.$$

where $\gamma \cap S$ means the restriction of γ to S.

Some examples...

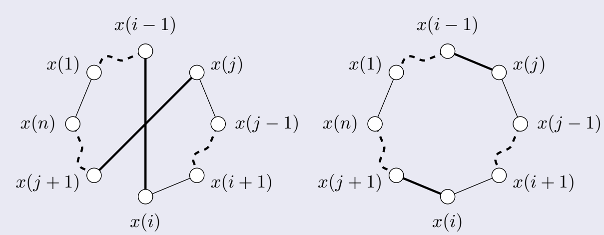
$$(p_1, v_4, v_6, p_2, v_1, v_3, p_3, p_4, v_2, p_5, v_7, p_6, p_7, v_5)$$

$$(v_7, v_5, p_1, v_4, p_2, v_2, v_6, p_3, p_4, p_5, v_1, v_3, p_6, p_7)$$

(1+1) EA in the black-box setting

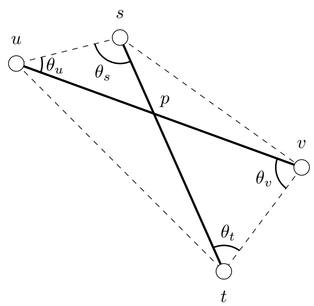
We start in the **black-box** setting (EA has no access to instance structure)

No crossover, mutation is by **edge-exchange** operations, e.g., 2-opt:



(1+1) EA in the black-box setting

Improvement in fitness if sum of edge lengths of new edges is strictly less than sum of edge lengths of old edges.



Main challenge with edge-exchange operations: if angles can be arbitrarily close, fitness improvements can be arbitrarily small.

Idea: assume the angles are **bounded** (or embedded on an $m \times m$ grid).

(1+1) EA in the black-box setting

Theorem

Given a set of (angle bounded) points, a (1+1) EA solves the Euclidean TSP with k inner points in expected time $O(n^{4k}(2k-1)!)$.

Proof idea.
If a tour has edges that cross, an improving move is possible.

With appropriate angle bounds, EA spends $poly(n)$ time on such tours (independent of k).

If the tour has no edges that cross, then it is γ-respecting.

γ-respecting tours are *closer* to optimal tours: the EA only must operate on the inner points to find a solution.

Time to fix inner points: $O(n^{4k}(2k-1)!)$.

(μ+1) EA

FPT evolutionary algorithms (we leave the black-box setting)

FPT (μ+1) EA: based on exact (μ+1) EA for TSP by Theile (2009)

Population of permutations on *subsets* of V with *special structure*

Ground set

- An integer $i \in \{1, \ldots, n-k\}$
- A set $S \subseteq \text{Inn}(V)$
- A vertex $r \in S \cup \{p_i\}$

we identify (i, S, r) with the set $S \cup \{p_1, \ldots, p_i\}$ distinguished by r

An individual $\pi = \pi_{(i,S,r)}$ is a permutation on the ground set $S \cup \{p_1, p_2 \ldots, p_i\}$ and a "tail" vertex r where

- $\pi(1) = p_1$ and $\pi(|S| + i) = r$,
- π is γ-respecting that is, p_1, p_2, …, p_i appear in order.

(μ+1) EA

The subtour defined by a permutation $\pi_{(i,S,r)}$:
$p_1 \Rightarrow v_{\pi(2)} \Rightarrow \cdots \Rightarrow v_{\pi(|S|+i-1)} \Rightarrow r \Rightarrow p_1$

- starts at p_1,
- runs over all nodes in $(S \cup \{p_2, \ldots, p_i\}) \setminus r$ (respecting γ)
- finally visits r before returning to p_1.

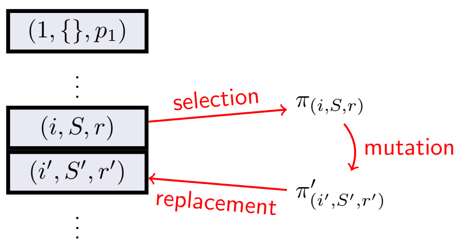
Full population consists of an individual for every $i \in \{1, \ldots, n-k\}$, for every $S \subseteq \text{Inn}(V)$, and every possible tail vertex r, given S and i.

The fitness of an individual is the cost of the corresponding subtour

(μ+1) EA

(μ+1) EA

- Maintain a population P such that each ground-set/tail vertex combination (i, S, r) is represented exactly once.
- While optimal tour is not in P
  - Select an individual $\pi_{(i,S,r)} \in P$ uniformly at random
  - Mutate $\pi_{(i,S,r)}$ to produce $\pi_{(i',S',r')}$ (mutation extends the ground set, and only creates γ-respecting permutations).
  - If the fitness of the mutant $\pi_{(i',S',r')}$ is better than the current individual representing (i', S', r'), then replace that individual with the mutant.



(μ+1) EA

Mutation

To mutate a single individual $\pi = \pi_{(i,S,r)}$,

- choose v uniformly at random from $(\mathsf{Inn}(V) \setminus S) \cup \{p_{i+1}\}$
- concatenate v to the linear order described by π. For $j \in \{1, \ldots, |S| + i + 1\}$,

$$\pi'(j) = \begin{cases} v & \text{if } j = |S| + i + 1; \\ \pi(j) & \text{otherwise.} \end{cases}$$

Thus π' is defined on a different (slightly larger) ground set than π using v as the new tail vertex.

$$\pi' = \begin{cases} \pi'_{(i,S\cup\{v\},v)} & \text{if } v \in \mathsf{Inn}(V); \\ \pi'_{(i+1,S,v)} & \text{if } v = p_{i+1}. \end{cases}$$

When $i = n - k$ and $S = \mathsf{Inn}(V)$ no effect.

Runtime of the (μ+1) EA

Lemma

The population size μ is bounded by $O(2^k kn)$.

Proof. For every ground set / tail vertex combination, there is exactly one individual (invariant).

- There are $\binom{k}{|S|}$ ways to choose a distinct set $S \subseteq \mathsf{Inn}(V)$
- There are $(n - k)$ ways to choose a distinct set of γ-respecting outer points
- There are $|S| + 1$ ways of choosing the tail vertex $r \in S \cup \{p_i\}$.

$$(n - k) \sum_{s=0}^{k} \binom{k}{s} (s + 1) = O(2^k kn).$$

$\square$

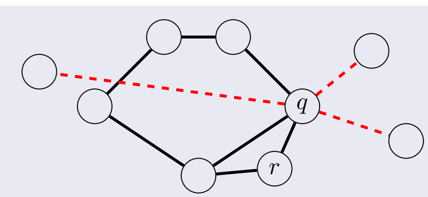
(μ+1) EA

Optimal substructure property

If there is an optimal permutation $\pi_{(i,S,r)}$ in P, there exists some correct mutation that can construct a slightly larger subtour that is also optimal.

$F(i, S, r) :=$ length of optimal tour for (i, S, r) – can be defined recursively using the Bellman Principle

$$F(i, S, r) = \begin{cases} \min\limits_{q \in S \cup \{p_{i-1}\}} F(i - 1, S, q) + d(q, r) & \text{if } r \in \mathsf{Out}(V); \\ \min\limits_{q \in (S \setminus \{r\}) \cup \{p_i\}} F(i, S \setminus \{r\}, q) + d(q, r) & \text{if } r \in \mathsf{Inn}(V). \end{cases}$$



(μ+1) EA

Theorem

Let V be a set of n points in the Euclidean plane with $|\text{Inn}(V)| = k$. After $O(2^k k^2 n^2)$ generations, the (μ+1) EA has solved the TSP on V to optimality in expectation and with probability $1 - e^{-\Omega(n)}$.

Proof. Suppose $\exists\, \pi = \pi_{(i,S,r)} \in P$ with $f(\pi_{(i,S,r)}) = F(i,S,r)$.

- with probability $1/\mu$, π is selected for mutation
- with probability at least $1/(k+1)$, π is extended optimally

Probability of extending an optimal path of length m is at least $\Omega(1/(\mu(k+1))$ (**Bernoulli trial**).

We can use induction on m since the permutation corresponding to $(1, \{\}, p_1)$ is already optimal.

Since optimal paths for a given (i,S,r) are never lost, the expected time until the optimal path of length n exists is $O(n\mu(k+1)) = O(2^k k^2 n^2)$. □

(1+1) EA

We already know that every optimal permutation must be γ-respecting

Suppose we only search the space of orderings on k inner points

Given a permutation π on $\text{Inn}(V)$, need to find where to insert the outer points into π so that the resulting permutation

1. respects γ and π
2. corresponds to the shortest tour of all permutations that respect γ and π

$\gamma = (p_1, p_2, \ldots, p_{n-k})$ $\pi = (q_1, q_2, \ldots, q_k)$

$(p_1, q_1, p_2, p_3, q_2, p_4, q_3, \ldots)$

Exhaustive search $O(n^k)$, but we can be more clever...

(1+1) EA

Direct dynamic programming

Maintain $F[i,j,m]$, a $(n-k) \times (k+1) \times 2$ array where $i \in \{1,\ldots,n-k\}$, $j \in \{0,1,\ldots,k\}$ and $m \in \{Inn, Out\}$.

$F[i,j,m]$ stores fitness of optimal permutation through points $p_1,\ldots,p_i$ and $q_1,\ldots,q_j$ ending on an outer point ($m = Out$) or an inner point ($m = Inn$).

Fitness of π

$Dyn(\pi) = \min\{F[n-k,k,Out]+d(p_{n-k},p_1), F[n-k,k,Inn]+d(q_k,p_1)\}$

Starting with $F[1,0,Out] = 0$, use dynamic programming to fill out F.

$F[i,j,Inn] = \min\{F[i,j-1,Out]+d[p_i,q_j], F[i,j-1,Inn]+d[q_{j-1},q_j]\}$

Cost of computing $Dyn(\pi)$ is $O(kn)$.

(1+1) EA

Search the space of permutations on $\text{Inn}(V)$

(1+1) EA

- Choose uniformly at random a permutation $x = (q_1,\ldots,q_k)$ on the inner points
- While optimum not found
  - Construct x' from x by applying $s+1$ random inversions where s is chosen according to $Pois(1)$
  - If $Dyn(x') \le Dyn(x)$ then $x \leftarrow x'$.

Inversion mutation (pick a subsequence of the permutation and invert it)

$(2, 1, 6, 7, 4, 5, 3) \Rightarrow (2, 4, 7, 6, 1, 5, 3)$

This corresponds to the common 2-opt operation for the TSP.

Runtime analysis of the (1+1) EA

Theorem

The (1+1) EA solves the TSP with k inner points in $O\left((k-1)!k^{2k-2}\right)$ expected calls to the fitness function.

Proof. The probability that a mutation operation for a specific sequence of ℓ basic operations is at least

$$\frac{1}{e(\ell-1)!}\cdot\frac{1}{k^{2\ell}}.$$

Expected waiting time for such a mutation operation is

$$\left(\frac{1}{e(\ell-1)!}\cdot\frac{1}{k^{2\ell}}\right)^{-1} = O(\ell!k^{2\ell}).$$

Need at most $(k-1)$ inversions to transform arbitrary permutation on k points to another. $\square$

Fitness function costs $O(kn)$ $\implies$ the (1+1) EA is FPT.

Makespan Scheduling

- Given a set of n jobs to be scheduled on two machines
- Job j time p_j on either machine.
- A schedule is a decision vector $x \in \{0,1\}^n$
- The *load* of a machine is the sum of processing times assigned to it
- The *makespan* is the maximum load over both machines:

$$f : \{0,1\}^n \to \mathbb{N} := x \mapsto \max\left\{\sum_{j=1}^{n} x_j p_j, \sum_{j=1}^{n}(1-x_j)p_j\right\}.$$

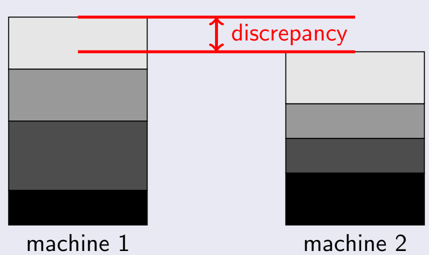
Objective is to find the schedule with the minimum makespan.

- $P = \sum_{j=1}^{n} p_j$.
- $P/2 \le f(x) \le P$.
- WLOG, $p_1 \ge \cdots \ge p_n$.

Makespan Scheduling – parameterization

Definition

The *discrepancy* $\Delta(x) = 2f(x) - P$ is the difference in load across machines.



Given an instance of makespan scheduling and an integer k, is $p_k \ge \Delta^* \ge p_{k+1}$?

- $\Delta^* \ge 0$ discrepancy of optimal schedule
- $p_{n+1} = 0$

Makespan Scheduling

Let $\ell(n)$ denote the run length.

k-biased RLS

$x \leftarrow$ an element of $\{0,1\}^n$ uniformly at random.
for $i \leftarrow 1$ **to** $\ell(n)$
  $y \leftarrow x$
  Choose $0 \le r \le 1$ uniformly at random.
  if $r < 1/n$, **then** Choose $j \in \{1,\ldots,k\}$ u.a.r.
  else Choose $j \in \{k+1,\ldots,n\}$ u.a.r.
  $y_j \leftarrow 1 - y_j$
  if $f(y) \le f(x)$ **then** $x \leftarrow y$

Same as traditional RLS, but prefers not to flip "large" jobs

Makespan Scheduling

Lemma.

Let k be such that $p_{k+1} \leq \Delta^*$. Let x' be a decision vector such that the contribution of jobs $1, \ldots, k$ is minimal. Then starting from x', k-biased RLS with a run length of $\ell(n) = \lceil 2n(\ln n + 1) \rceil$ solves the problem with probability bounded below by $\Omega(n^{-2})$.

Proof sketch.

- Jobs 1 through k are already correct.
- Need to move any small jobs (index $> k$) off of the fuller machine.
- Always possible since $\Delta(x)$ is always larger than a small job, ELSE it is optimal.
- Coupon collector and Markov inequality: $\lceil 2n(\ln n + 1) \rceil$ probability $\Omega(1)$ as long as k large jobs aren't moved.
- Prob. large jobs aren't touched in $\lceil 2n(\ln n + 1) \rceil$ steps: $(1 - 1/n)^{\lceil 2n(\ln n+1) \rceil} = \Omega(n^{-2})$.

Makespan Scheduling

Theorem.

A multi-start k-biased RLS procedure using a run length of $\lceil 2n(\ln n + 1) \rceil$ solves the problem after $O(2^k n^3 \log n)$ steps with probability at least $1 - 1/e$.

Proof sketch.

- Probability that run starts with the first k jobs correctly placed is at least 2^{-k}. Let $q(n)$ be the probability that such a run is successful.
- Failure probability of t consecutive runs is at most

$$\left(1 - \frac{1}{2^k q(n)^{-1}}\right)^t$$

- Setting $t = 2^k q(n)^{-1}$ makes the failure probability at most $1/e$
- By the previous lemma $q(n) = \Omega(n^{-2})$ so $t = O(2^k n^2)$
- Run length is $O(n \log n)$

Thank you

Further reading I

[1] Karl Bringmann and Tobias Friedrich.
Parameterized average-case complexity of the hypervolume indicator.
In *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO)*, pages 575–582. ACM Press, 2013.

[2] Dogan Corus, Per Kristian Lehre, and Frank Neumann.
The generalized minimum spanning tree problem: a parameterized complexity analysis of bi-level optimisation.
In Christian Blum and Enrique Alba, editors, *GECCO*, pages 519–526. ACM, 2013.

[3] Stefan Kratsch, Per Kristian Lehre, Frank Neumann, and Pietro Oliveto.
Fixed parameter evolutionary algorithms and maximum leaf spanning trees: A matter of mutation.
In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature - PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 204–213. Springer Berlin / Heidelberg, 2011.

[4] Stefan Kratsch and Frank Neumann.
Fixed-parameter evolutionary algorithms and the vertex cover problem.
Algorithmica, 65(4):754–771, 2013.

[5] Samadhi Nallaperuma, Andrew M. Sutton, and Frank Neumann.
Fixed-parameter evolutionary algorithms for the Euclidean traveling salesperson problem.
In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 2037–2044, 2013.

Further reading II

[6] Frank Neumann and Carsten Witt.
Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity.
Springer, 2010.

[7] Andrew M. Sutton, Jareth Day, and Frank Neumann.
A parameterized runtime analysis of evolutionary algorithms for MAX-2-SAT.
In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2012)*, 2012.

[8] Andrew M. Sutton and Frank Neumann.
A parameterized runtime analysis of simple evolutionary algorithms for makespan scheduling.
In Carlos A. Coello Coello, Vincenzo Cutello, Kalyanmoy Deb, Stephanie Forrest, Giuseppe Nicosia, and Mario Pavone, editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 52–61. Springer Berlin Heidelberg, 2012.

[9] Andrew M. Sutton, Frank Neumann, and Samadhi Nallaperuma.
Parameterized runtime analyses of evolutionary algorithms for the planar Euclidean traveling salesperson problem.
Evolutionary Computation, (in press), 2014.