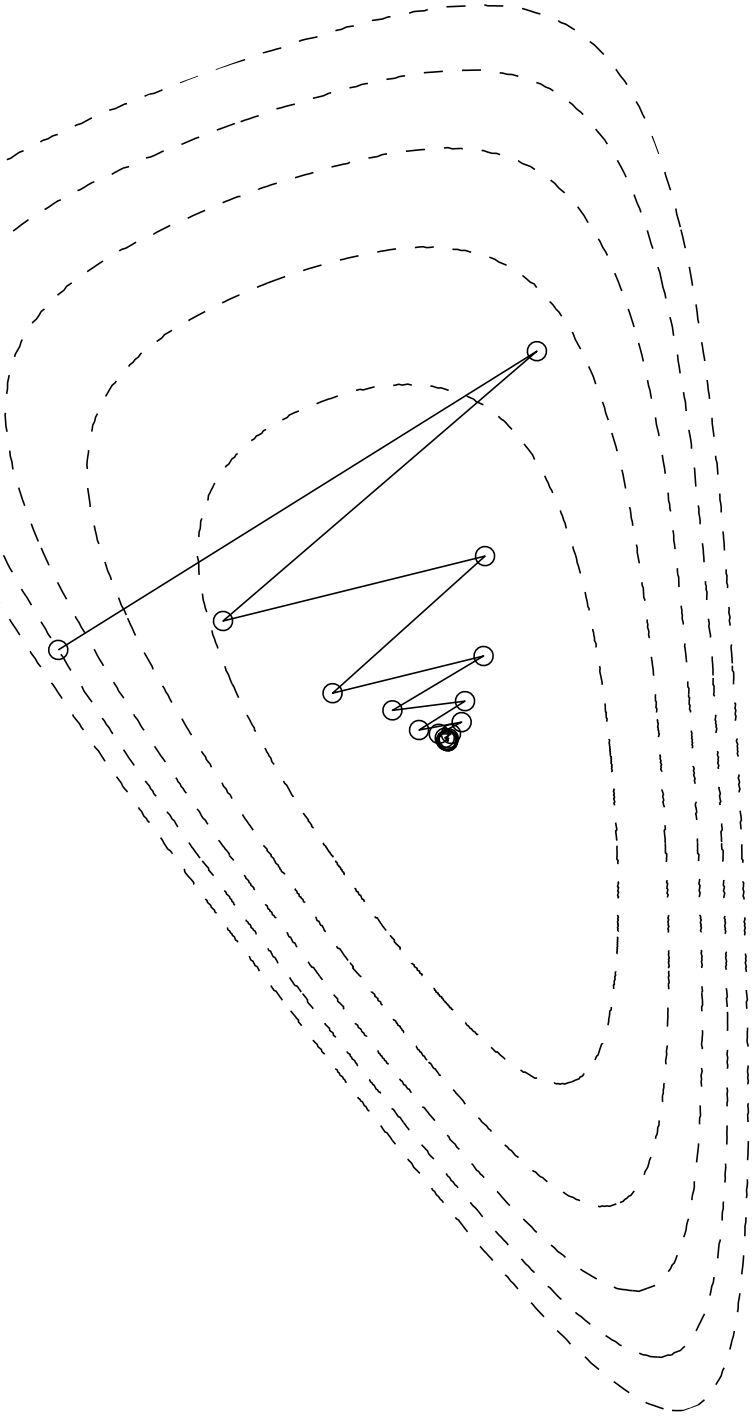


Introduction to Optimization

Blackbox I: Local & Stochastic Search

Marc Toussaint
University of Stuttgart
Summer 2014

“Blackbox Optimization”

- We use the term to denote the problem: Let $x \in \mathbb{R}^n$, $f:\ \mathbb{R}^n \to \mathbb{R}$, find

$$\min_x\ f(x)$$

  where we can *only* evaluate $f(x)$ for any $x \in \mathbb{R}^n$
  $\nabla f(x)$ or $\nabla^2 f(x)$ are not (directly) accessible

- A constrained version: Let $x \in \mathbb{R}^n$, $f:\ \mathbb{R}^n \to \mathbb{R}$, $g:\ \mathbb{R}^n \to \{0,1\}$, find

$$\min_x\ f(x) \quad \text{s.t.} \quad g(x) = 1$$

  where we can only evaluate $f(x)$ and $g(x)$ for any $x \in \mathbb{R}^n$
  I haven’t seen much work on this. Would be interesting to consider this more rigorously.

“Blackbox Optimization” – terminology/subareas

- **Stochastic Optimization** (aka. Stochastic **Search**, Metaheuristics)
  - Simulated Annealing, Stochastic Hill Climing, Tabu Search
  - Evolutionary Algorithms, esp. Evolution Strategies, Covariance Matrix Adaptation, Estimation of Distribution Algorithms
  - Some of them (implicitly or explicitly) locally approximating gradients or 2nd order models

- **Derivative-Free Optimization** (see Nocedal et al.)
  - Methods for (locally) convex/unimodal functions; extending gradient/2nd-order methods
  - Gradient estimation (finite differencing), model-based, Implicit Filtering

- **Bayesian/Global Optimization**
  - Methods for arbitrary (smooth) blackbox functions that get not stuck in local optima.
  - Very interesting domain – close analogies to (active) Machine Learning, bandits, POMDPs, optimal decision making/planning, optimal experimental design

Outline

- Basic downhill running
  - Greedy local search, stochastic local search, simulated annealing
  - Iterated local search, variable neighborhood search, Tabu search
  - Coordinate & pattern search, Nelder-Mead downhill simplex

- Memorize or model something
  - General stochastic search
  - Evolutionary Algorithms, Evolution Strategies, CMA, EDAs
  - Model-based optimization, implicit filtering

- Bayesian/Global optimization: *Learn & approximate optimal optimization*
  - Belief planning view on optimal optimization
  - GPs & Bayesian regression methods for belief tracking
  - bandits, UBC, expected improvement, etc for decision making

Basic downhill running

– Greedy local search, stochastic local search, simulated annealing
– Iterated local search, variable neighborhood search, Tabu search
– Coordinate & pattern search, Nelder-Mead downhill simplex

Greedy local search (greedy downhill, hill climbing)

- Let $x \in \mathcal{X}$ be continuous or discrete
- We assume there is a finite neighborhood $\mathcal{N}(x) \subset \mathcal{X}$ defined for every x

- Greedy local search (variant 1):

Input: initial x, function $f(x)$

1: **repeat**
2: $\quad x \leftarrow \operatorname{argmin}_{y \in \mathcal{N}(x)} f(y)$ *// convention: we assume* $x \in \mathcal{N}(x)$
3: **until** x converges

- Variant 2: $x \leftarrow$ the "first" $y \in \mathcal{N}(x)$ such that $f(y) < f(x)$
- Greedy downhill is a basic ingredient of discrete optimization
- In the continuous case: what is $\mathcal{N}(x)$? Why should it be fixed or finite?

Stochastic local search

- Let $x \in \mathbb{R}^n$
- We assume a "neighborhood" probability distribution $q(y|x)$, typically a Gaussian $q(y|x) \propto \exp\{-\frac{1}{2}(y-x)^\top \Sigma^{-1}(y-x)\}$

Input: initial x, function $f(x)$, proposal distribution $q(y|x)$

1: **repeat**
2: Sample $y \sim q(y|x)$
3: **If** $f(y) < f(x)$ **then** $x \leftarrow y$
4: **until** x converges

- The choice of $q(y|x)$ is crucial, e.g. of the covariance matrix Σ
- Simple heuristic: decrease variance if many steps "fail"; increase variance if sufficient success steps
- Covariance Matrix Adaptation (discussed later) memorizes the recent successful steps and adapts Σ based on this.

Simulated Annealing (run also uphill)

- An extension to avoid getting stuck in local optima is to also accept steps with $f(y) > f(x)$:

Input: initial x, function $f(x)$, proposal distribution $q(y|x)$

1: initialilze the temperature $T = 1$
2: **repeat**
3: Sample $y \sim q(y|x)$
4: Acceptance probability $A = \min\left\{1,\ e^{\frac{f(x)-f(y)}{T}}\ \frac{q(x|y)}{q(y|x)}\right\}$
5: With probability A update $x \leftarrow y$
6: Decrease T, e.g. $T \leftarrow (1-\epsilon)T$ for small ϵ
7: **until** x converges

- Typically: $q(y|x) \propto \exp\{-\frac{1}{2}(y-x)^2/\sigma^2\}$

Simulated Annealing

- Simulated Annealing is a Markov chain Monte Carlo (MCMC) method.
  - Must read!: *An Introduction to MCMC for Machine Learning*
  - These are iterative methods to sample from a distribution, in our case

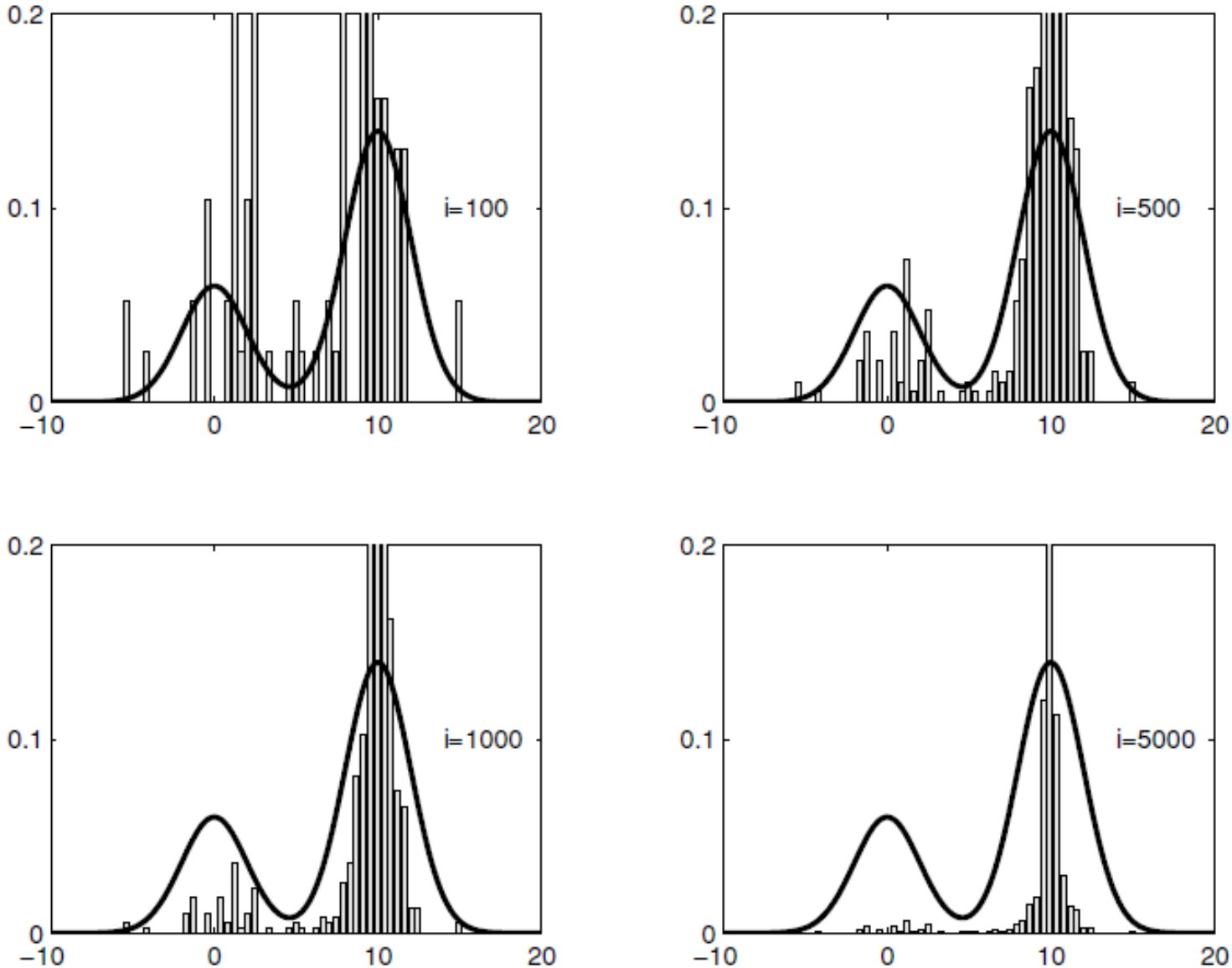
$$p(x) \propto e^{\frac{-f(x)}{T}}$$

- For a fixed temperature T, one can prove that the set of accepted points is distributed as $p(x)$ (but non-i.i.d.!) The acceptance probability

$$A = \min\left\{1, e^{\frac{f(x)-f(y)}{T}} \frac{q(x|y)}{q(y|x)}\right\}$$

  compares the $f(y)$ and $f(x)$, but also the reversibility of $q(y|x)$
- When cooling the temperature, samples focus at the extrema. Guaranteed to sample all extrema *eventually*
- Of high theoretical relevance, less of practical

Simulated Annealing



Random Restarts (run downhill multiple times)

- Greedy local search is typically only used as an ingredient of more robust methods
- We assume to have a start distribution $q(x)$

- Random restarts:

1: **repeat**
2: Sample $x \sim q(x)$
3: $x \leftarrow \texttt{GreedySearch}(x)$ or $\texttt{StochasticSearch}(x)$
4: **If** $f(x) < f(x^*)$ **then** $x^* \leftarrow x$
5: **until** run out of budget

- Greedy local search requires a neighborhood function $\mathcal{N}(x)$
  Stochastic local search requires a transition proposal $q(y|x)$

Iterated Local Search

- Random restarts may be rather expensive, sampling $x \sim q(x)$ is fully uninformed
- Iterated Local Search picks up the last visited local minima x and restarts in a **meta-neighborhood** $\mathcal{N}^*(x)$

- Iterated Local Search (variant 1):

Input: initial x, function $f(x)$

1: **repeat**

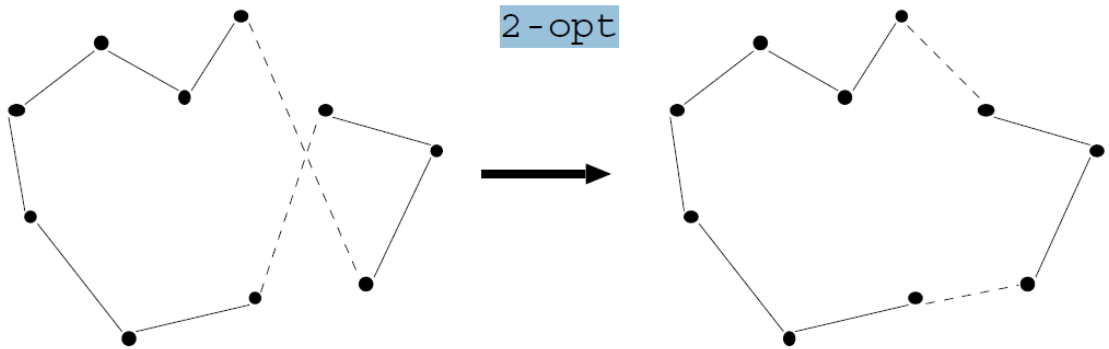
2: $\quad x \leftarrow \operatorname{argmin}_{y' \in \{\texttt{GreedySearch}(y) \,:\, y \in \mathcal{N}^*(x)\}} f(y')$

3: **until** x converges

  - This version evalutes a GreedySearch for all meta-neighbors $y \in \mathcal{N}^*(x)$ of the last local optimum x
  - The inner GreedySearch uses another neighborhood function $\mathcal{N}(x)$
- Variant 2: $x \leftarrow$ the "first" $y \in \mathcal{N}^*(x)$ such that $f(\texttt{GS}(y)) < f(x)$
- Stochastic variant: Neighborhoods $\mathcal{N}(x)$ and $\mathcal{N}^*(x)$ are replaced by transition proposals $q(y|x)$ and $q^*(y|x)$

Iterated Local Search

- Application to Travelling Salesman Problem:
  k-opt neighbourhood: solutions which differ by at most k edges



from Hoos & Stützle: *Tutorial: Stochastic Search Algorithms*

- GreedySearch uses 2-opt or 3-opt neighborhood
  Iterated Local Search uses 4-opt meta-neighborhood (double bridges)

Very briefly...

- Variable Neighborhood Search:
  - Switch the neighborhood function in different phases
  - Similar to Iterated Local Search

- Tabu Search:
  - Maintain a *tabu list* points (or points features) which may not be visited again
  - The list has a fixed finite size: FILO
  - Intensification and diversification heuristics make it more global

Coordinate Search

Input: Initial $x \in \mathbb{R}^n$

1: **repeat**
2: **for** $i = 1, .., n$ **do**
3: $\alpha^* = \text{argmin}_\alpha f(x + \alpha \boldsymbol{e}_i)$ *// Line Search*
4: $x \leftarrow x + \alpha^* \boldsymbol{e}_i$
5: **end for**
6: **until** x converges

- The LineSearch must be approximated
  - E.g. abort on any improvement, when $f(x + \alpha \boldsymbol{e}_i) < f(x)$
  - Remember the last successful stepsize α_i for each coordinate
- Twiddle:

Input: Initial $x \in \mathbb{R}^n$, initial stepsizes α_i for all $i = 1 : n$

1: **repeat**
2: **for** $i = 1, .., n$ **do**
3: $x \leftarrow \text{argmin}_{y \in \{x - \alpha_i e_i, x, x + \alpha_i e_i\}} f(y)$ *// twiddle* x_i
4: Increase α_i if x changed; decrease α_i otherwise
5: **end for**
6: **until** x converges

Pattern Search

- In each iteration k, have a (new) set of search directions $D_k = \{d_{ki}\}$ and test steps of length α_k in these directions
- In each iteration, adapt the search directions D_k and step length α_k

Details: See Nocedal et al.

Nelder-Mead method – Downhill Simplex Method

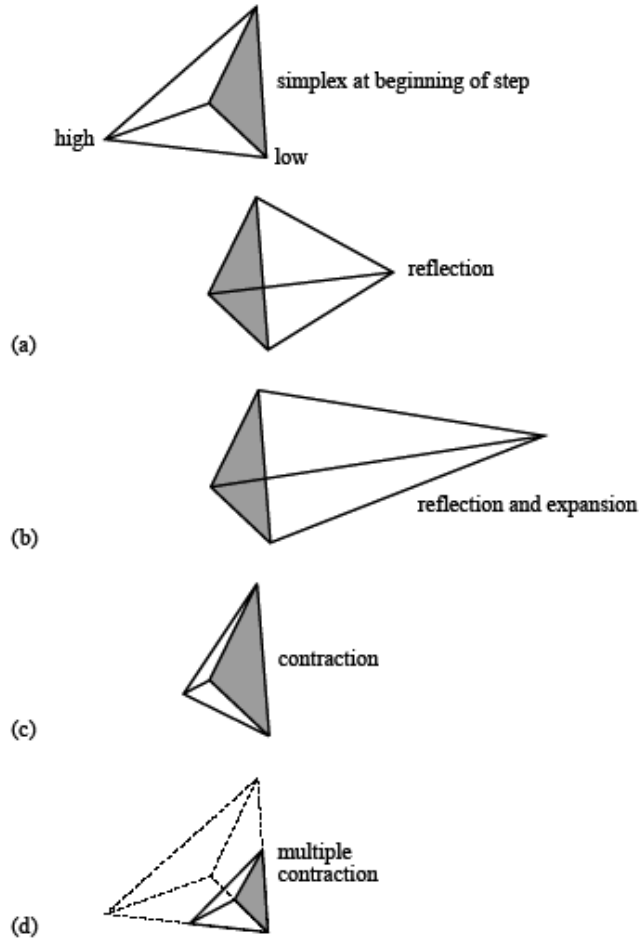


Figure 10.4.1. Possible outcomes for a step in the downhill simplex method. The simplex at the beginning of the step, here a tetrahedron, is shown, top. The simplex at the end of the step can be any one of (a) a reflection away from the high point, (b) a reflection and expansion away from the high point, (c) a contraction along one dimension from the high point, or (d) a contraction along all dimensions towards the low point. An appropriate sequence of such steps will always converge to a minimum of the function.

Nelder-Mead method – Downhill Simplex Method

- Let $x \in \mathbb{R}^n$
- Maintain $n+1$ points $x_0, .., x_n$, sorted by $f(x_0) < ... < f(x_n)$
- Compute center c of points
- Reflect: $y = c + \alpha(c - x_n)$
- If $f(y) < f(x_0)$: Expand: $y = c + \gamma(c - x_n)$
- If $f(y) > f(x_{n\text{-}1})$: Contract: $y = c + \varrho(c - x_n)$
- If still $f(y) > f(x_n)$: Shrink $\forall_{i=1,..,n} x_i \leftarrow x_0 + \sigma(x_i - x_0)$

- Typical parameters: $\alpha = 1, \gamma = 2, \varrho = -\frac{1}{2}, \sigma = \frac{1}{2}$

Summary: Basic downhill running

- These methods are highly relevant! Despite their simplicity
- Essential ingredient to iterative approaches that try to find as many local minima as possible

- Methods essentially differ in the notion of *neighborhood, transition proposal, or pattern of next search points* to consider
- Iterated downhill can be very effective

- However: **There should be ways to better exploit data!**
  - Learn from previous evaluations where to test new point
  - Learn from previous local minima where to restart

Memorize or model something

– Stochastic search schemes
– Evolutionary Algorithms, Evolution Strategies, CMA, EDAs
– Model-based optimization, implicit filtering

A general stochastic search scheme

- The general scheme:
  - The algorithm maintains a probability distribution $p_\theta(x)$
  - In each iteration it takes n samples $\{x_i\}_{i=1}^n \sim p_\theta(x)$
  - Each x_i is evaluated $\rightarrow$ data $\{(x_i, f(x_i))\}_{i=1}^n$
  - **That data is used to update θ**

Input: initial parameter θ, function $f(x)$, distribution model $p_\theta(x)$, update heuristic $h(\theta, D)$

Output: final θ and best point x

1: **repeat**
2: Sample $\{x_i\}_{i=1}^n \sim p_\theta(x)$
3: Evaluate samples, $D = \{(x_i, f(x_i))\}_{i=1}^n$
4: Update $\theta \leftarrow h(\theta, D)$
5: **until** θ converges

Example: Gaussian search distribution "(μ, λ)-ES"

From 1960s/70s. Rechenberg/Schwefel

- The simplest distribution family

$$\theta = (\hat{x}) \ , \quad p_\theta(x) = \mathcal{N}(x \,|\, \hat{x}, \sigma^2)$$

  a n-dimenstional isotropic Gaussian with fixed variance σ^2

- Update heuristic:
  - Given $D = \{(x_i, f(x_i))\}_{i=1}^{\lambda}$, select μ best: $D' = \text{bestOf}_\mu(D)$
  - Compute the new mean $\hat{x}$ from D'

- This algorithm is called "Evolution Strategy (μ, λ)-ES"
  - The Gaussian is meant to represent a "species"
  - λ offspring are generated
  - the best μ selected

θ is the "knowledge/information" gained

- The parameter θ is the only "knowledge/information" that is being propagated between iterations
  θ encodes what has been learned from the history
  θ defines where to search in the future

- The downhill methods of the previous section did not store any information other than the current x. (Exception: Tabu search, Nelder-Mead)

- Evolutionary Algorithms are a special case of this stochastic search scheme

Evolutionary Algorithms (EAs)

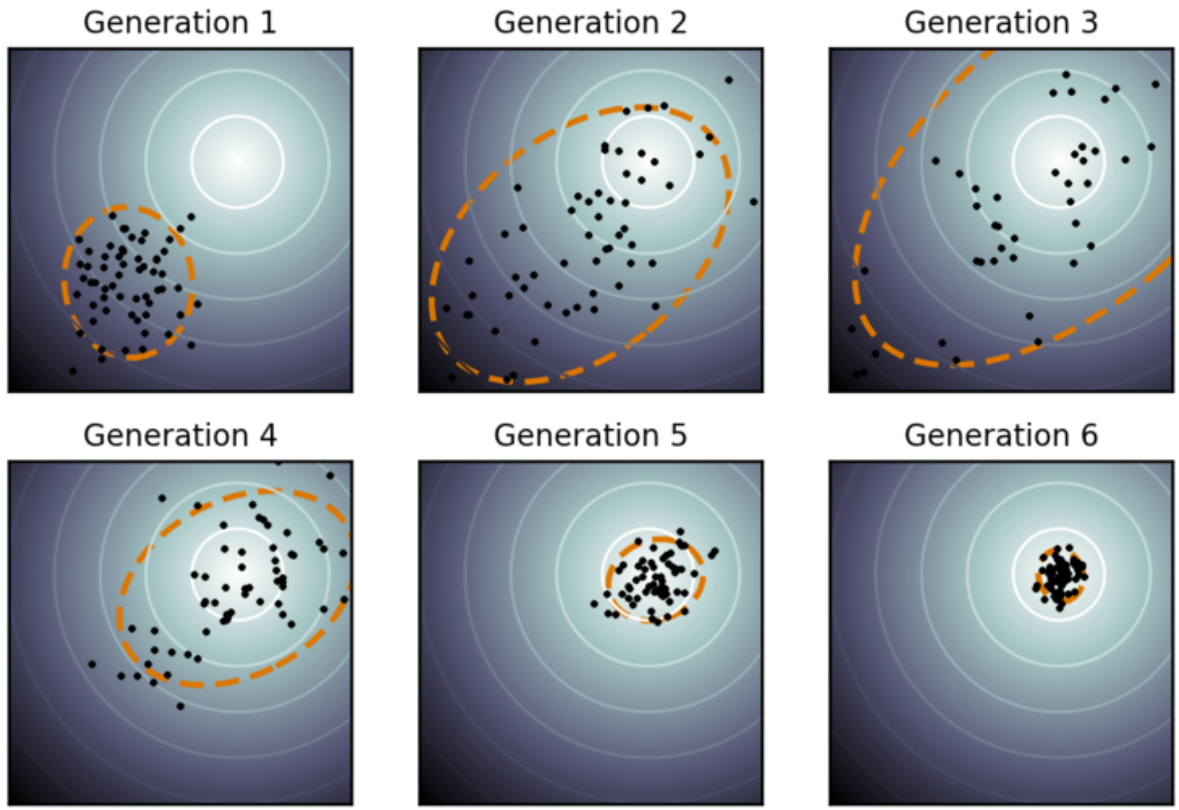
- EAs can well be described as special kinds of parameterizing $p_\theta(x)$ and updating θ
  - The θ typically is a set of good points found so far (parents)
  - Mutation & Crossover define $p_\theta(x)$
  - The samples D are called offspring
  - The θ-update is often a selection of the best, or "fitness-proportional" or rank-based

- Categories of EAs:
  - **Evolution Strategies**: $x \in \mathbb{R}^n$, often Gaussian $p_\theta(x)$
  - **Genetic Algorithms**: $x \in \{0,1\}^n$, crossover & mutation define $p_\theta(x)$
  - **Genetic Programming**: x are programs/trees, crossover & mutation
  - **Estimation of Distribution Algorithms**: θ directly defines $p_\theta(x)$

Covariance Matrix Adaptation (CMA-ES)

- An obvious critique of the simple Evolution Strategies:
  - The search distribution $\mathcal{N}(x \mid \hat{x}, \sigma^2)$ is isotropic (no going *forward*, no preferred direction)
  - The variance σ is fixed!

- Covariance Matrix Adaptation Evolution Strategy (CMA-ES)



Covariance Matrix Adaptation (CMA-ES)

- In Covariance Matrix Adaptation

$$\theta = (\hat{x}, \sigma, C, \varrho_\sigma, \varrho_C) \ , \quad p_\theta(x) = \mathcal{N}(x \,|\, \hat{x}, \sigma^2 C)$$

  where C is the covariance matrix of the search distribution
- The θ maintains two more pieces of information: ϱ_σ and ϱ_C capture the "path" (motion) of the mean $\hat{x}$ in recent iterations
- Rough outline of the θ-update:
  - Let $D' = \text{bestOf}_\mu(D)$ be the set of selected points
  - Compute the new mean $\hat{x}$ from D'
  - Update ϱ_σ and ϱ_C proportional to $\hat{x}_{k+1} - \hat{x}_k$
  - Update σ depending on $|\varrho_\sigma|$
  - Update C depending on $\varrho_c \varrho_c^\top$ (rank-1-update) and $\text{Var}(D')$

CMA references

Hansen, N. (2006), ”The CMA evolution strategy: a comparing review”
Hansen et al.: Evaluating the CMA Evolution Strategy on Multimodal Test Functions, PPSN 2004.

Function	f_{stop}	init	n	CMA-ES	DE	RES	LOS
$f_{\text{Ackley}}(x)$	1e-3	$[-30, 30]^n$	20	**2667**	.	.	6.0e4
			30	**3701**	12481	1.1e5	9.3e4
			100	**11900**	36801	.	.
$f_{\text{Griewank}}(x)$	1e-3	$[-600, 600]^n$	20	**3111**	8691	.	.
			30	**4455**	11410 *	*8.5e-3/2e5*	.
			100	**12796**	31796	.	.
$f_{\text{Rastrigin}}(x)$	0.9	$[-5.12, 5.12]^n$	20	68586	**12971**	.	9.2e4
		DE: $[-600, 600]^n$	30	147416	**20150** *	1.0e5	2.3e5
			100	1010989	**73620**	.	.
$f_{\text{Rastrigin}}(\boldsymbol{A}x)$	0.9	$[-5.12, 5.12]^n$	30	**152000**	*171/1.25e6* *	.	.
			100	**1011556**	*944/1.25e6* *	.	.
$f_{\text{Schwefel}}(x)$	1e-3	$[-500, 500]^n$	5	43810	**2567** *	.	7.4e4
			10	240899	**5522** *	.	5.6e5

- For “large enough” populations local minima are avoided

CMA conclusions

- It is a good starting point for an off-the-shelf blackbox algorithm
- It includes components like estimating the local gradient (ϱ_σ, ϱ_C), the local "Hessian" (Var(D')), smoothing out local minima (large populations)

Estimation of Distribution Algorithms (EDAs)

- Generally, EDAs fit the distribution $p_\theta(x)$ to model the distribution of previously good search points
  For instance, if in all previous distributions, the 3rd bit equals the 7th bit, then the search distribution $p_\theta(x)$ should put higher probability on such candidates.
  $p_\theta(x)$ is meant to capture the *structure* in previously good points, i.e. the dependencies/correlation between variables.

- A rather successful class of EDAs on discrete spaces uses graphical models to learn the dependencies between variables, e.g. Bayesian Optimization Algorithm (BOA)

- In continuous domains, CMA is an example for an EDA

Stochastic search conclusions

Input: initial parameter θ, function $f(x)$, distribution model $p_\theta(x)$, update heuristic $h(\theta, D)$

Output: final θ and best point x

1: **repeat**
2: Sample $\{x_i\}_{i=1}^n \sim p_\theta(x)$
3: Evaluate samples, $D = \{(x_i, f(x_i))\}_{i=1}^n$
4: Update $\theta \leftarrow h(\theta, D)$
5: **until** θ converges

- The framework is very general
- The crucial difference between algorithms is their choice of $p_\theta(x)$

Model-based optimization

following Nodecal et al. “Derivative-free optimization”

Model-based optimization

- The previous stochastic serach methods are heuristics to update θ

Why not store the previous data directly?

- Model-based optimization takes the approach
  - Store a data set $\theta = D = \{(x_i, y_i)\}_{i=1}^n$ of previously explored points (let $\hat{x}$ be the current minimum in D)
  - Compute a (quadratic) model $D \mapsto \hat{f}(x) = \phi_2(x)^\top \beta$
  - Choose the next point as

$$x^+ = \operatorname*{argmin}_x \hat{f}(x) \quad \text{s.t.} \quad |x - \hat{x}| < \alpha$$

  - Update D and α depending on $f(x^+)$
- The argmin is solved with constrained optimization methods

Model-based optimization

Initialize D with at least $\frac{1}{2}(n+1)(n+2)$ data points
repeat
  Compute a regression $\hat{f}(x) = \phi_2(x)^\top \beta$ on D
  Compute $x^+ = \operatorname{argmin}_x \hat{f}(x) \quad \text{s.t.} \quad |x - \hat{x}| < \alpha$
  Compute the improvement ratio $\varrho = \frac{f(\hat{x}) - f(x^+)}{\hat{f}(\hat{x}) - \hat{f}(x^+)}$
  if $\varrho > \epsilon$ **then**
    Increase the stepsize α
    Accept $\hat{x} \leftarrow x^+$
    Add to data, $D \leftarrow D \cup \{(x^+, f(x^+))\}$
  else
    if $\det(D)$ is too small **then** *// Data improvement*
      Compute $x^+ = \operatorname{argmax}_x \det(D \cup \{x\}) \quad \text{s.t.} \quad |x - \hat{x}| < \alpha$
      Add to data, $D \leftarrow D \cup \{(x^+, f(x^+))\}$
    else
      Decrease the stepsize α
    end if
  end if
  Prune the data, e.g., remove $\operatorname{argmax}_{x \in \Delta} \det(D \setminus \{x\})$
until x converges

- **Variant:** Initialize with only $n+1$ data points and fit a linear model as long as $|D| < \frac{1}{2}(n+1)(n+2) = \dim(\phi_2(x))$

Model-based optimization

- Optimal parameters (with data matrix $X \in \mathbb{R}^{n \times \dim(\beta)}$)

$$\hat{\beta}^{\text{ls}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top y$$

  The determinant $\det(\boldsymbol{X}^\top \boldsymbol{X})$ or $\det(\boldsymbol{X})$ (denoted $\det(D)$ on the previous slide) is a measure for well the data supports the regression. The data improvement explicitly selects a next evaluation point to increase $\det(D)$.

- Nocedal describes in more detail a geometry-improving procedure to update D.

- Model-based optimization is closely related to Bayesian approaches. But
  - Should we really prune data to have only a minimal set D (of size $\dim(\beta)$?)
  - Is there another way to think about the "data improvement" selection of x^+? ($\rightarrow$ maximizing uncertainty/information gain)

Implicit Filtering (briefly)

- Estimates the local gradient using finite differencing

$$\nabla_\epsilon f(x) \approx \left[\frac{1}{2\epsilon}(f(x+\epsilon e_i) - f(x-\epsilon e_i))\right]_{i=1,..,n}$$

- Lines search along the gradient; if not succesful, decrease ϵ
- Can be extended by using $\nabla_\epsilon f(x)$ to update an approximation of the Hessian (as in BFGS)

Conclusions

- We covered
  - "downhill running"
  - Two flavors of methods that exploit the recent data:
    – stochastic search (& EAs), maintaining θ that defines $p_\theta(x)$
    – model-based opt., maintaining local data D that defines $\hat{f}(x)$

- These methods can be very efficient, but somehow the problem formalization is unsatisfactory:
  - What would be optimal optimization?
  - What exactly is the information that we can gain from data about the optimum?
  - If the optimization algorithm would be an "AI agent", selecting points his actions, seeing $f(x)$ his observations, what would be his optimal decision making strategy?
  - And what about **global** blackbox optimization?