Computational Complexity of Evolutionary Computation

Basics

Our task:

Given a function $f : X \rightarrow R$
and $D \subseteq X$ "set of feasible solutions"

Find $\arg\max_{x \in D} f(x)$

General purpose algorithms that can be applied without problem knowledge

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 a new or complex problems?
- Often there are no good problem specific algorithms.

Points that may rule out problem specific algorithms

- Problems that are rarely understood.
- Quality of solutions is determined by simulations.
- Problem falls into the black box scenario.

- Not enough resources such as time, money, knowledge.

General purpose algorithms are often a good choice.

General purpose algorithms for

optimizing a function $f : X \rightarrow R$

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.

Two simple general purpose algorithms

(1+1) EA

1. Choose $s \in \{0,1\}^n$ randomly.
2. Produce s' by flipping each bit of s with probability $1/n$.
3. Replace s by s' if $f(s') \geq f(s)$.
4. Repeat Steps 2 and 3 forever.

RLS

1. Choose $s \in \{0,1\}^n$ randomly.
2. Produce s' from s by flipping **one** randomly chosen bit.
3. Replace s by s' if $f(s') \geq f(s)$.
4. Repeat Steps 2 and 3 forever.

Computational Complexity of Evolutionary Algorithms

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.

Interest:

- Expected number of fitness evaluations E[T].
- Success probability after a fixed number of t steps.

Initialization of (1+1) EA: Choose x $\in \{0,1\}^n$ uniformly at random

$$Prob(x_i = 1) = Prob(x_i = 0) = \frac{1}{2}, 1 \leq i \leq n$$

Sample space

Random variables B_1, B_2, …, B_n

$$B_i : \Omega \rightarrow \{0, 1\}$$

$B_i = 1$ if $x_i = 1$
and
$B_i = 0$ if $x_i = 0$

$$B := \sum_{i=1}^{n} B_i$$

$$E[B] = \sum_{i=1}^{n} (Prob(x_i = 1) \cdot 1 + Prob(x_i = 0) \cdot 0)$$

$$= \sum_{i=1}^{n} (Prob(x_i = 1)) = \frac{n}{2}$$

Expected number of 1-bits in the initial solution is n/2.

Markov's Inequality

Markov's inequality:

$X \geq 0$ random variable, $s > 0$

$Prob(X \geq s \cdot E(X)) \leq \frac{1}{s}$

Probability to have at least $s \cdot \frac{n}{2}$ 1-bits, $s > 1$, is at most $1/s$

Example:

Select $x \in \{0,1\}^{100}$ uniformly at random

$E(B) = 50$

$s = 3/2$: $Prob(B \geq 75) = Prob(B \geq \frac{3}{2} \cdot 50) \leq 2/3.$

Chernoff Bounds

Chernoff bounds:

Let $X_1, X_2, \ldots, X_n : \Omega \to \{0,1\}$ independent random variables with $\forall i \in \{1,2,\ldots,n\} : 0 < Prob(X_i = 1) < 1$.

Let $X := \sum_{i=1}^{n} X_i$.

$$\forall \delta > 0 : Prob(X > (1+\delta) \cdot E(X)) < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^{E(X)}$$

$$\forall 0 < \delta < 1 : Prob(X < (1-\delta) \cdot E(X)) < e^{-E(X)\delta^2/2}$$

$$\delta = \tfrac{1}{2} \qquad Prob(B \geq 75) = Prob(B \geq \left(1 + \tfrac{1}{2}\right) \cdot 50)$$
$$\leq \left(\frac{\sqrt{e}}{(3/2)^{3/2}} \right)^{50} < 0.0045$$

Much better bound than by Markov's inequality (bound was 2/3)

Asymptotic:

$$\delta,\ 0 < \delta \leq \tfrac{1}{2},\ \text{a constant}$$

$$\text{Prob}(\text{B} \geq (\tfrac{1}{2} + \delta)n) \leq (\tfrac{e^{\delta}}{(1+\delta)^{(1+\delta)}})^{n/2} = e^{-\Omega(n)}$$

$$\text{At most } \tfrac{2n}{3} \text{ 1-bit with probability } 1 - e^{-\Omega(n)}$$

Algorithm starts with high probability with a solution that has roughly n/2 1-bits

Assume that the function f to be optimized has a unique optimum

(w.l.o.g. 1^n)

How much time do we need at least to obtain this optimal solution?

Initial solution has at most 2n/3 1-bits with high probability.

Need to collect at least n/3 1-bits.

Simplification: Examine what happens if in each step exactly one randomly chosen bit is flipped

Coupon Collector's Theorem:

Given n different coupons.
Choose at each trial a coupon uniformly at random.
Let X be a random variable describing the number
of trials required to choose each coupon at least once.

Then $E(X) = nH_n$ holds,
where H_n denotes the nth Harmonic number,

$\lim_{n\to\infty} Prob(X > n(\ln n + c)) = 1 - e^{-e^{-c}}$ holds for each constant $c \in R$.

c > 0: c is large, probability close to 0

c < 0: |c| is large, probability close to 1

Collecting n/3 missing 1-bits takes expected time Ω(n log n)
if exactly one randomly bit is flipped.

For general mutation flipping each bit with probability 1/n.

n/3 1-bits are missing after initialization.

Probability that at least one of these bits has not been flipped during t = (n-1) ln n steps is at least

$$1 - (1 - (1 - 1/n)^{(n-1)\ln n})^{n/3} \geq 1 - e^{-1/3} = \Omega(1).$$

Lower bound Ω(n log n) on the expected optimization of the (1+1) EA for each function with a unique optimum

Same lower bound also holds for a polynomial number of optima

Upper Bounds for simple functions:

Assume that the algorithm can always achieve a better function value with good probability.

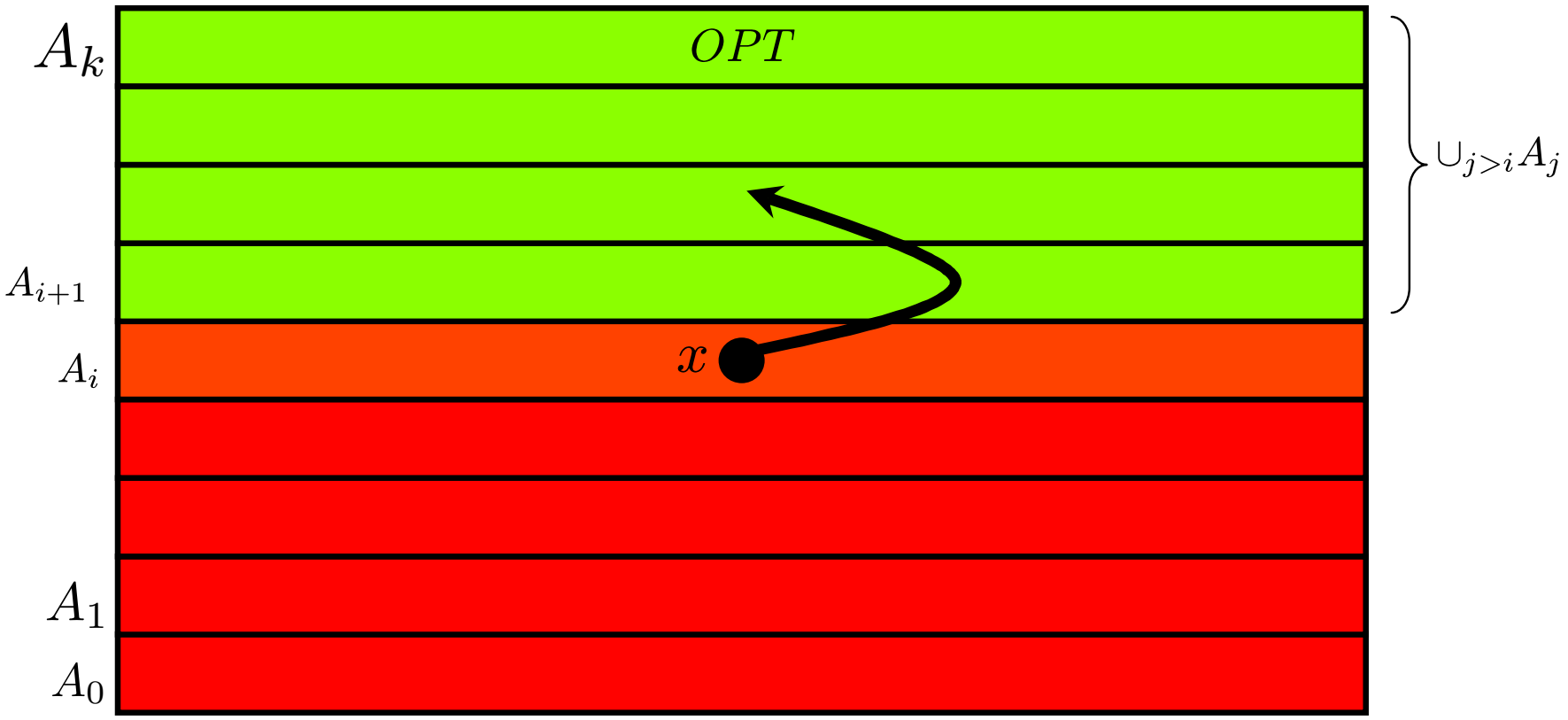
Number of different function values is not too large.

Then: Partition search space according different function values

Consider waiting times for an improvement

Sum up the waiting time for the different improvements

Illustration Fitness-Based Partitions



H(x,y): Hamming distance between bitstrings x and y

Theorem (Upper Bound fitness-based partitions):

Consider (1+1) EA on $f: \{0,1\}^n \to \mathbb{R}$ and an f-based partition $A_0, A_1, \ldots, A_k$.

Let $s_i := \min_{x \in A_i} \sum_{j=i+1}^{k} \sum_{y \in A_j} \left(\frac{1}{n}\right)^{H(x,y)} \left(1 - \frac{1}{n}\right)^{n-H(x,y)}$ for all $i \in \{0, 1, \ldots, k-1\}$.

Then

$$E(T) \leq \sum_{i=0}^{k-1} \frac{1}{s_i}$$

Let $p_i \leq s_i$ be a lower bound on the probability of leaving level A_i

Waiting time to leave level A_i is at most $1/p_i$

Sum up waiting times to get upper bound on the expected optimization time E(T)

Example (1+1) EA and Onemax:

$$\text{OneMax}(x) = \sum_{i=1}^{n} x_i$$

$A_i = \{x \mid \text{OneMax}(x) = i\}$, $0 \leq i \leq n$

Lower bound: $(n-i) \cdot (1/n)^1 (1-1/n)^{n-1} \geq (n-i)/(en) := p_i$

$$\text{E(T)} \leq \sum_{i=0}^{n-1} 1/p_i = \sum_{i=0}^{n-1} en/(n-i) = en \sum_{i=1}^{n-1} \frac{1}{n-i} = O(n \log n)$$

For OneMax the number of different function values is n+1.

Function values give a partitioning into fitness layers.

What about functions that attain many (exponentially) many different values?

Method of fitness-based partitions can also be applied to functions that have many (exponentially) different function values

Idea: Find a small number of partitions such that the probability for an improvement is still good.

Linear Functions:

$f\colon \{0,1\}^n \to \mathbb{R}$ is called linear

$\Leftrightarrow \exists w_0, w_1, \ldots, w_n \in \mathbb{R}\colon \forall x \in \{0,1\}^n\colon f(x) = w_0 + \sum_{i=1}^{n} w_i \cdot x_i$

For (1+1) EA, w.l.o.g. $w_0 = 0,\ w_1 \geq w_2 \geq \cdots \geq w_n \geq 0$

OneMax: $w_0 = 0, w_i = 1, 1 \leq i \leq n$

BinVal(x)= $\sum_{i=1}^{n} 2^{n-i} x_i$ $w_0 = 0, w_i = 2^{n-i}, 1 \leq i \leq n$

Consider (1+1) EA on the class of linear functions.

Fitness-based partitions for linear functions

$$A_i := \left\{ x \in \{0,1\}^n \mid \sum_{j=1}^{i} w_j \leq f(x) < \sum_{j=1}^{i+1} w_j \right\},\ 0 \leq i \leq n$$

Observation:

In partition A_i we can flip one bit and increase the function value by at least w_{i+1}.

Leads to level A_j where j>i

Lower bound for leaving A_i: $\frac{1}{n} \cdot (1-1/n)^{n-1} \geq 1/(en) := p_i$

$$E(T) \leq \sum_{i=0}^{n-1} en = O(n^2)$$

Plateaus

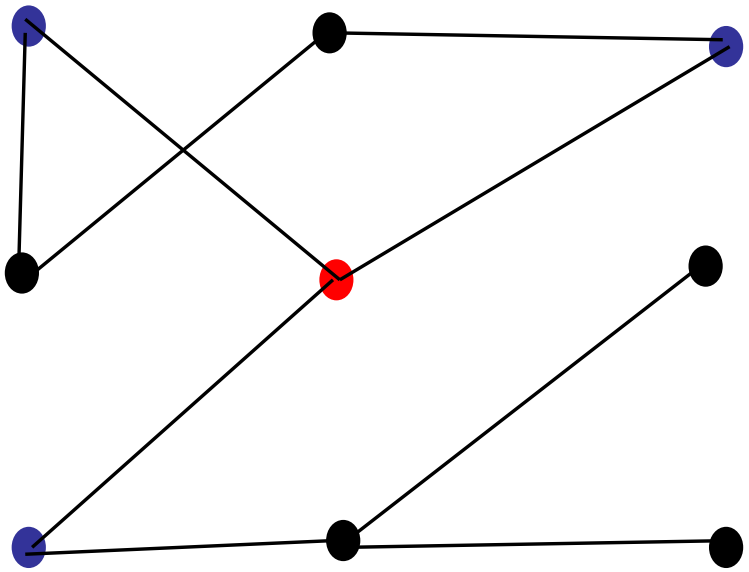
- Plateaus are regions in the search space where all search points have the same fitness
- RLS and (1+1) EA move between solutions that have the same fitness.
- We want to understand how they can escape from plateaus that are non-optimal.

Random Walks

Random Walks on Graphs

Given: An undirected connected graph.

- A random walk starts at a vertex v.
- Whenever it reaches a vertex w, it chooses in the next step a random neighbor of w.



Cover Time

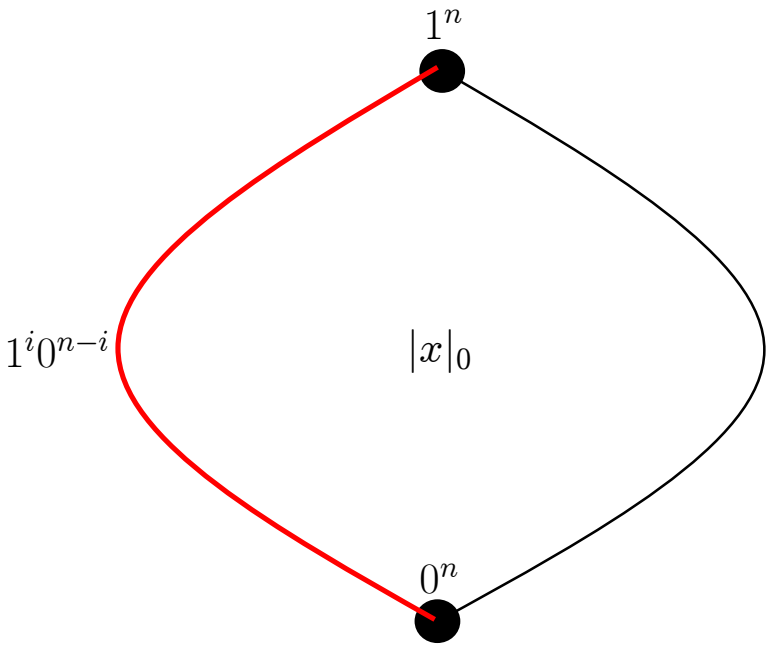
Theorem (Upper bound for Cover Time)

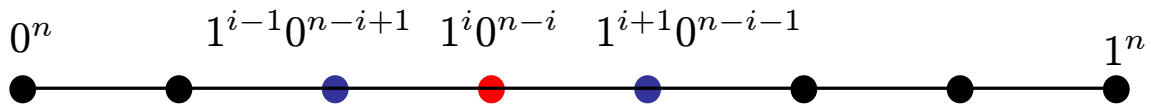
Given an undirected connected graph with n vertices and m edges, the expected number of steps until a random walk has visited all vertices is at most $2m(n-1)$.

Plateaus

Definition

$$\textsc{Plateau}(x) := \begin{cases} |x|_0 & : \quad x \notin \{1^i 0^{n-i}, 0 \leq i \leq n\} \\ n+1 & : \quad x \in \{1^i 0^{n-i}, 0 \leq i < n\} \\ n+2 & : \quad x = 1^n. \end{cases}$$





Upper bound (RLS)

- Solution with fitness $\geq n+1$ in expected time $O(n \log n)$.
- Random walk on the plateau of fitness $n+1$.
- Probability $1/2$ to increase (reduce) the number of ones.
- Expected waiting time for an accepted step $\Theta(n)$.
- Optimum reached within $O(n^2)$ expected accepted steps.
- Upper bound $O(n^3)$ (same holds for (1+1)-EA).

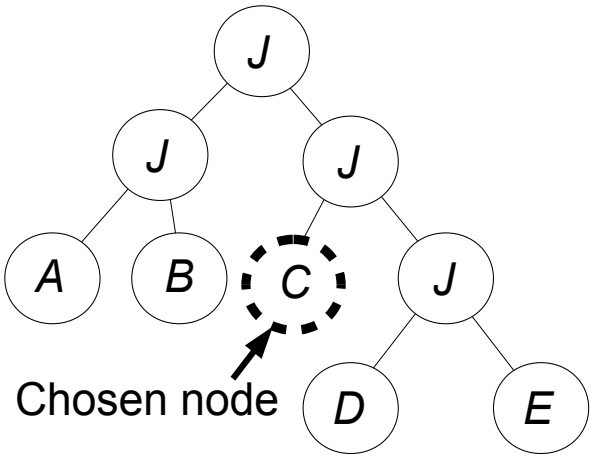
Summary:

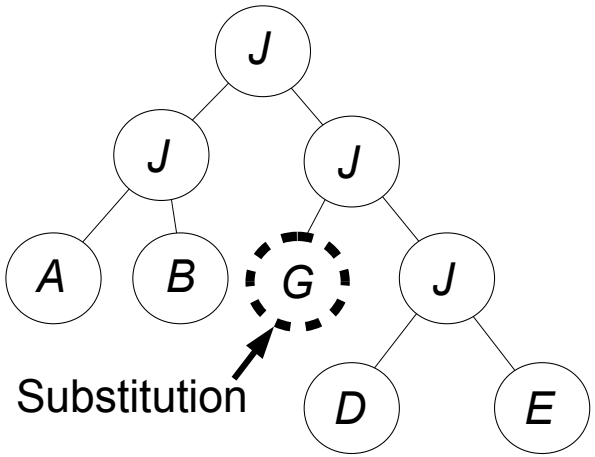
- Runtime Analysis of EAs
- Chernoff bounds, Markov inequality
- Coupon Collectors Theorem
- Fitness-based Partitions
- (1+1) EA on OneMax, Linear Functions
- Random Walks and Plateaus

Genetic Programming

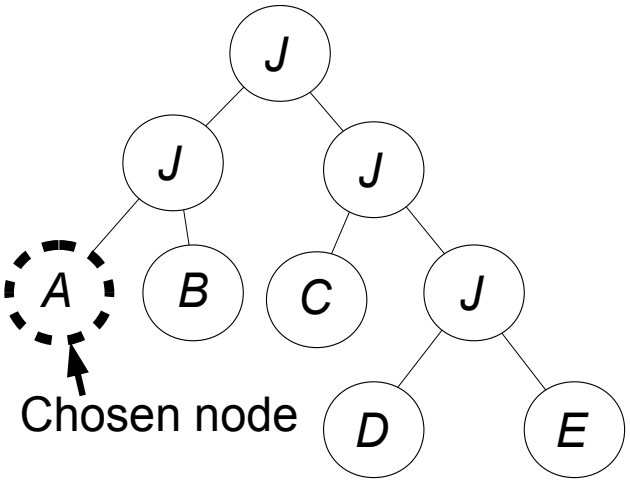
- Type of evolutionary algorithm
- Evolves tree structures for a given problem
- Often used to learn a function
- Consider simple mutation-based genetic programming algorithms

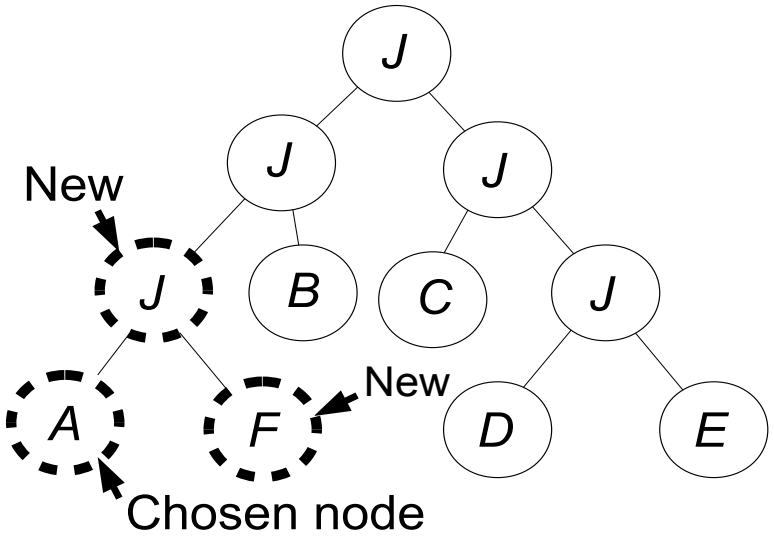
Substitution



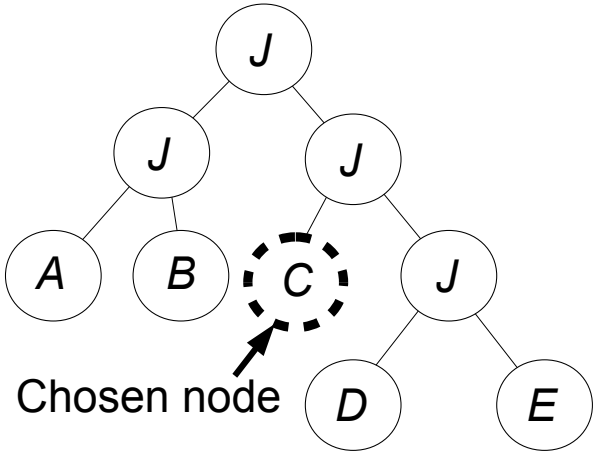


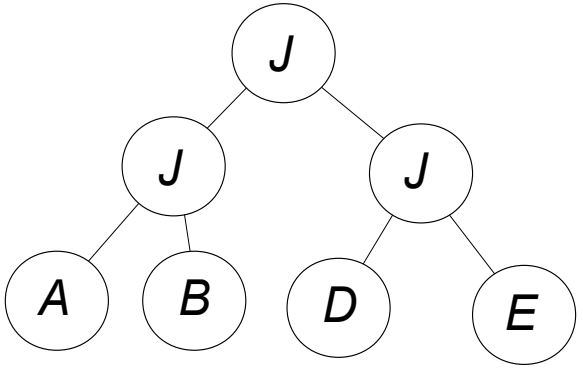
Insertion





Deletion





Simple GP Algorithm

Algorithm 1 ((1+1) GP)**.**

1. *Choose an initial solution* X*.*
2. *Set* $X' := X$*.*
3. *Mutate* X' *by applying HVL-Mutate*$'$ k *times. For each application, randomly choose to either substitute, insert, or delete.*

Substitute
- *If substitute, replace a randomly chosen leaf of* X' *with a new leaf* $u \in L$ *selected uniformly at random.*

Insert
- *If insert, randomly choose a node* v *in* X' *and select* $u \in L$ *uniformly at random. Replace* v *with a join node whose children are* u *and* v*, with the order of the children chosen randomly.*

Delete
- *If delete, randomly choose a leaf node* v *of* X'*, with parent* p *and sibling* u*. Replace* p *with* u *and delete* p *and* v*.*

4. *If* $f(X') \geq f(X)$*, set* $X := X'$*.*
5. *Go to 2.*

Algorithms:

k=1 called (1+1) GP-single

k-1 according to Pois(1) called (1+1) GP-multi

Rejecting Neutral Moves

Algorithm 2 (Acceptance for (1+1) GP*)**.**

4'. If $f(X') > f(X)$, *set* $X := X'$.

ORDER and MAJORITY

Simple Functions

ORDER and MAJORITY (Goldberg/O'Reilly 1998)

- $F := \{J\}$, J has arity 2.
- $L := \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$

Properties of functions

- Separable
- Admit multiple solutions

ORDER

- Imitate semantics of a conditional execution path
- Output depends on the order of leaves in an in-order parse of a program tree

1. *Derive conditional execution path P of X:*

   Init: *l an empty leaf list, P an empty conditional execution path*

   1.1 *Parse X inorder and insert each leaf at the rear of l as it is visited.*

   1.2 *Generate P by parsing l front to rear and adding ("expressing") a leaf to P only if it or its complement are not yet in P (i.e. have not yet been expressed).*

2. $f(X) = |\{x_i \in P\}|$.

Example: X $l = (x_1, \bar{x}_4, x_2, \bar{x}_1, x_3, \bar{x}_6)$, $P = (x_1, \bar{x}_4, x_2, x_3, \bar{x}_6)$ and $f(X) = 3$

Analysis for ORDER:

- Use fitness-based partitions
- Let current fitness be k and current tree size T then probability for an improvement is

$$p_k = \Omega\left(\frac{(n-k)^2}{n \max\{T, n\}}\right)$$

- Summing up waiting times, the expected optimization time is upper bounded by $O(nT_{max})$

MAJORITY

- Imitate semantics of multiple statements
- Output depends on the number of leaves in an in-order parse of a program tree

1. *Derive the combined execution statements S of X:*

   Init: *l an empty leaf list, S is an empty statement list.*

   1.1 *Parse X inorder and insert each leaf at the rear of l as it is visited.*

   1.2 *For $i \leq n$: if $count(x_i \in l) \geq count(\bar{x}_i \in l)$ and $count(x_i \in l) \geq 1$, add x_i to S*

2. $f(X) = |S|$.

Example: X $l = (x_1, \bar{x}_4, x_2, \bar{x}_1, \bar{x}_3, \bar{x}_6, x_1, x_4), S = (x_1, x_2, x_4)$ and $f(X) = 3$.

Analysis for MAJORITY:

Observation:

- Plateaus in the search space.
- Inserts are uniform.
- Probability for deletion of a certain type of variable depends on its fraction in the current tree.
- Enables to use random walk arguments.

Can not move on plateau:

- (1+1) GP*-single expected optimization time is infinite
- (1+1) GP*-multi expected optimization time is exponential.

(1+1) GP-single

- Worst case bound: $O(n^2 T_{\max} \log n)$
- Average case for uniform initialization: $O(nT_{\max} \log n)$

Results for ORDER and MAJORITY

	ORDER	
	(1+1) GP	(1+1) GP*
single	$O(nT_{\max})$ w.c. †	$O(n^2)$ w.c.
multi	$O(nT_{\max})$ w.c. †	$O(nT_{\max})$ w.c. †

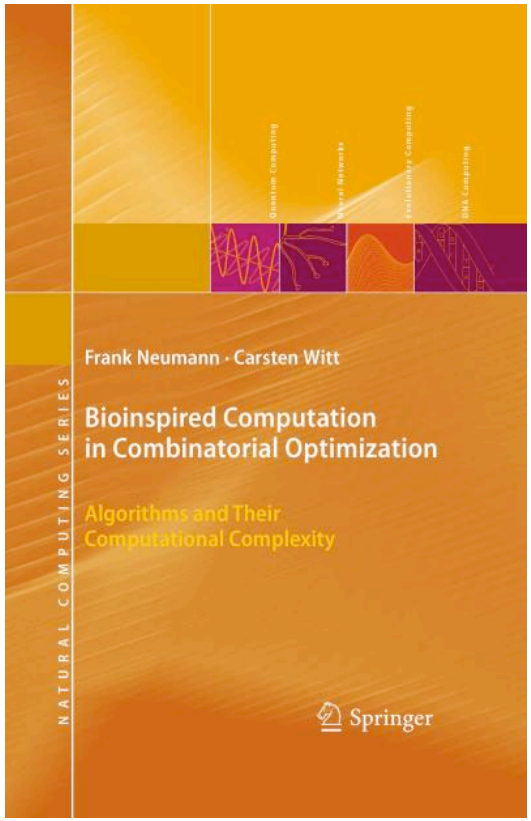
	MAJORITY	
	(1+1) GP	(1+1) GP*
single	$O(n^2 T_{\max} \log n)$ w.c. † $O(nT_{\max} \log n)$ a.c.	$\Omega(\infty)$ a.c.
multi	?	$\Omega\left(\left(\frac{n}{2e}\right)^{\frac{n}{2}}\right)$ w.c.

Techniques: Fitness-based partitions (ORDER),
Random walks and general coupon collector arguments (MAJORITY).

Combinatorial Optimization

Analysis of runtime and approximation quality on combinatorial optimization problems, e. g.,

- sorting problems
- shortest path problems,
- subsequence problems,
- vertex cover,
- Eulerian cycles,
- minimum (multi)-cuts,
- minimum spanning trees,
- maximum matchings,
- partition problem,
- set cover problem,
- . . .



Book available at
www.bioinspiredcomputation.com

Understand the behavior of bio-inspired computation on "natural" examples

Fixed Parameter Evolutionary Algorithms

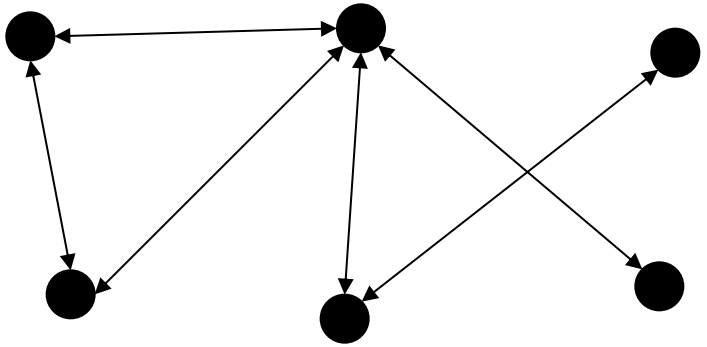
- What makes a problem hard for an EA?
- Consider an additional parameter k to measure "hardness" of an instance
- Fixed parameter algorithm runs in time O(f(k) poly(n))
- Fixed parameter evolutionary algorithm runs in expected time O(f(k) poly(n))
- Consider maximum leaf spanning trees and minimum vertex covers as initial examples

Maximum Leaf Spanning Trees

The Problem

The Maximum Leaf Spanning Tree Problem:

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

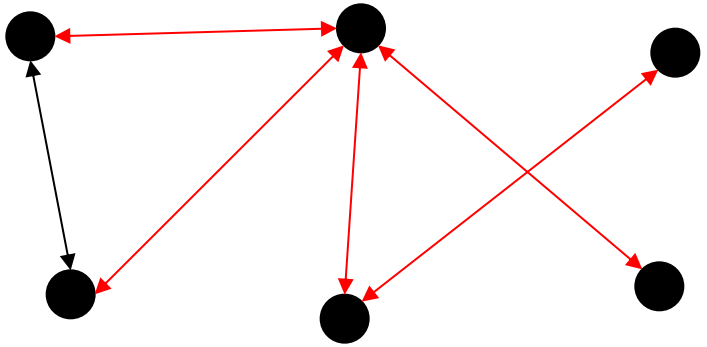


Find a spanning tree with a maximum number of leaves.

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

Two Evolutionary Algorithms

Algorithm 1 (Generic (1+1) EA)

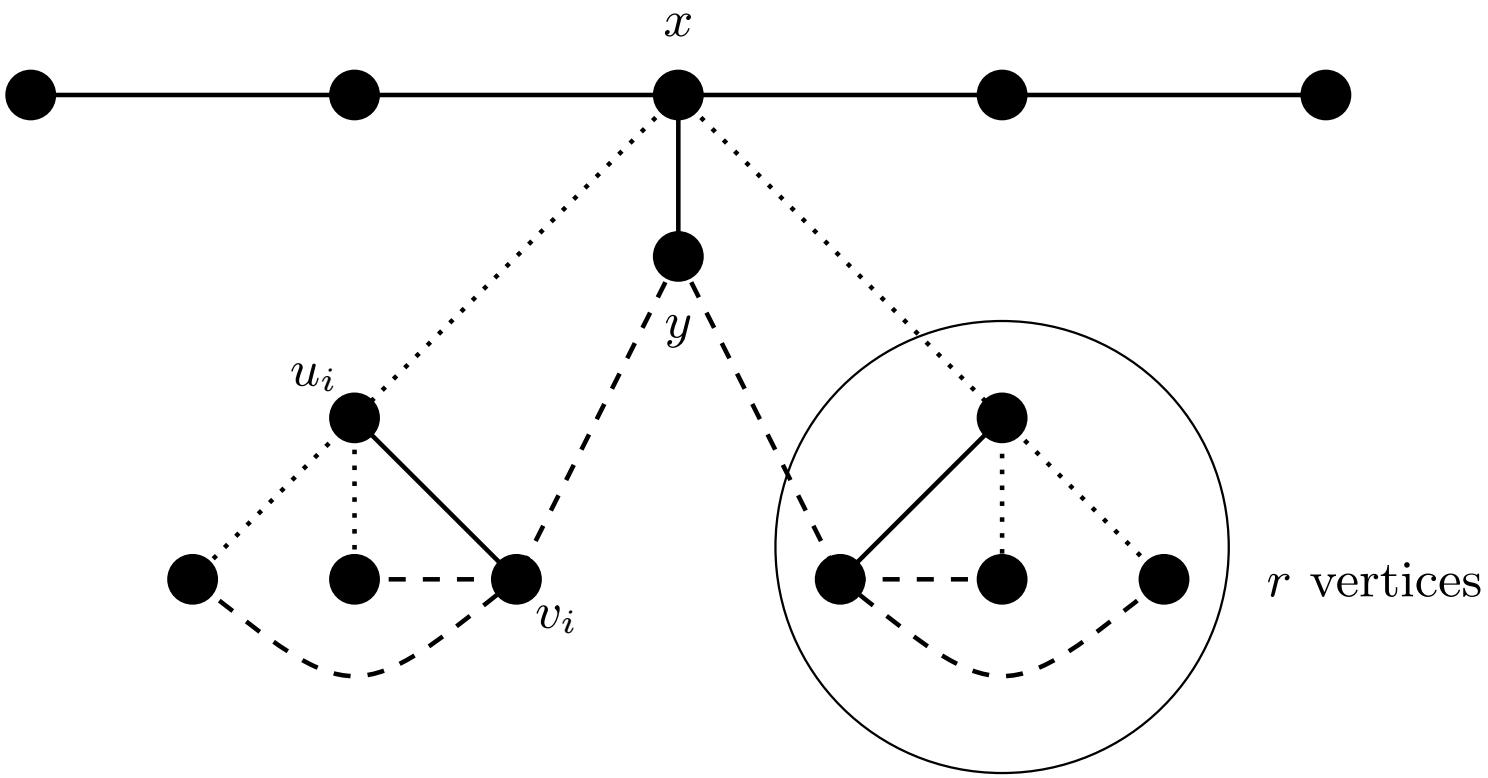
1. *Choose a spanning tree of T uniformly at random.*
2. *Produce T' by swapping each edge of T independently with probability $1/m$.*
3. *If T' is a tree and $\ell(T') \geq \ell(T)$, set $T := T'$.*
4. *Go to 2.*

Algorithm 2 (Tree-Based (1+1) EA)

1. *Choose an arbitrary spanning tree T of G.*
2. *Choose S according to a Poisson distribution with parameter $\lambda = 1$ and perform sequentially S random edge-exchange operations to obtain a spanning tree T'. A random exchange operation applied to a spanning tree $\tilde{T}$ chooses an edge $e \in E \setminus \tilde{T}$ uniformly at random. The edge e is inserted and one randomly chosen edge of the cycle in $\tilde{T} \cup \{e\}$ is deleted.*
3. *If $\ell(T') \geq \ell(T)$, set $T := T'$.*
4. *Go to 2.*

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* $(\frac{r-2}{c})^{r-2}$ *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 local optimum as it often flips edges on the path.

Structural insights

Similar to Fellows, Lokshtanov, 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. to 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 the number of edges.

Upper Bound

Theorem 3. *If the maximal number of leaf nodes in any spanning tree of G is k, then Algorithm 2 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 <= $n+5k^2-7k$
- 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 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

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

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

□

General assumption:

- Multi-objective optimization is more (as least as) difficult as single-objective optimization.
- True, if criteria to be optimized are independent.

Examples:

- Minimum Spanning Tree Problem (MST) (in P).
- MST with at least 2 weight functions (NP-hard).
- Shortest paths (SP) (in P).
- SP with at least 2 weight functions (NP-hard).

- Assume that the criteria to be optimized are not independent.
- Question: Can a multi-objective model give better hints for the optimization of single-objective problems by evolutionary algorithms ?
- Yes!!!

Examples:

- Minimum Spanning Trees (N., Wegener (2006)).
- (Multi)-Cut Problems (N., Reichel (2008)).
- Helper Objectives (Brockhoff, Friedrich, Hebbinghaus, Klein, N., Zitzler (2007)).

Interest here:

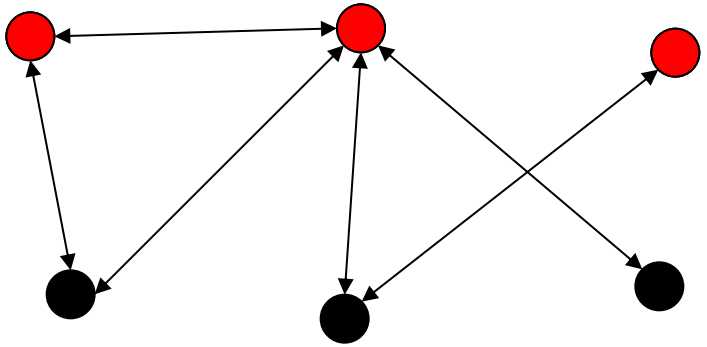
- Theoretical investigations for the Vertex Cover Problem.

The Minimum Vertex Cover Problem

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!!!

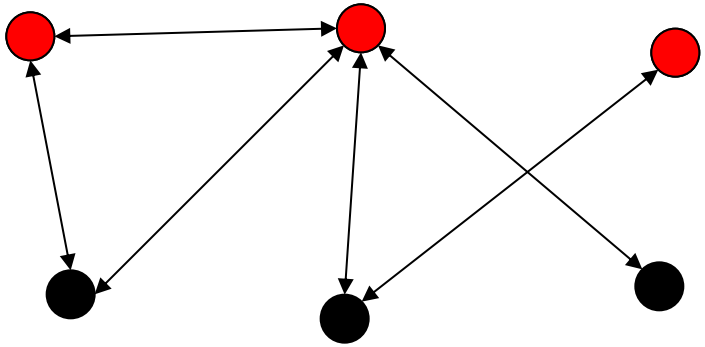
The Problem

The Vertex Cover Problem:

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

Integer Linear Program (ILP)

$$\begin{aligned} &\min \textstyle\sum_{i=1}^{n} x_i \\ &\text{s.t. } x_i + x_j \geq 1 \qquad \forall \{i,j\} \in E \\ &x_i \in \{0,1\} \end{aligned}$$



Linear Program (LP)

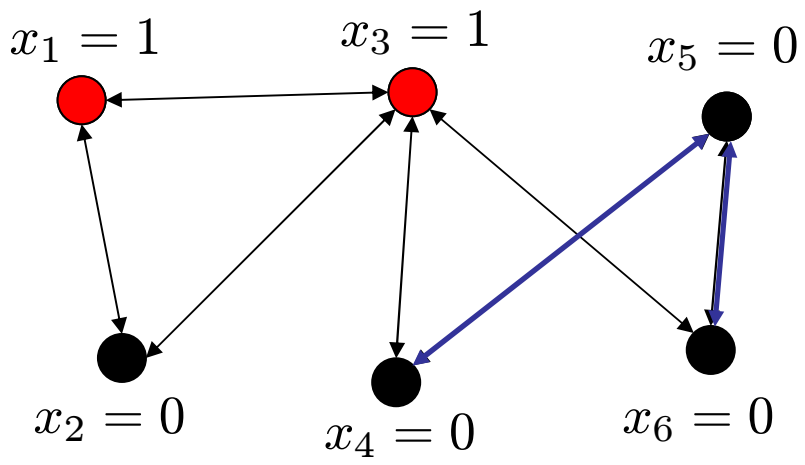
$$\begin{aligned} &\min \textstyle\sum_{i=1}^{n} x_i \\ &\text{s.t. } x_i + x_j \geq 1 \qquad \forall \{i,j\} \in E \\ &x_i \in [0,1] \end{aligned}$$

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

Our parameter: Value of an optimal solution (OPT)

Evolutionary Algorithm

Representation: Bitstrings of length n



Minimize fitness function:

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

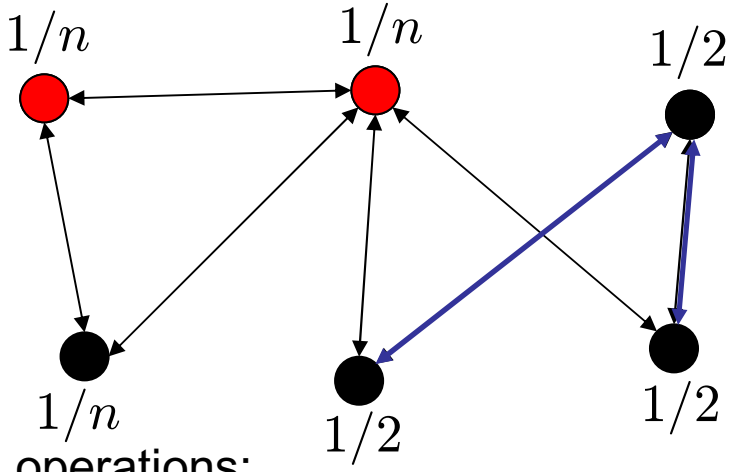
$$f_2(x) = (|x|_1, LP(x))$$
$$f_2(x) = (2, 1)$$

$U(x)$: Edges not covered by x

$G(x) = G(V, U(x))$

$LP(x)$: value of LP applied to $G(x)$

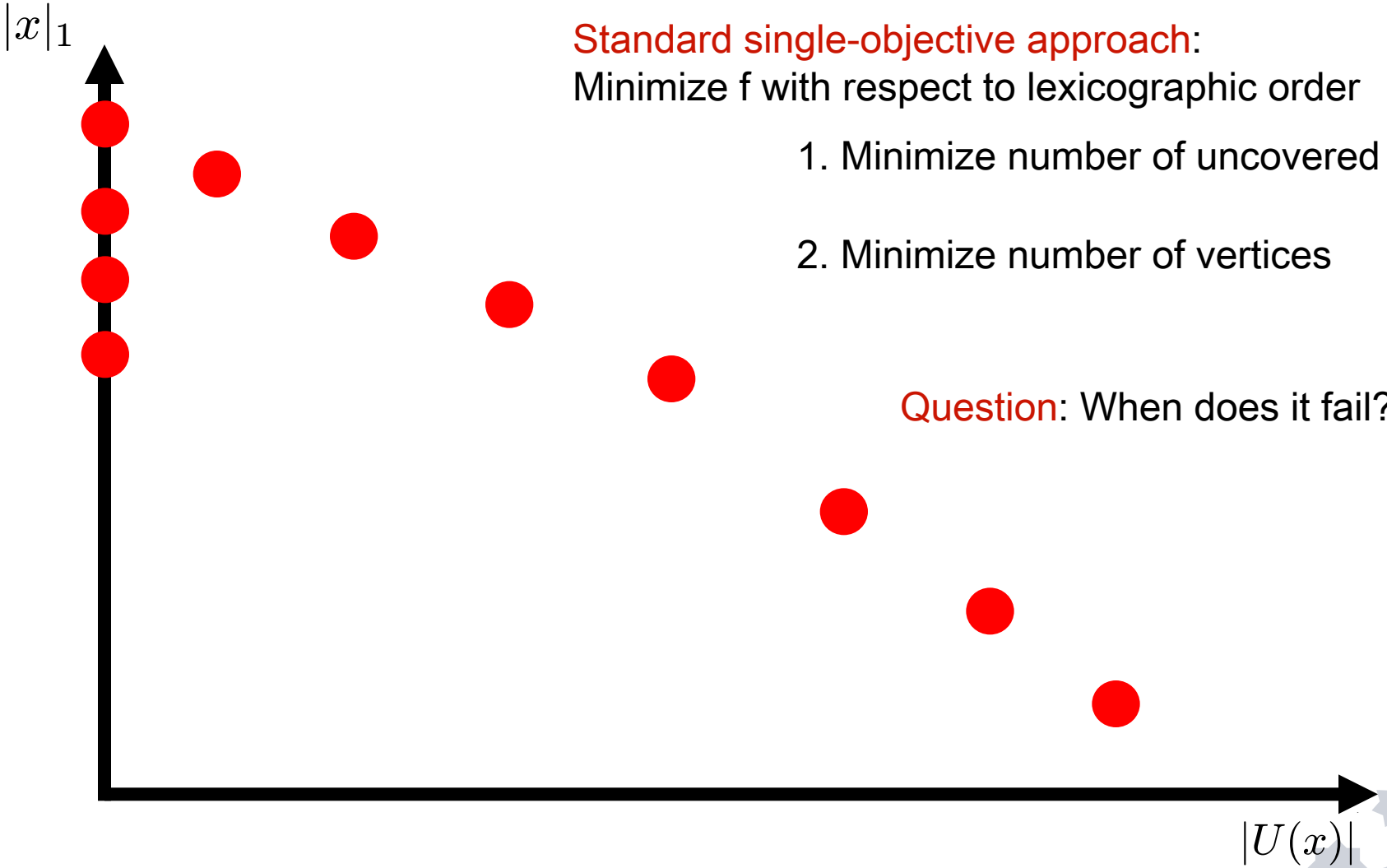
Evolutionary Algorithm



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 next iteration



Standard single-objective approach:
Minimize f with respect to lexicographic order

1. Minimize number of uncovered edges

2. Minimize number of vertices

Question: When does it fail?

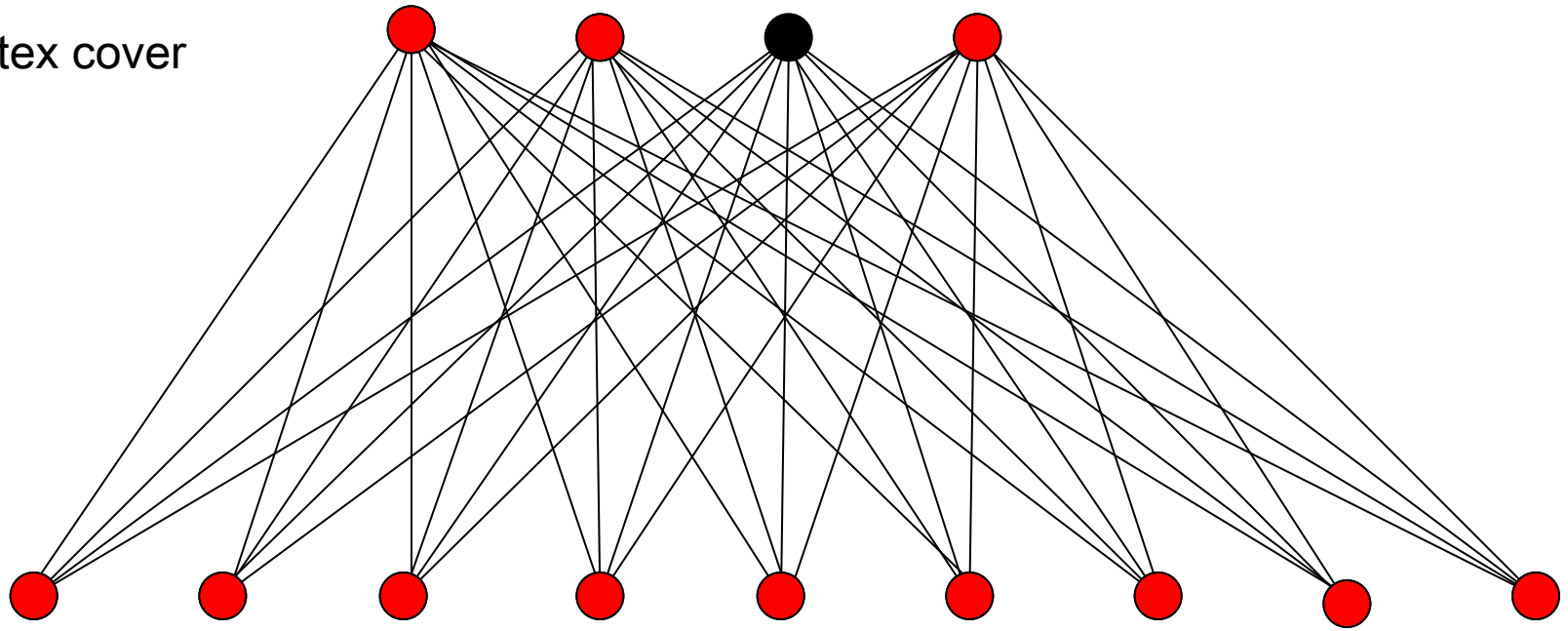
(1+1) EA and Vertex Cover Problem

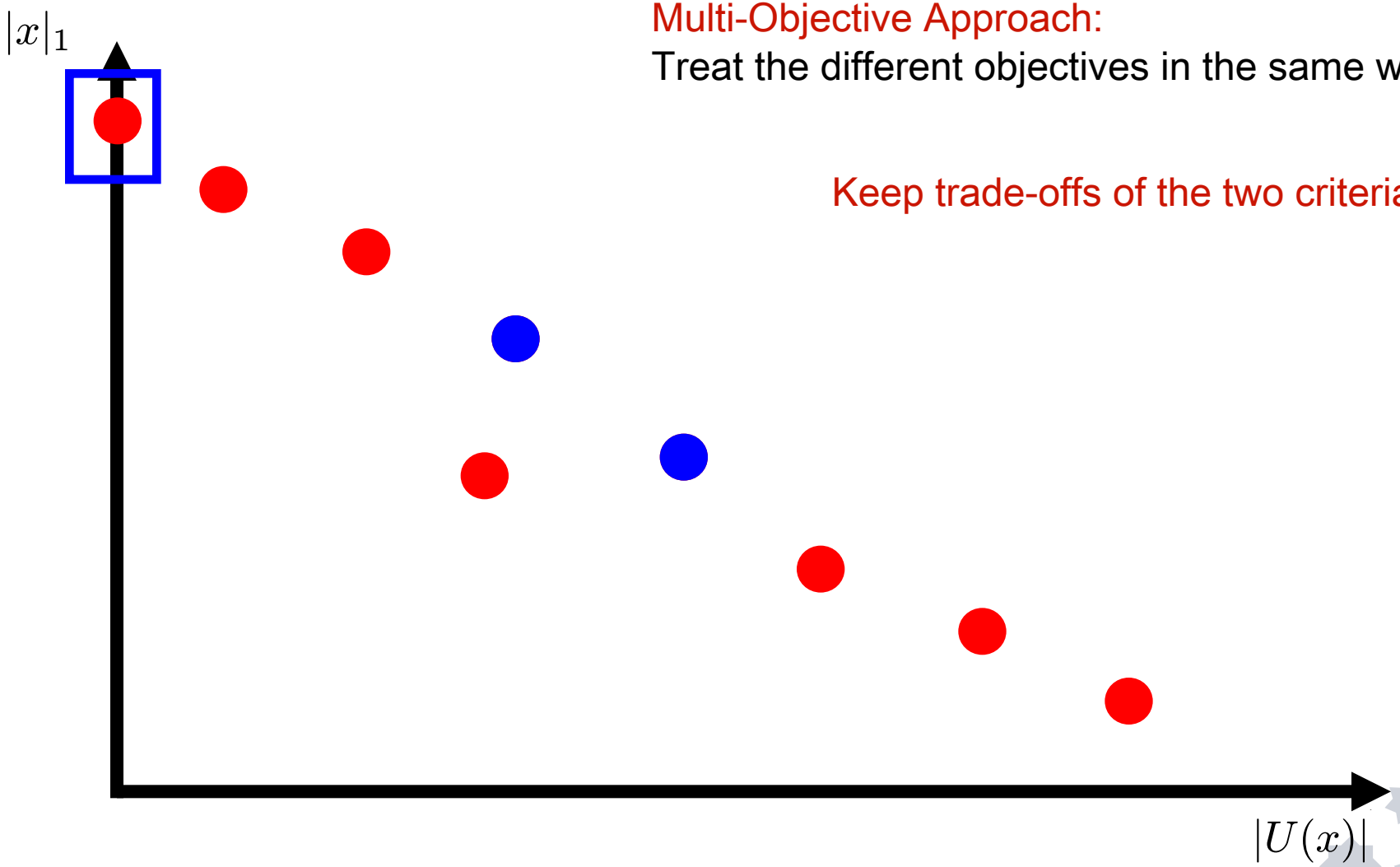
Friedrich, He, Hebbinghaus, N., Witt (2007)

Exponential expected optimization time

Approximation may be arbitrary bad

vertex cover

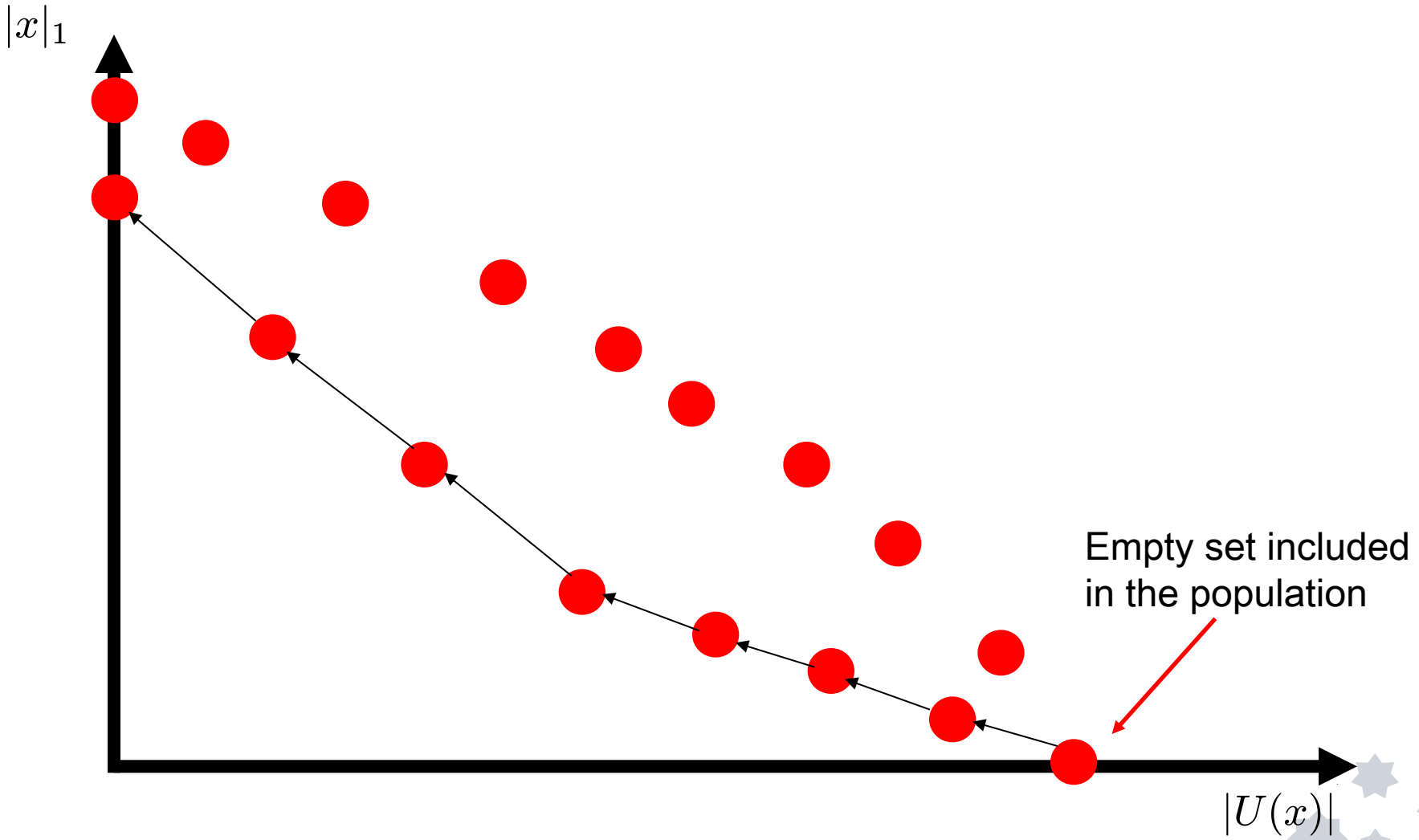


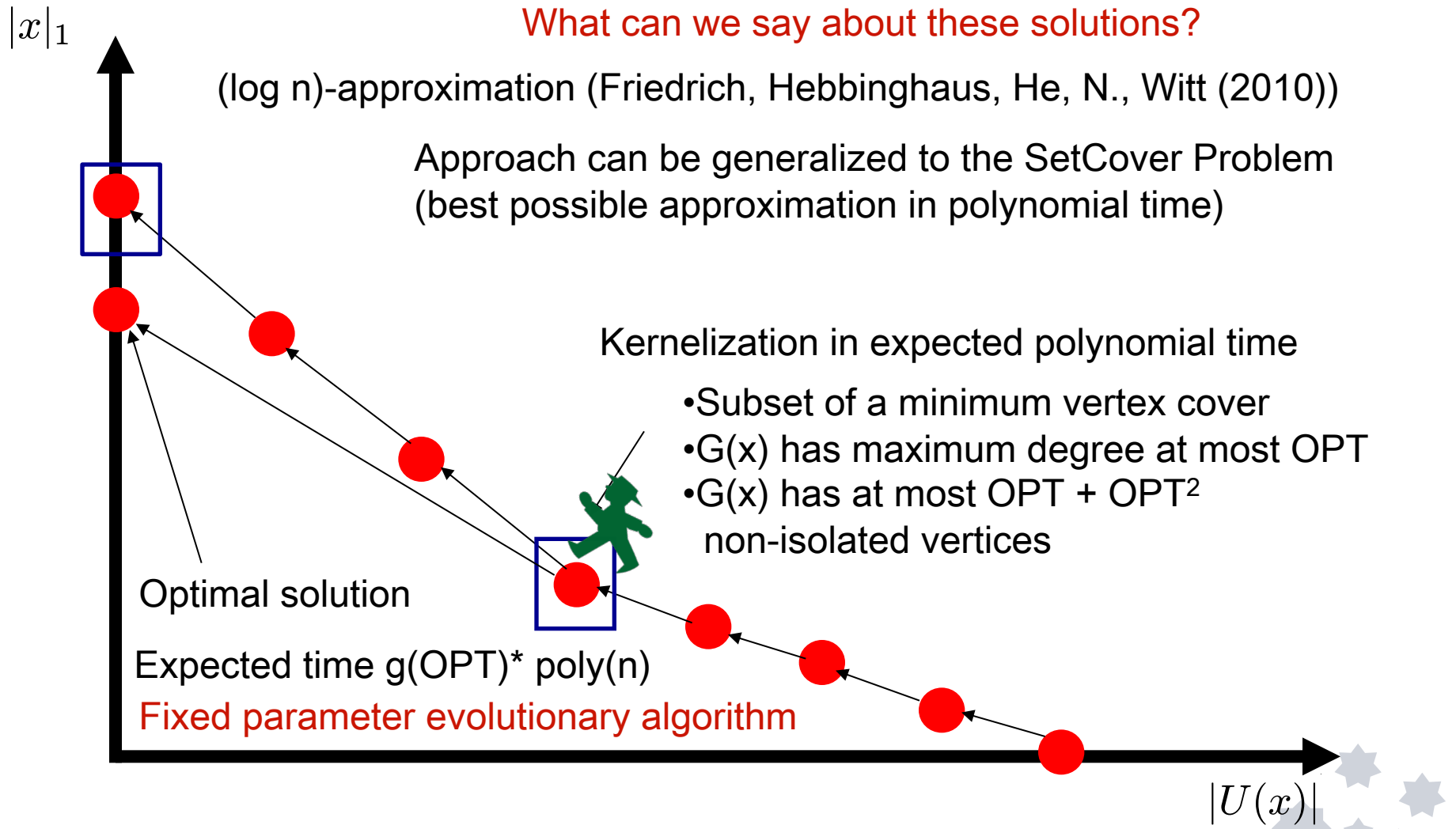


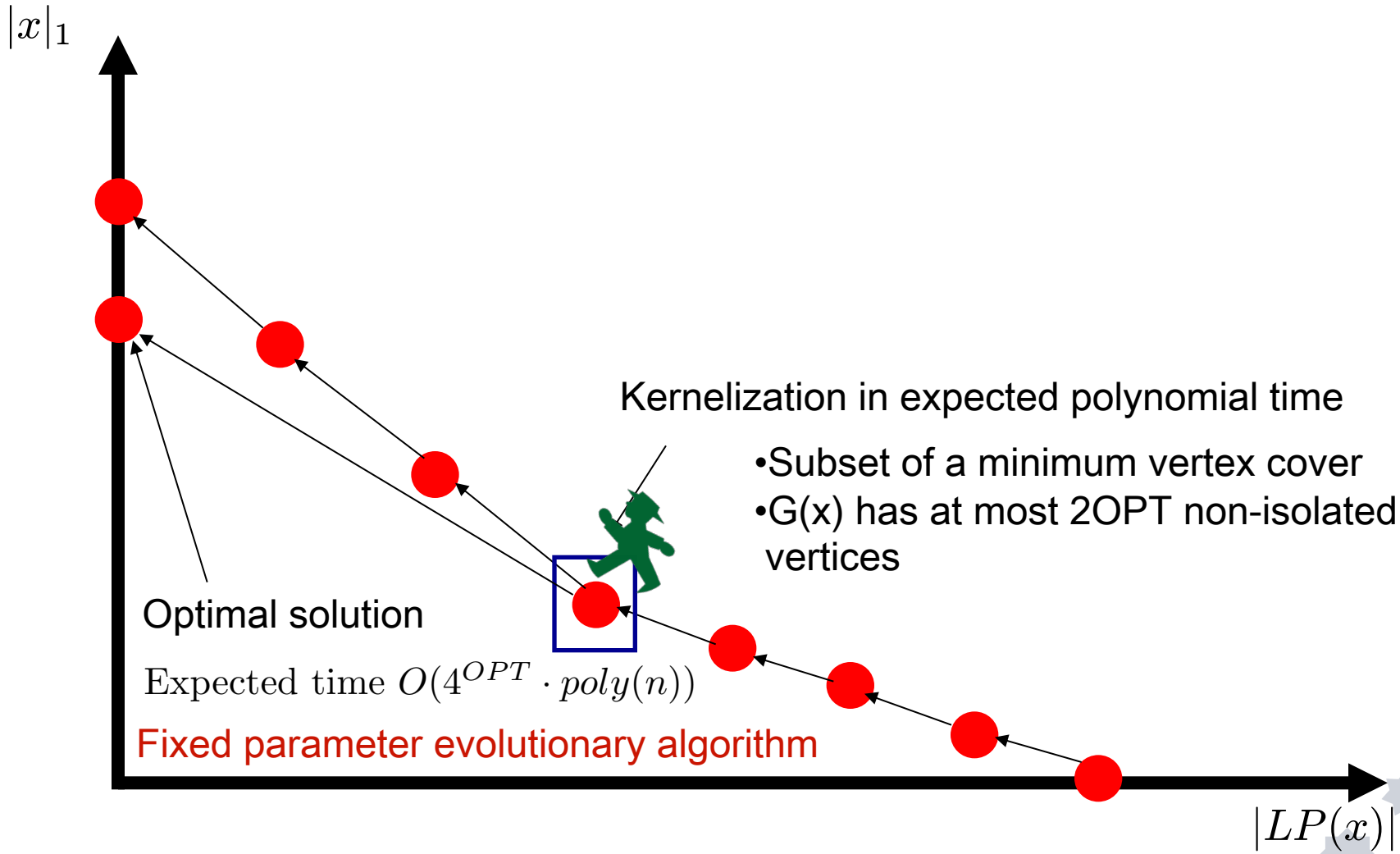
Multi-Objective Approach:

Treat the different objectives in the same way

Keep trade-offs of the two criteria


$|x|_1$
Empty set included
in the population
$|U(x)|$


$|x|_1$
What can we say about these solutions?
(log n)-approximation (Friedrich, Hebbinghaus, He, N., Witt (2010))
Approach can be generalized to the SetCover Problem
(best possible approximation in polynomial time)
Kernelization in expected polynomial time
•Subset of a minimum vertex cover
•G(x) has maximum degree at most OPT
•G(x) has at most OPT + OPT²
non-isolated vertices
Optimal solution
Expected time g(OPT)* poly(n)
Fixed parameter evolutionary algorithm
$|U(x)|$


$|x|_1$
Kernelization in expected polynomial time
•Subset of a minimum vertex cover
•G(x) has at most 2OPT non-isolated vertices
Optimal solution
Expected time $O(4^{OPT} \cdot poly(n))$
Fixed parameter evolutionary algorithm
$|LP(x)|$

Linear Programming

Combination with Linear Programming

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

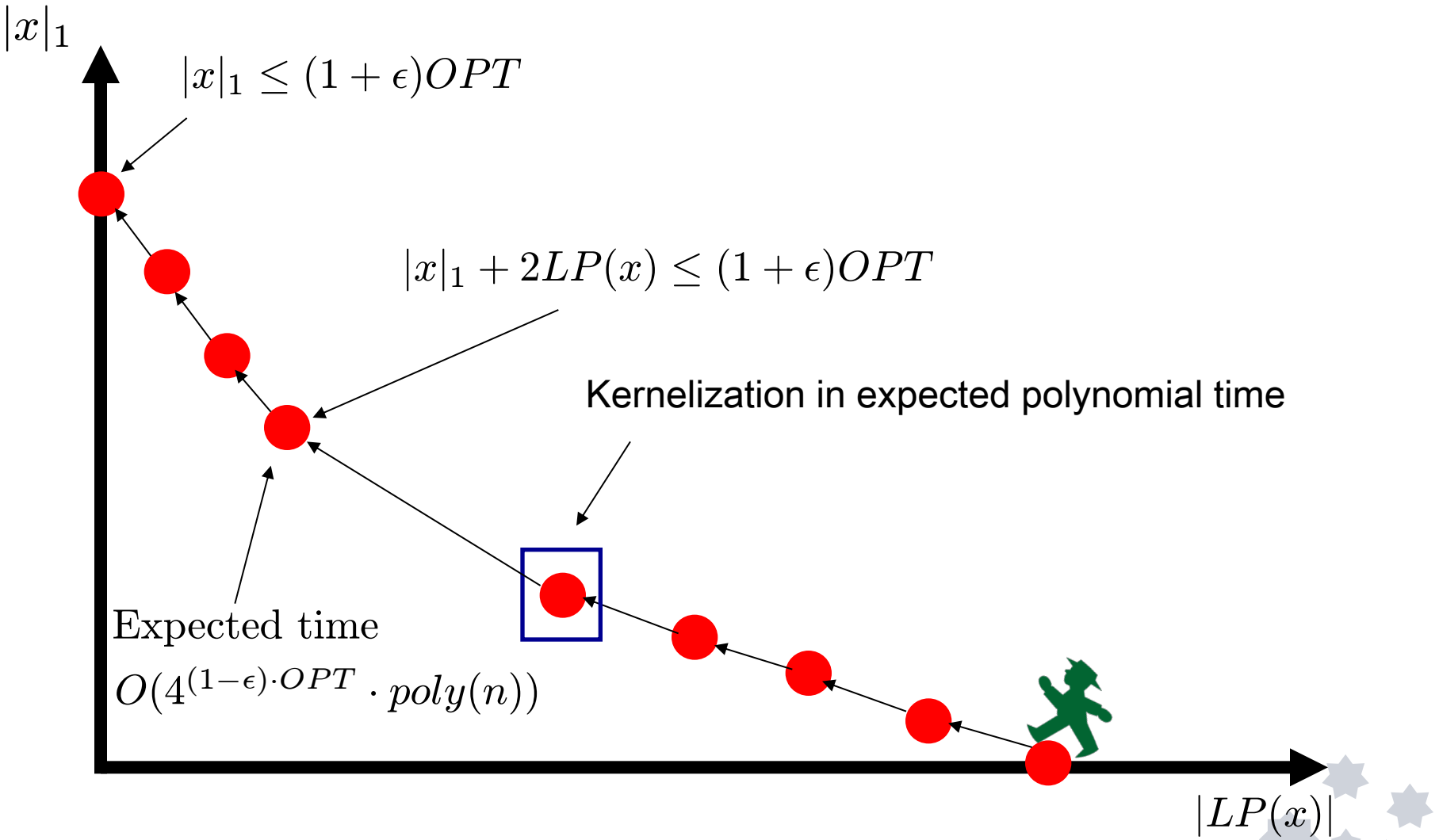
$$x_i \in \{0, 1/2, 1\}, 1 \leq i \leq n$$

Theorem (Nemhauser, 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



Summary

- Evolutionary algorithms are successful for many complex optimization problems.
- Goal is to get a better theoretical understanding.
- There are some nice results for combinatorial optimization.
- Using parameterized analysis looks very promising.

Thank you!

Interesting Problems

- Knapsack (parameter max weight/profit)
- Independent Set (degree/OPT)
- Euclidean TSP (inner points)
- Makespan scheduling (ratio processing times)