SAND2006-5209C

An asynchronous parallel derivative-free algorithm for handling general constraints

Josh Griffin

Computational Sciences and Mathematics Research
Sandia National Laboratories
Livermore, California USA

Second International Congress on Mathematical Software
Castro Urdiales, SPAIN
September 1–3, 2006

Joint work with Tammy Kolda, Robert Michael Lewis, and Virginia Torczon

Talk outline

1. Problems of interest
2. Generating set search background
3. Linear constraints
4. Nonlinear equality constraints
5. Numerical results

Why use derivative-free?

Answer: Sometimes you don't have choice

Derivative-based if ...	Derivative-free if ...
• Function evaluations quick	• Function evaluations slow
• All points in Ω finite/defined	• Points in Ω may be undefined
• Continous and smooth in Ω	• Discontinous, nonsmooth, okay
• Little to no noise	• Noise okay
• Looking for nearest local min	• Wanting something more global

Should I take the

or the

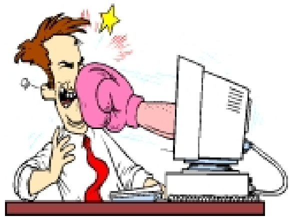
?

Derivative-based methods place stronger restrictions on $f(x)$ and Ω but require fewer function evaluation to reach solution

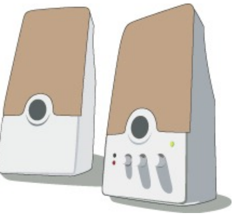
Problems we are interested in

- Function evaluations CPU-intensive, often a single evaluations requires multiple processors and may take hours/days to compute

- The objective is often based upon large simulation based codes that can periodically crash, returning an undefined point



- If derivatives exists, noise limits ability to estimate

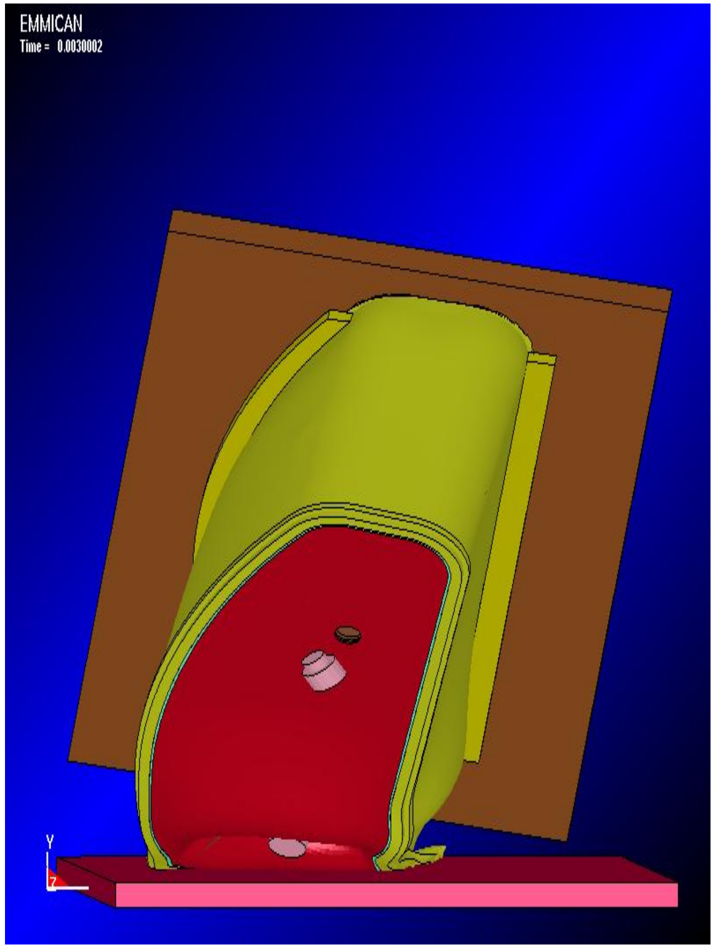


- Because function evaluations are simulation-based, access to objective exists through shell script interfaces

Sandia optimization problem (supporting nuclear safety studies)

Goal: *Determine if accidental drop could jeopardize integrity of internal components.*

1. Model developed to simulate drop from different angles.
2. Optimization problem: determine angle that maximizes damage.
3. Single function eval involves:
   - Rotating/remeshing: 2-5 min.
   - Simulating drop: 1 to 15 hrs.



Generating Set Search and APPSPACK

APPSPACK developed for following problem types

We will consider problems of the form

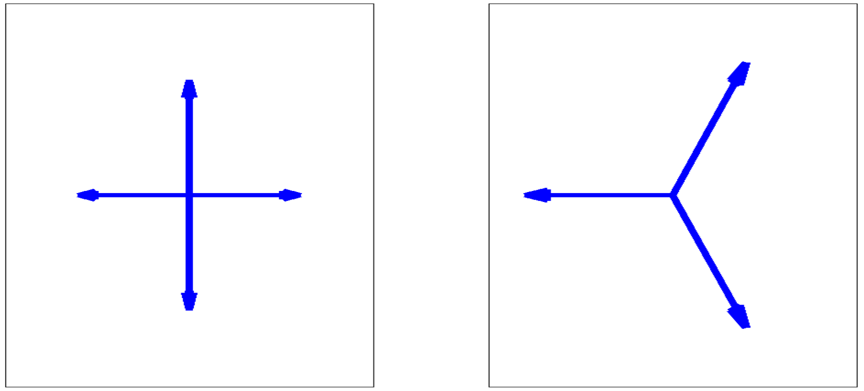
$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & f(x) \\ \text{subject to} & c(x) = 0 \\ & Ax \leq b \end{array}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $c : \mathbb{R}^n \rightarrow \mathbb{R}^p$, and A is an $m \times n$ matrix.

- linear equalities permitted
- derivatives unavailable
- number of variables relatively small (≤ 100)

Generating set search algorithms

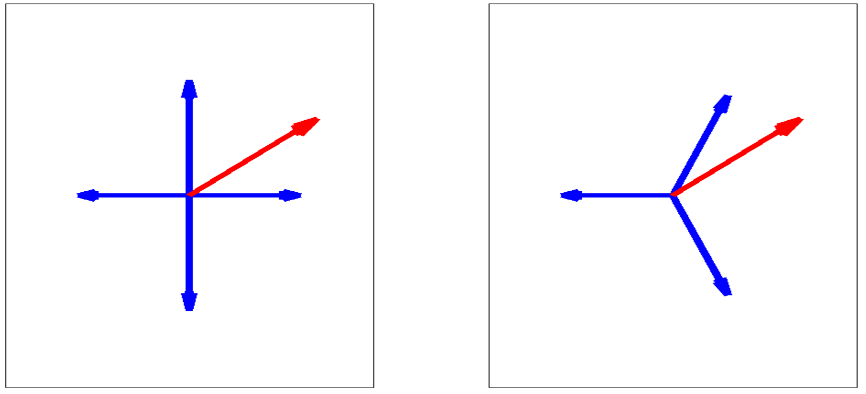
Generating set search algorithms explore the feasible region with a set of search directions



positively spanning $\mathbb{R}^n$ (in unconstrained case), with the property that no matter where the direction of steepest descent lies in $\mathbb{R}^n$, at least one search direction lies within $90°$.

Generating set search algorithms

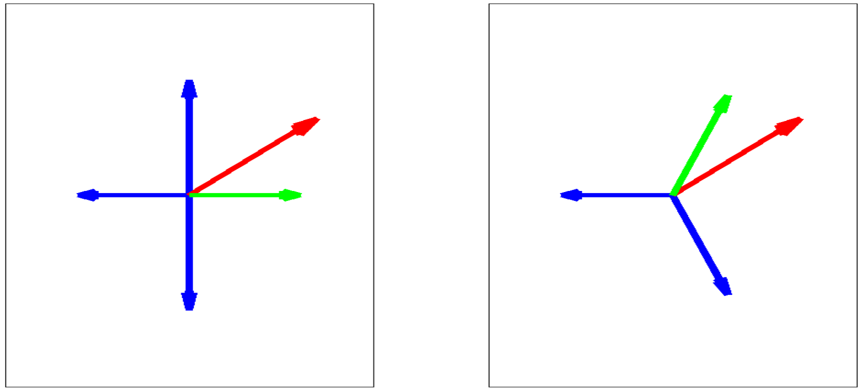
Generating set search algorithms explore the feasible region with a set of search directions



positively spanning $\mathbb{R}^n$ (in unconstrained case), with the property that no matter where the direction of steepest descent lies in $\mathbb{R}^n$, at least one search direction lies within 90°.

Generating set search algorithms

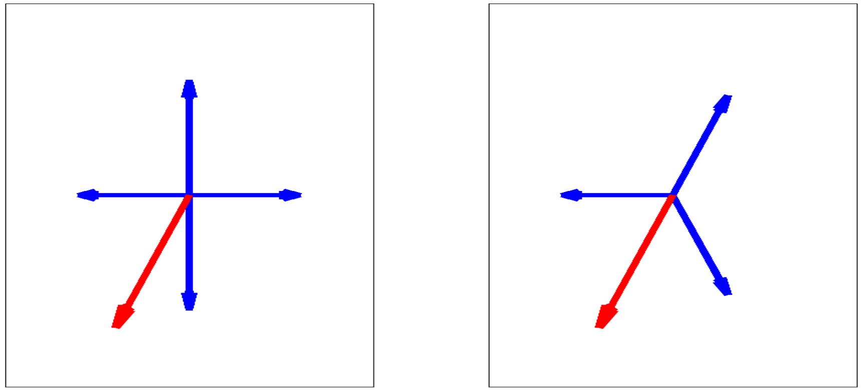
Generating set search algorithms explore the feasible region with a set of search directions



positively spanning $\mathbb{R}^n$ (in unconstrained case), with the property that no matter where the direction of steepest descent lies in $\mathbb{R}^n$, at least one search direction lies within $90°$.

Generating set search algorithms

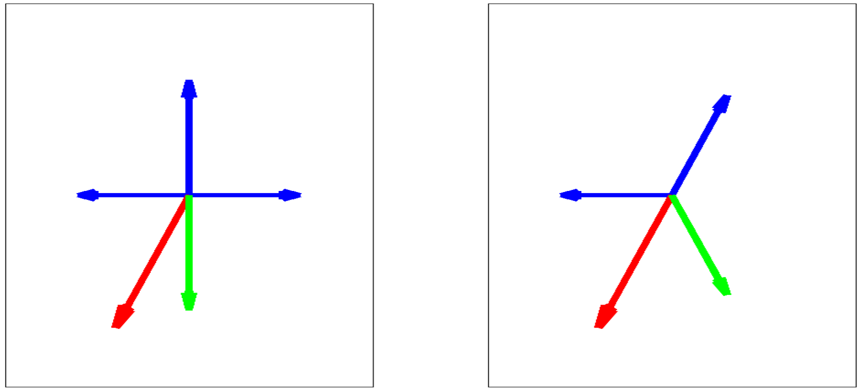
Generating set search algorithms explore the feasible region with a set of search directions



positively spanning $\mathbb{R}^n$ (in unconstrained case), with the property that no matter where the direction of steepest descent lies in $\mathbb{R}^n$, at least one search direction lies within $90°$.

Generating set search algorithms

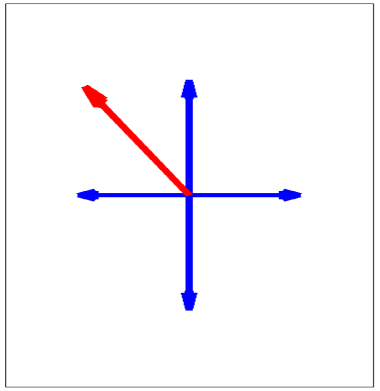
Generating set search algorithms explore the feasible region with a set of search directions

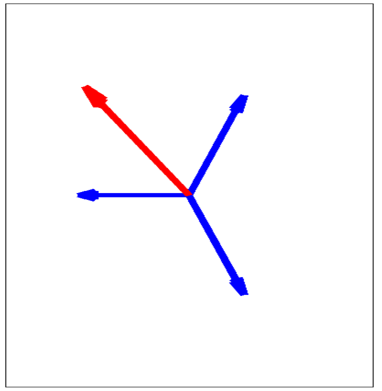


positively spanning $\mathbb{R}^n$ (in unconstrained case), with the property that no matter where the direction of steepest descent lies in $\mathbb{R}^n$, at least one search direction lies within $90°$.

Generating set search algorithms

Generating set search algorithms explore the feasible region with a set of search directions

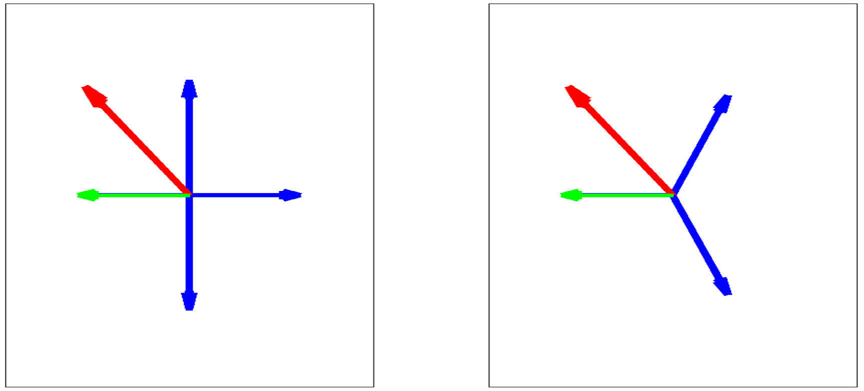




positively spanning $\mathbb{R}^n$ (in unconstrained case), with the property that no matter where the direction of steepest descent lies in $\mathbb{R}^n$, at least one search direction lies within 90°.

Generating set search algorithms

Generating set search algorithms explore the feasible region with a set of search directions



positively spanning $\mathbb{R}^n$ (in unconstrained case), with the property that no matter where the direction of steepest descent lies in $\mathbb{R}^n$, at least one search direction lies within 90°.

This property ensures us that if derivative's happen to exists we will converge to a local minimum.

Basic synchronous framework (unconstrained)

- Trial point generation:

$$\mathcal{X} = \{x + \Delta d^{(i)} : d^{(i)} \in \text{search pattern}\}$$

  and send to evaluation queue.

- Trial point evaluation: Collect evaluated points $\mathcal{Y} = \mathcal{X}$.
- Decision: If a point $y \in \mathcal{Y}$ is determined to be "better than" x, iteration is considered successful.
- Successful: $x \leftarrow y$
- Unsuccessful: $\Delta \leftarrow .5\Delta$
- Stop: if $\Delta < \Delta_{\text{tol}}$

Basic synchronous framework (unconstrained)

- Trial point generation:

$$\mathcal{X} = \{x + \Delta d^{(i)} : d^{(i)} \in \text{search pattern}\}$$

  and send to evaluation queue.

- Trial point evaluation: Collect evaluated points $\mathcal{Y} = \mathcal{X}$.
- Decision: If a point $y \in \mathcal{Y}$ is determined to be "better than" x, iteration is considered successful.
- Successful: $x \leftarrow y$
- Unsuccessful: $\Delta \leftarrow .5\Delta$
- Stop: if $\Delta < \Delta_{\text{tol}}$

We enforce a sufficient decrease conditions based on step size Δ

$$f(y) \leq f(x) - \alpha\Delta^2$$

Basic synchronous framework (unconstrained)

- Trial point generation:

$$\mathcal{X} = \{x + \Delta d^{(i)} : d^{(i)} \in \text{search pattern}\}$$

  and send to evaluation queue.

- Trial point evaluation: Collect evaluated points $\mathcal{Y} = \mathcal{X}$.
- Decision: If a point $y \in \mathcal{Y}$ is determined to be "better than" x, iteration is considered successful.
- Successful: $x \leftarrow y$
- Unsuccessful: $\Delta \leftarrow .5\Delta$
- Stop: if $\Delta < \Delta_{\text{tol}}$

Step where asynchronous algorithms wins in parallel $\mathcal{Y} \neq \mathcal{X}$

Asynchronous framework (unconstrained)

- Trial point generation:

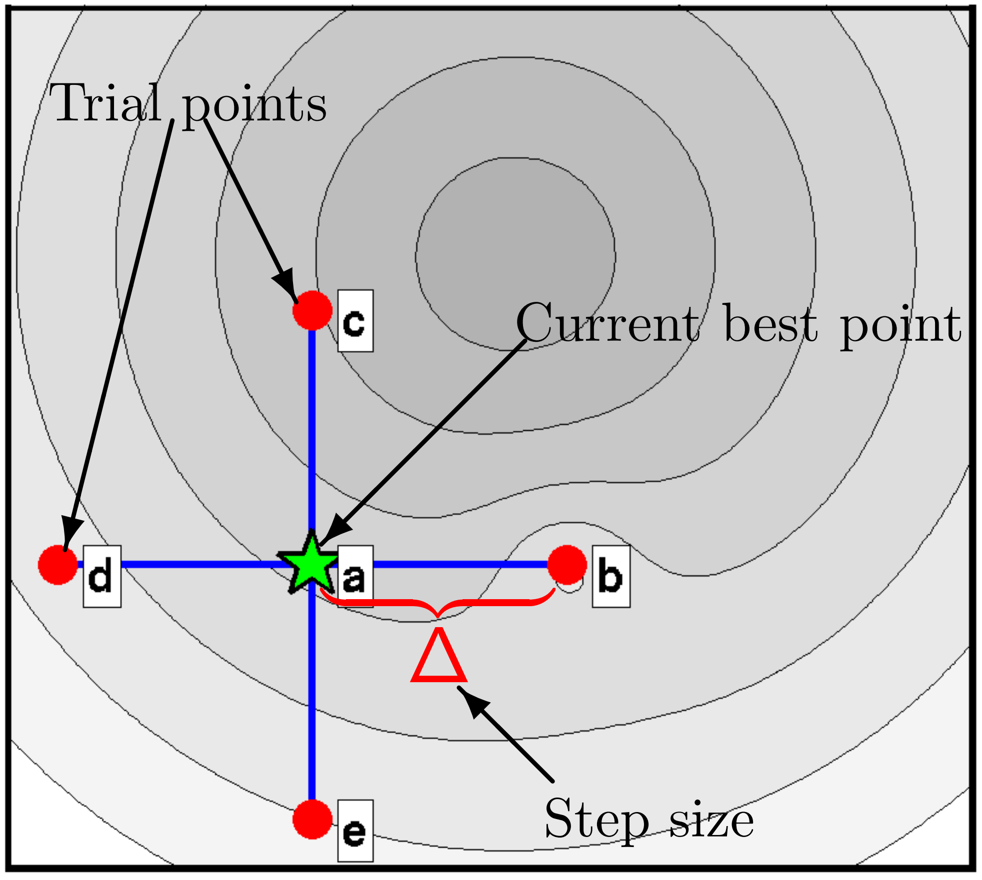
$$\mathcal{X} = \{x + \Delta^{(i)} d^{(i)} : d^{(i)} \in \text{search pattern and inactive}\}$$

  and submit to the evaluation queue.

- Trial point evaluation: Collect a nonempty set of evaluated point $\mathcal{Y}$.
- Decision: If a point $y \in \mathcal{Y}$ is determined to be "better than" x, iteration is considered successful.
- Successful: $x \leftarrow y$, reset $\Delta^{(i)} = \max(\Delta_{\text{min}}, \text{step that generated } y)$. Prune evaluation queue.
- Unsuccessful: $\Delta^{(i)} \leftarrow .5\Delta^{(i)}$ for all direction indices corresponding to points in $\mathcal{Y}$.
- Stop: If $\Delta^{(i)} < \Delta_{\text{tol}}$ for all i

Here Δ_{min} denotes minimum step-size. Must be $\geq \Delta_{\text{tol}}$.

Unconstrained optimization demo



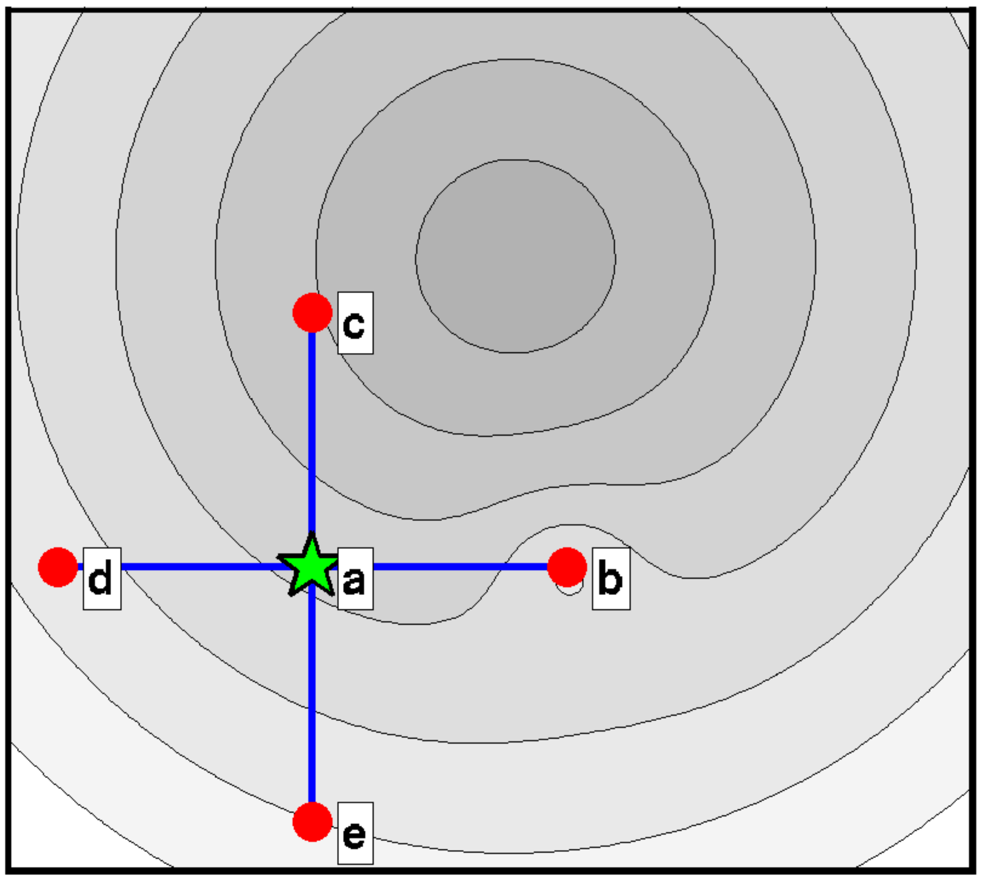
best: **a**

pending: **b c d e**

evaluated:

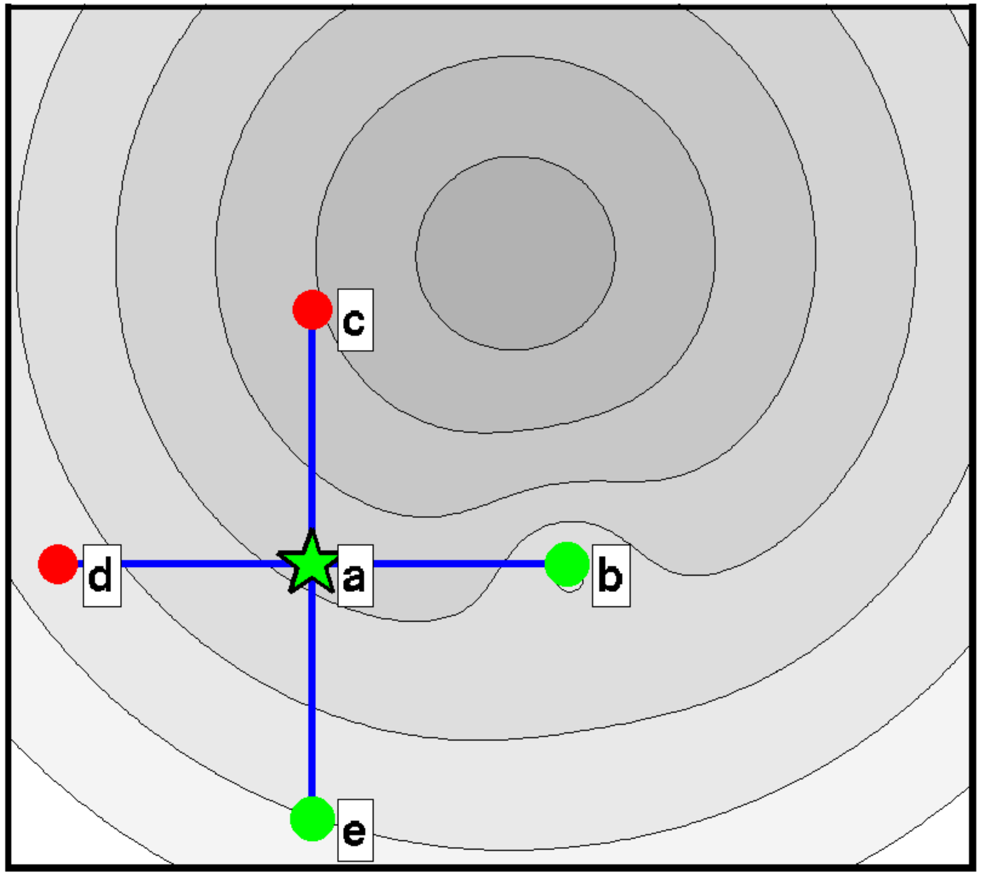
pruned:

Unconstrained optimization demo



best:	**a**
pending:	**b c d e**
evaluated:	
pruned:	

Unconstrained optimization demo



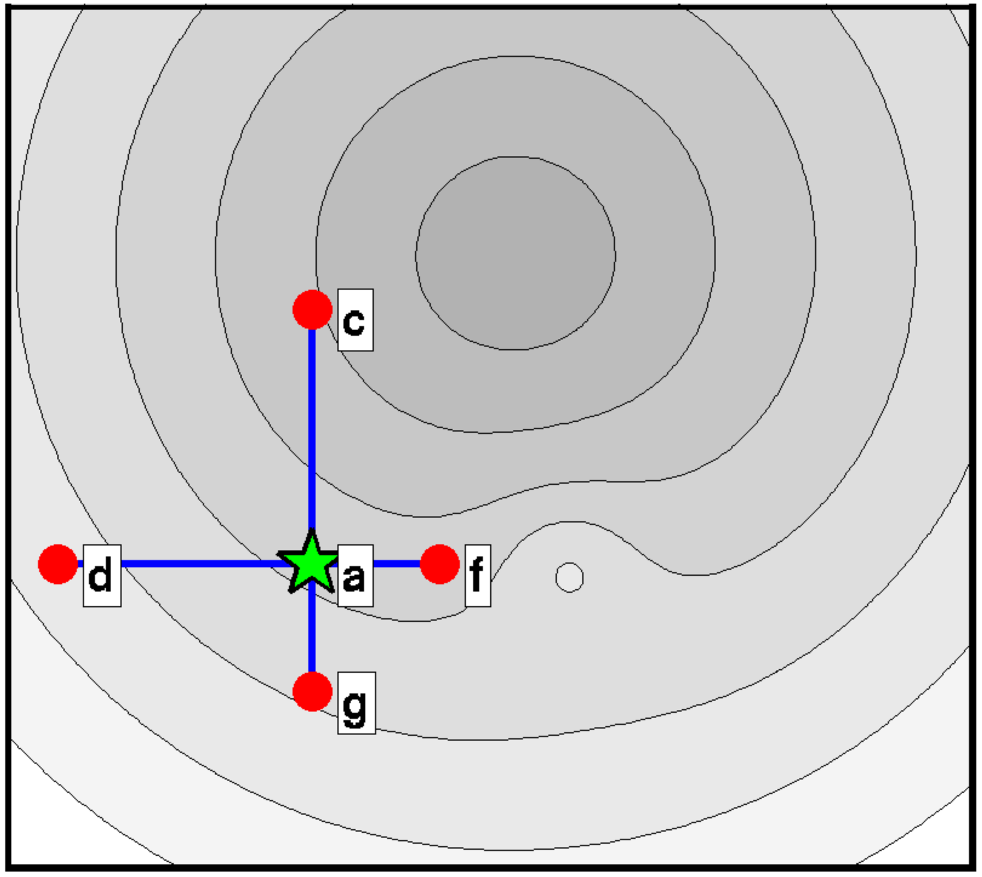
best: **a**

pending: **c d**

evaluated: **b e**

pruned:

Unconstrained optimization demo



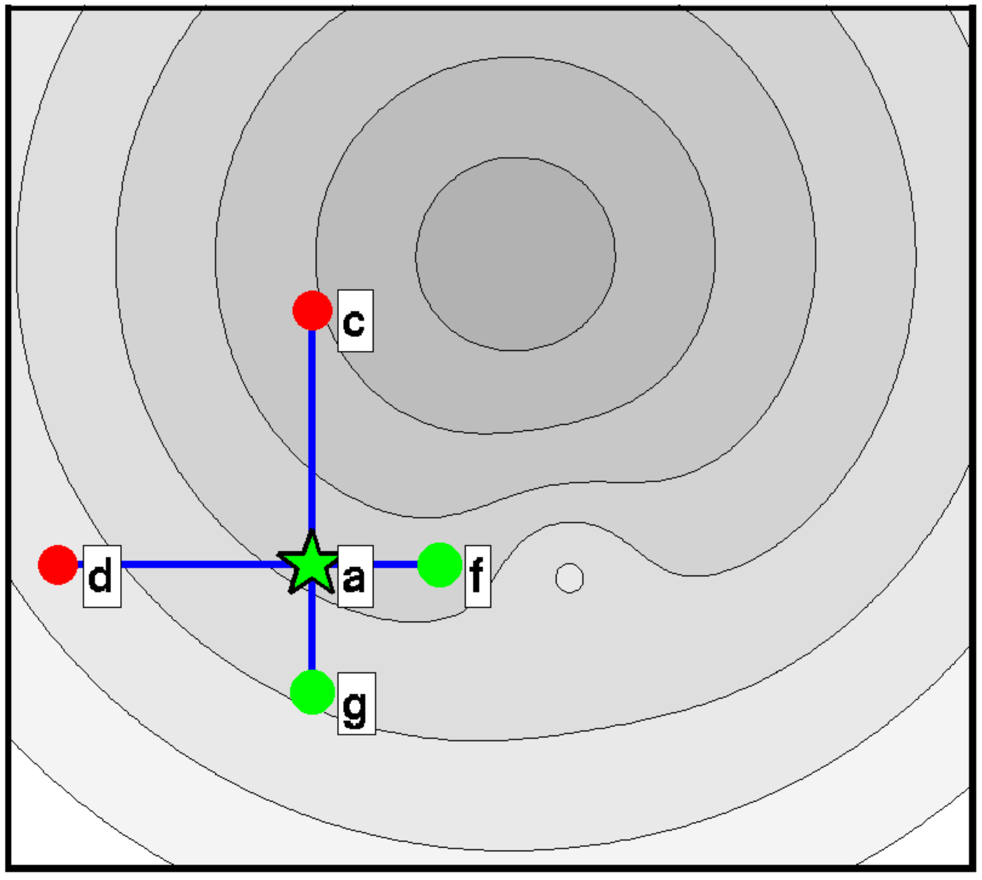
best: **a**

pending: **f g c d**

evaluated:

pruned:

Unconstrained optimization demo



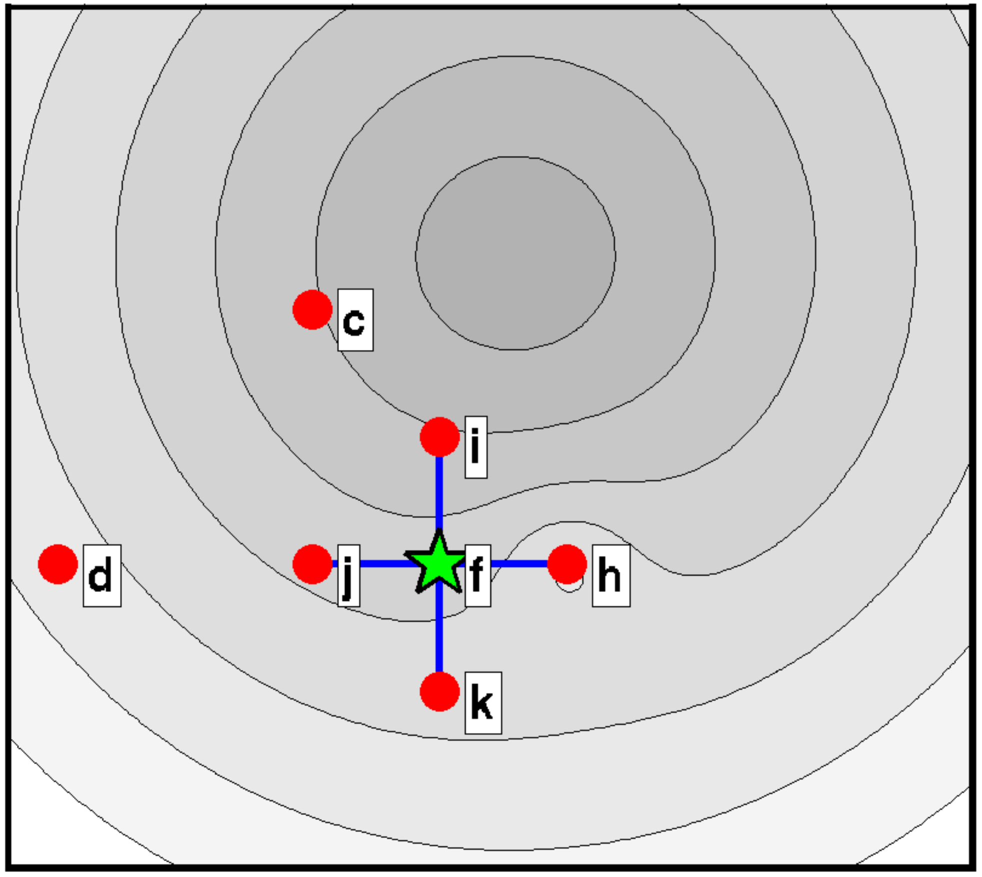
best: **a**

pending: **c d**

evaluated: **f g**

pruned:

Unconstrained optimization demo



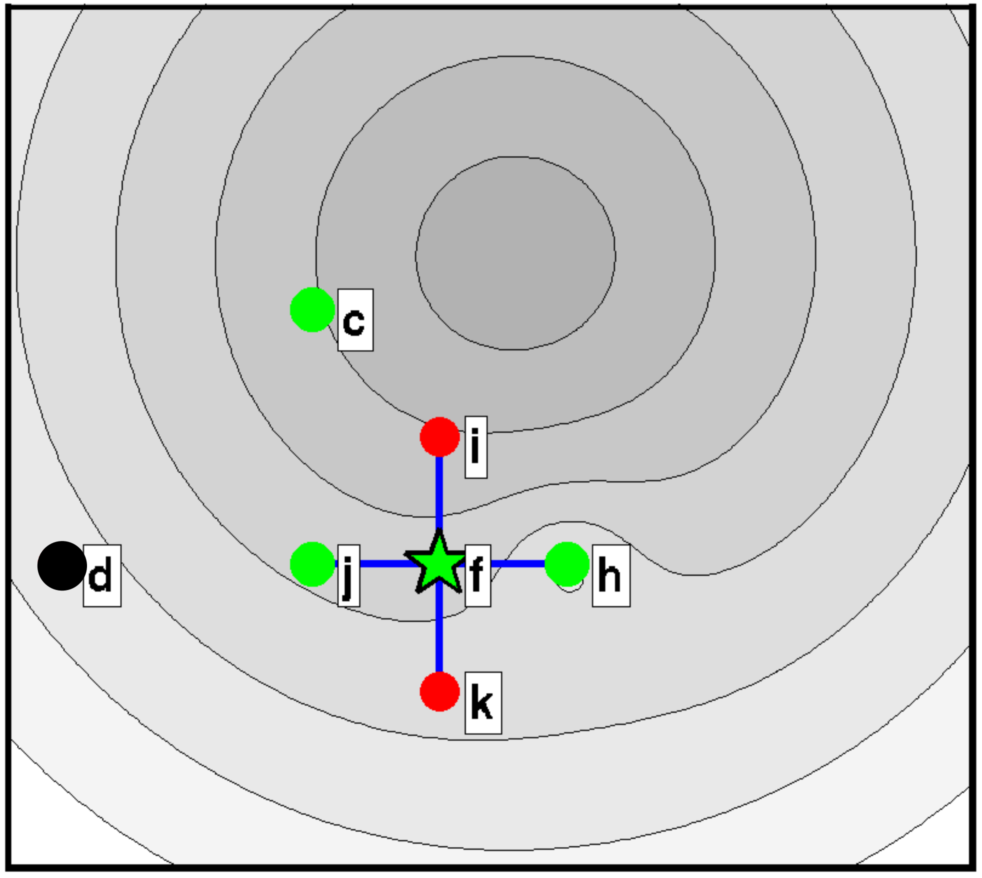
best: **f**

pending: **h i j k c d**

evaluated:

pruned:

Unconstrained optimization demo



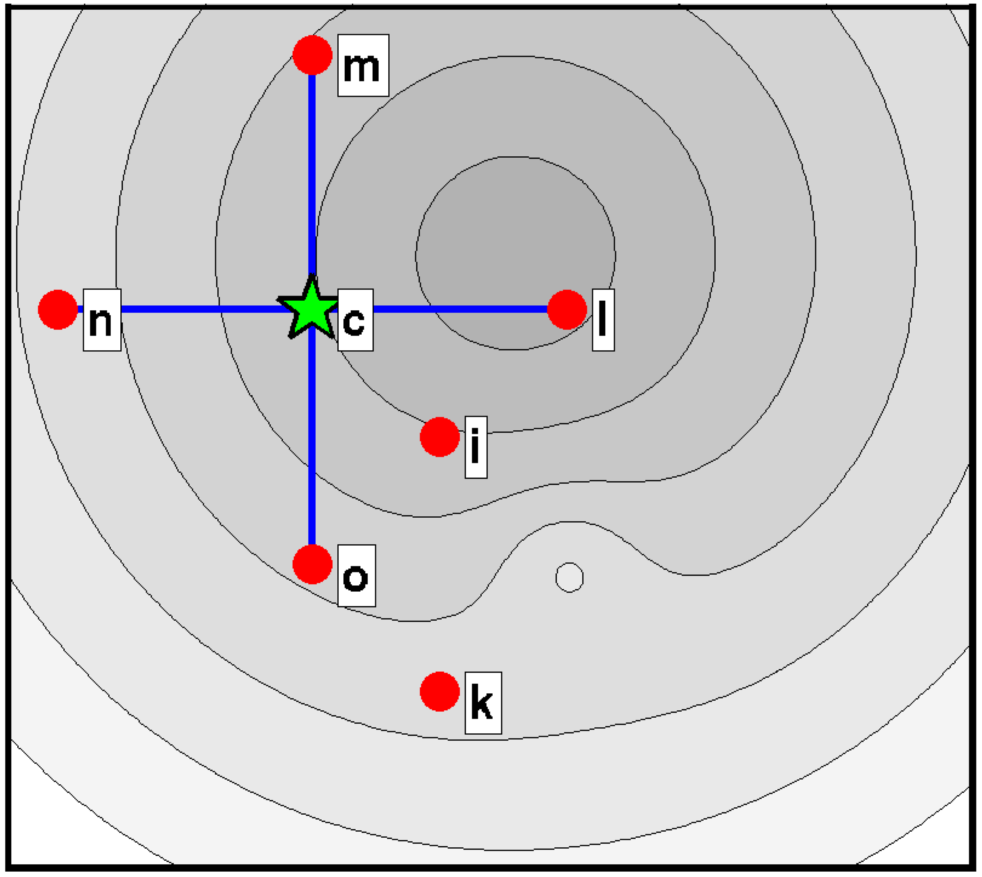
best: **f**

pending: **i k**

evaluated: **c j h**

pruned: **d**

Unconstrained optimization demo



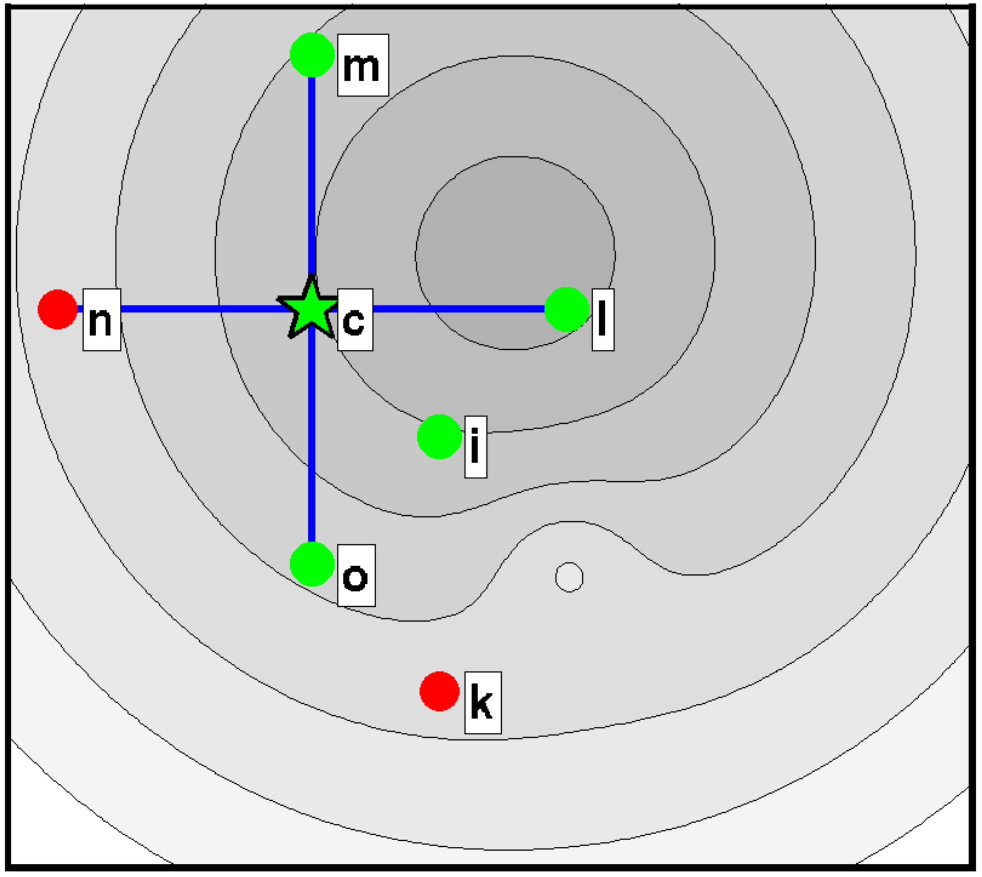
best: **c**

pending: **l m n o i k**

evaluated:

pruned:

Unconstrained optimization demo



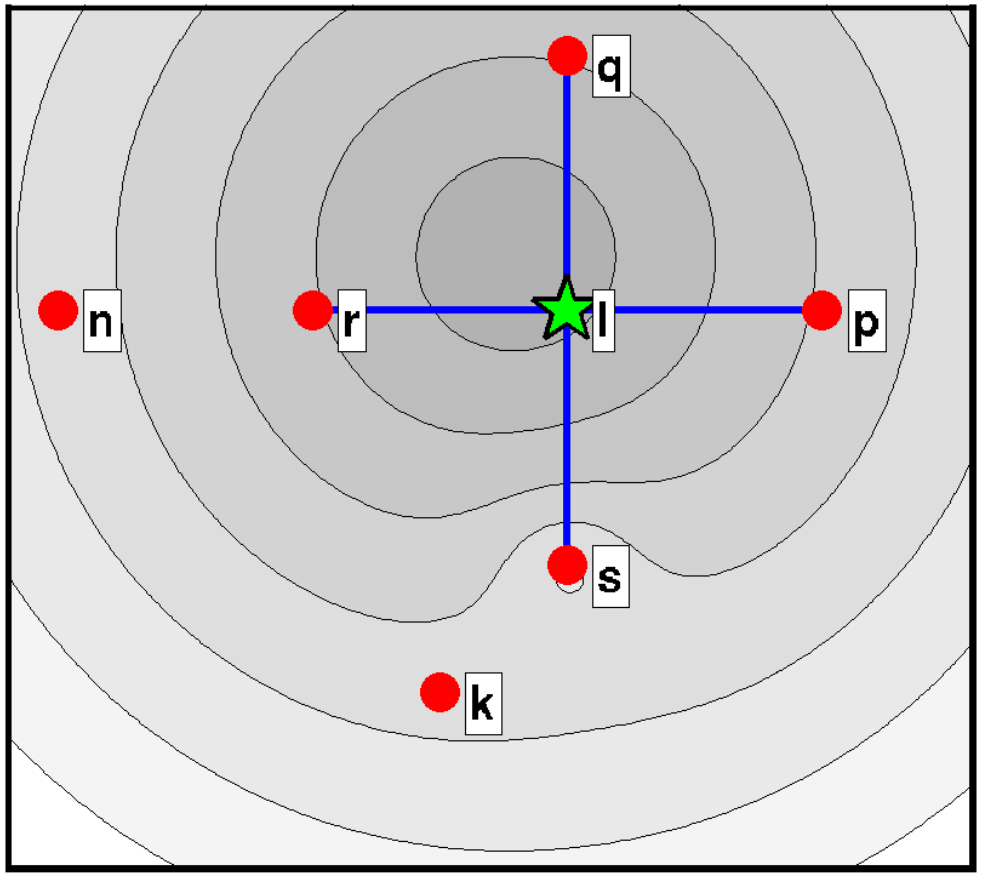
best: **c**

pending: **n k**

evaluated: **l m o i**

pruned:

Unconstrained optimization demo



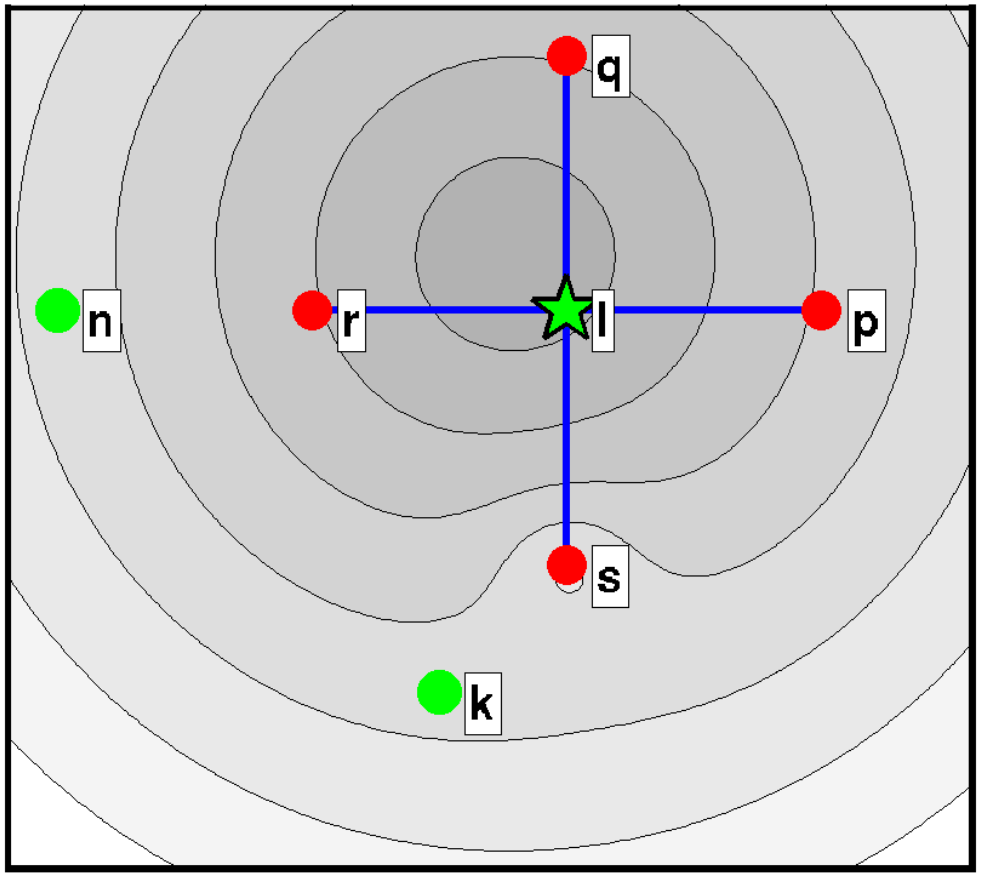
best: **l**

pending: **p q r s n k**

evaluated:

pruned:

Unconstrained optimization demo



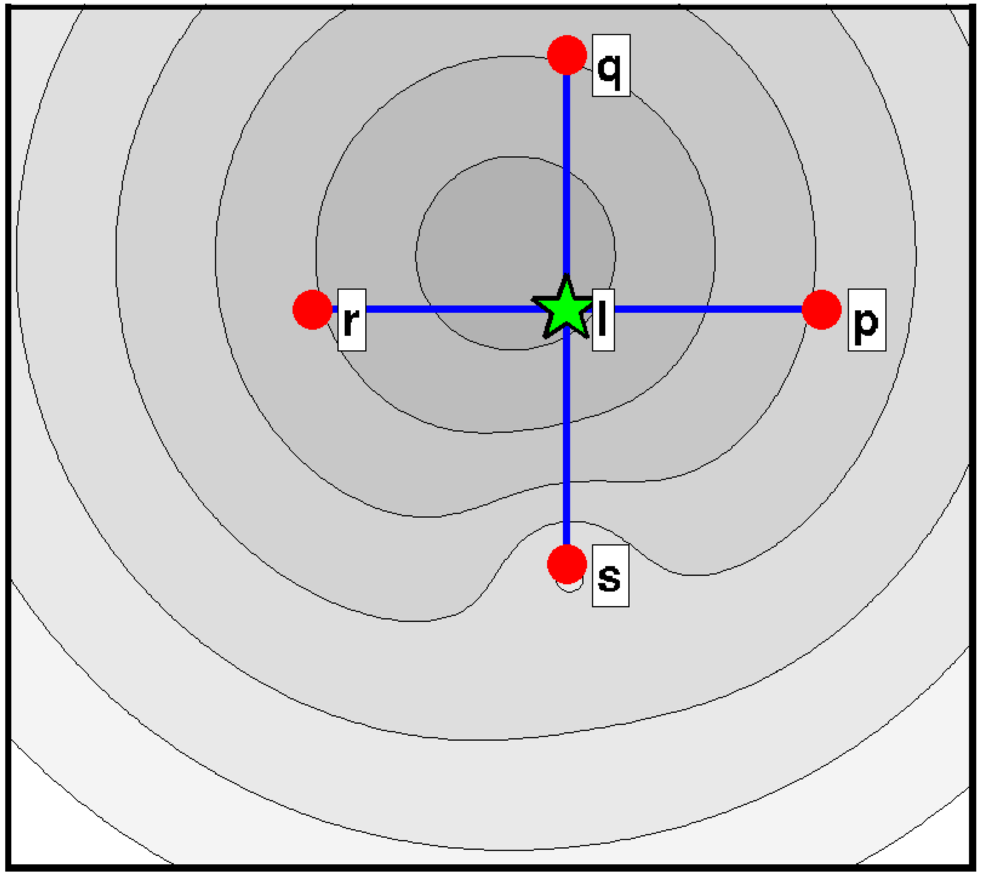
best: **l**

pending: **p q r s**

evaluated: **n k**

pruned:

Unconstrained optimization demo



best: **l**

pending: **p q r s**

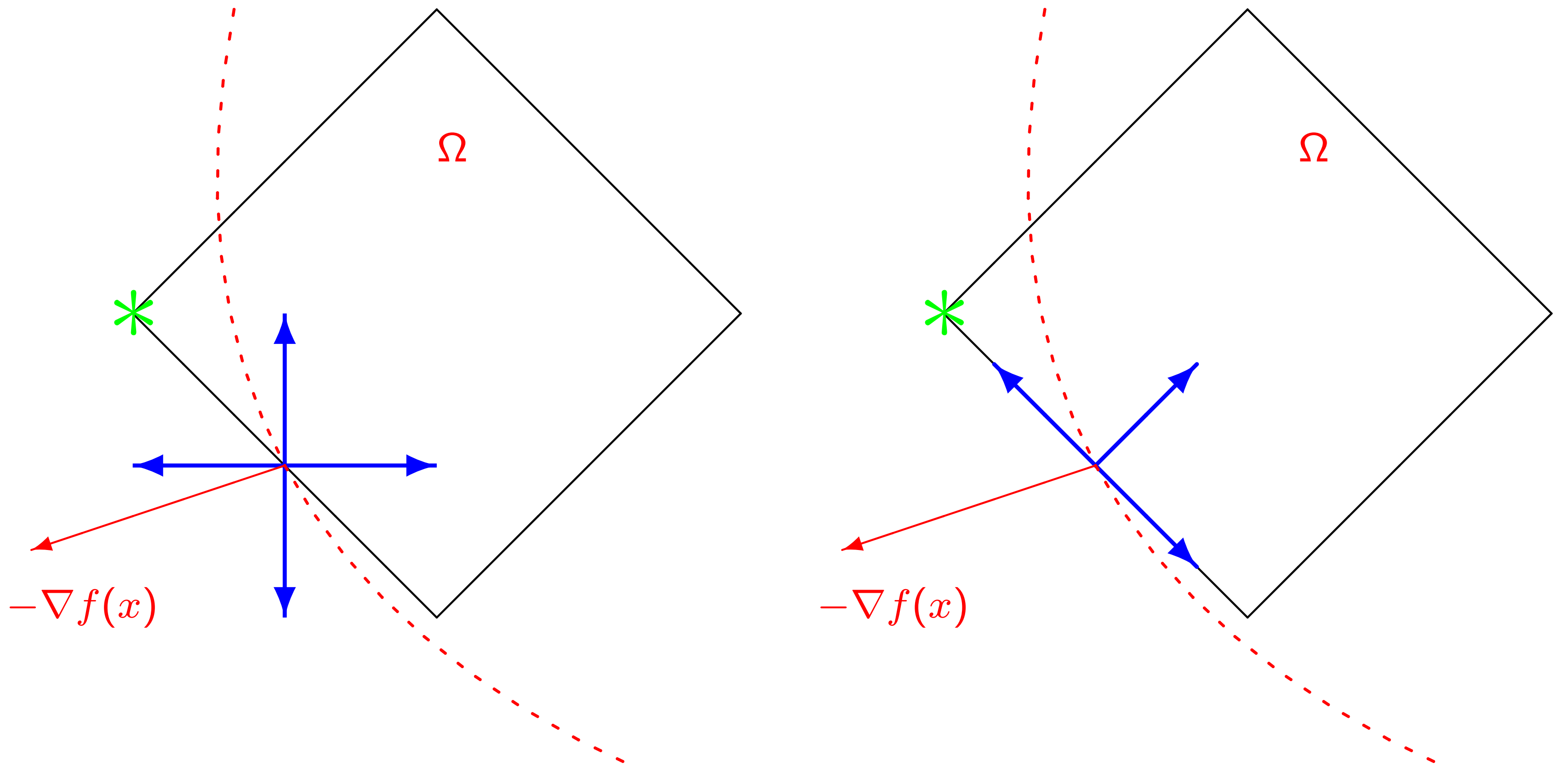
evaluated:

pruned:

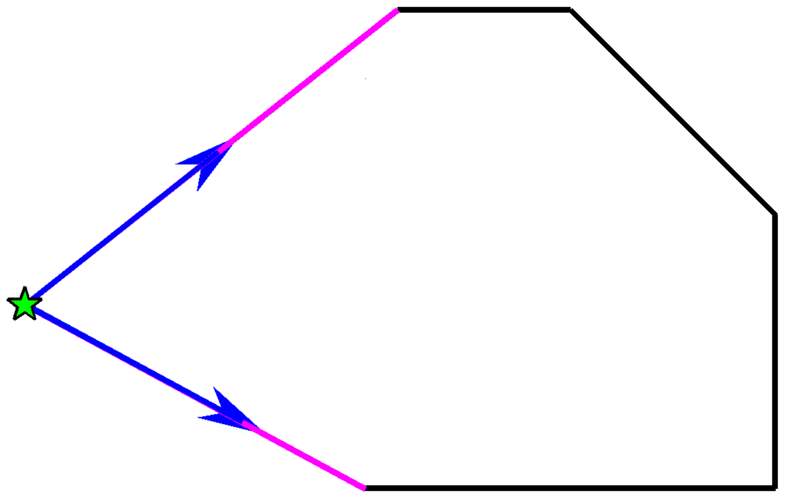
Handling linear constraints:

Same algorithm, different directions

Computing conforming search directions

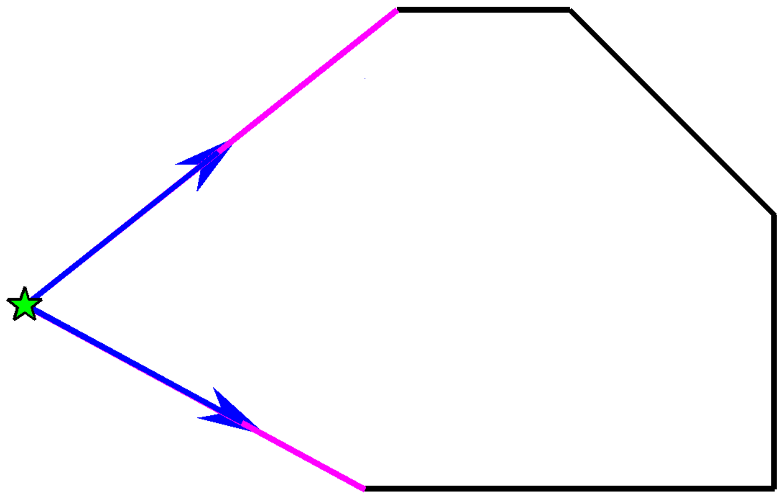


Locally conforming directions

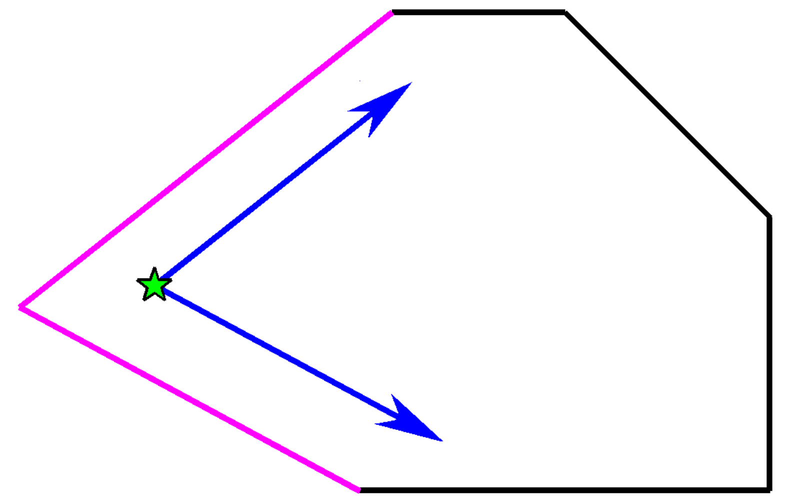


We want the ability to move parallel to active constraints

Locally conforming directions



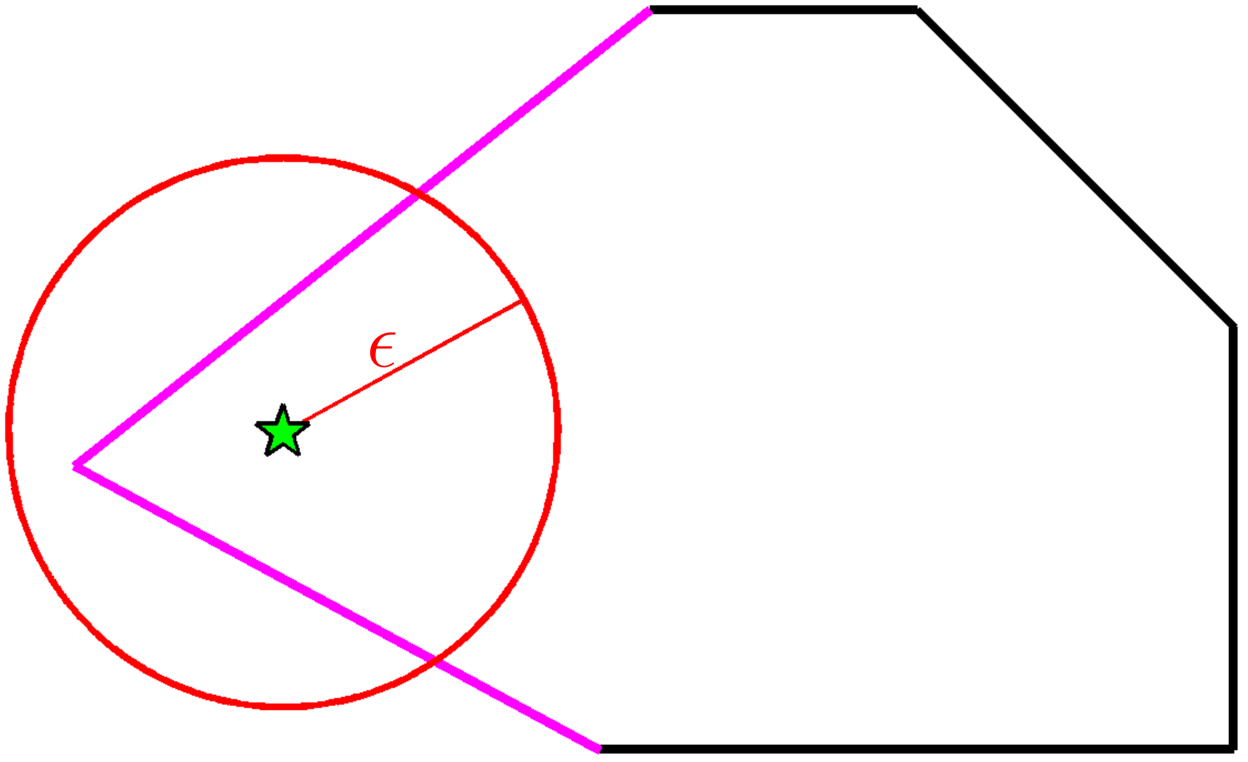
We want the ability to move parallel to active constraints



We also want the ability to move parallel to “nearby” constraints

ϵ-active constraints

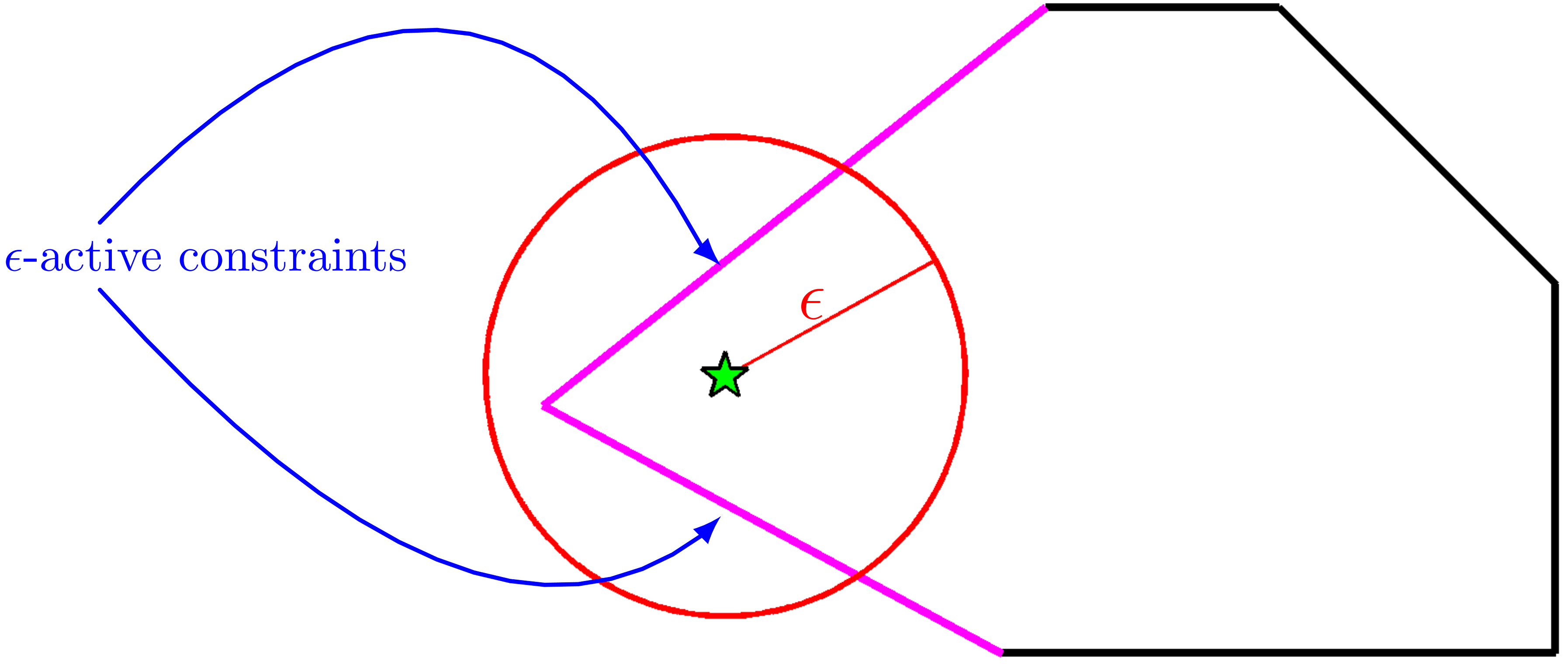
We place a ball of radius ϵ about current best point.



Constraints passing through this ϵ-ball are considered ϵ-active constraints.

ϵ-active constraints

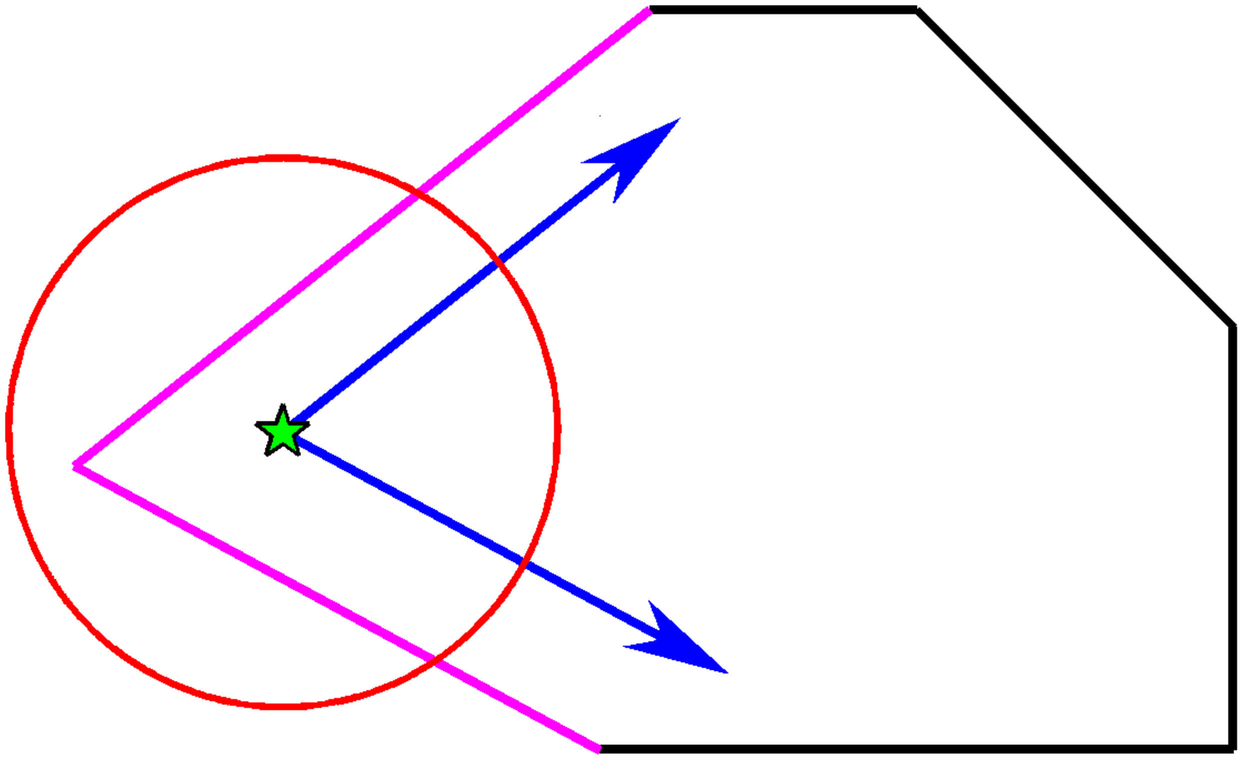
We place a ball of radius ϵ about current best point.



Constraints passing through this ϵ-ball are considered ϵ-active constraints.

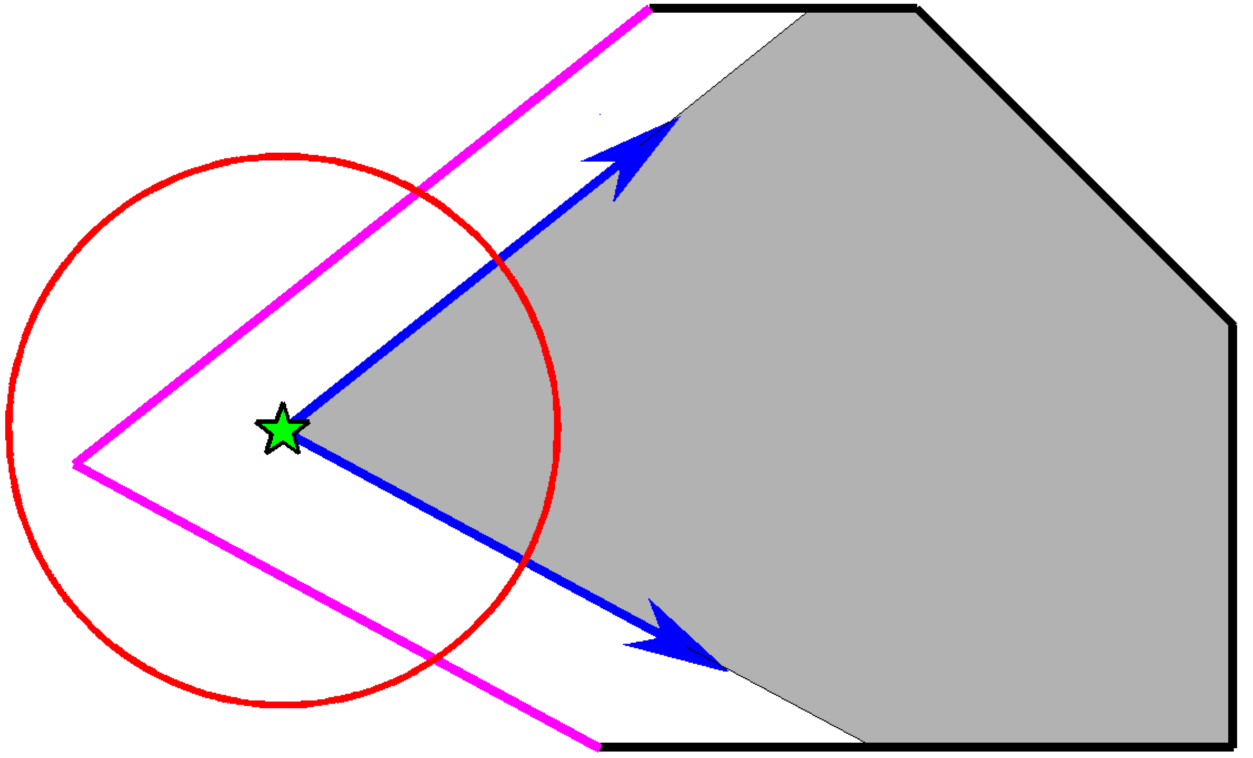
Conforming directions

We then compute corresponding conforming search directions



ϵ-tangent cone

The positive-span of conforming directions forms an ϵ-tangent cone



Summarizing

Punch-line: *generating directions in this manner ensures that we can always travel a distance of at least ϵ along each search direction and remain feasible.*

Thus it makes sense to set ϵ equal to the current step size:

$$\epsilon = \Delta.$$

In asynchronous mode we have multiple step size:

$$\Delta^{(i)},\ i = 1, ..., p.$$

Thus we must work with multiple tangent cones.

Normal and tangent cones definitions

- Lewis & Torczon (2000) define the ϵ-normal cone to be the cone generated by the outward pointing normals of the linear constraints within a distance ϵ of x:

$$\mathcal{N}(x,\epsilon) = \text{positive span}\left\{ a_i \in A : \frac{|a_i^T x - b_i|}{\|a_i\|} \leq \epsilon \right\}$$

- Define the ϵ-tangent cone, $\mathcal{T}(x,\epsilon)$, to be the polar of the normal cone:

$$\mathcal{T}(x,\epsilon) \triangleq \mathcal{N}(x,\epsilon)^\circ$$

Finding generators for $\mathcal{N}(x,\epsilon)$ easy

Finding generators for $\mathcal{T}(x,\epsilon)$ not so easy

Linearly constrained optimization

Conforming directions derived from tangent cones of nearby constraints:

- nondegenerate case: basic linear algebra sufficient, generators computed with LAPACK.
- degenerate case: basic linear algebra insufficient, generators formed with C-library cddlib:
  - Double description method of Motzkin et al. written by Komei Fukuda.

Synchronous framework for linear constraints

Choose $\epsilon_{\mathsf{max}} > \Delta_{\mathsf{tol}}$.

- Form conforming search directions for ϵ-active constraints, $\epsilon = \mathsf{min}(\Delta, \epsilon_{\mathsf{max}})$.
- Trial point generation:

$$\mathcal{X} = \{x + \tilde{\Delta} d^{(i)} : d^{(i)} \in \text{search pattern}\}, \ \tilde{\Delta} \in [0, \Delta]$$

  and send to evaluation queue.
- Trial point evaluation: Collect evaluated points $\mathcal{Y}$ $(= \mathcal{X})$.
- Decision: If a point $y \in \mathcal{Y}$ is determined to be "better than" x, iteration is considered successful.
- Successful: $x \leftarrow y$
- Unsuccessful: $\Delta \leftarrow .5\Delta$
- Stop: if $\Delta < \Delta_{\mathsf{tol}}$

Note: Theoretically, we need $\epsilon_{\mathsf{max}} > \Delta_{\mathsf{tol}}$ to ensure convergence. Choosing ϵ_{max} to large can limit step size however.

Asynchronous tricky

- Multiple step sizes implies multiple tangent cones may be relevant.
- In the synchronous case, only one tangent cone per iteration has theoretical importance.
  - Thus, merely swap out cone generators whenever the tangent cone changes.
- In the asynchronous case, extra bookkeeping is needed to keep track of when we can swap and when we must append search directions.
- Ultimately, we must ensure that at each iteration, the search directions contain generators for

$$\bigcup_{\{i:\, \Delta^{(i)} \leq \epsilon_{\mathsf{max}}\}} \mathcal{T}(x, \Delta^{(i)}) \cup \mathcal{T}(x, \epsilon_{\mathsf{max}})$$

Asynchronous framework for linear constraints

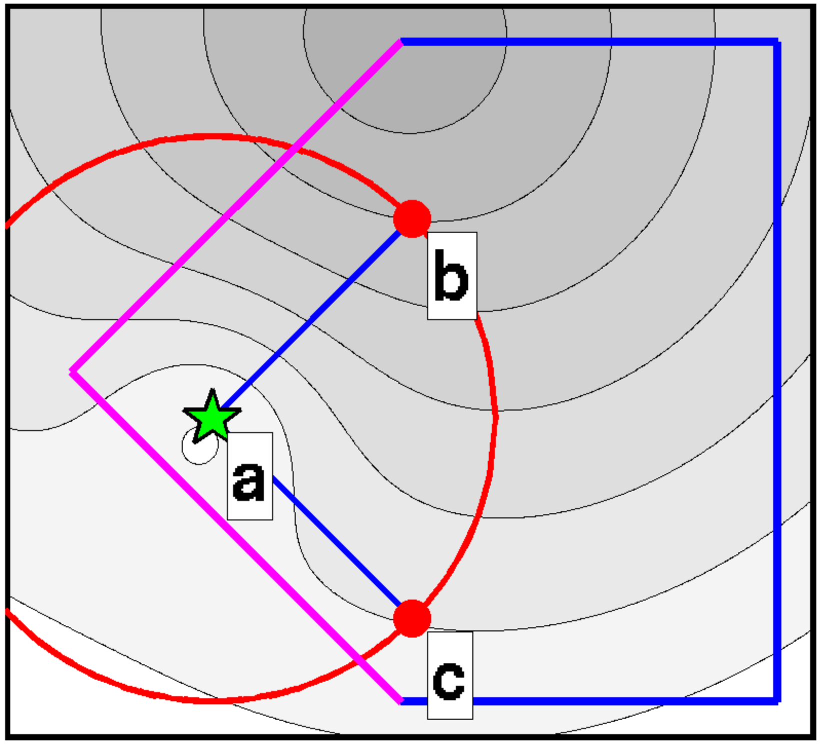
Choose $\epsilon_{\mathsf{max}} > \Delta_{\mathsf{tol}}$.

- Trial point generation: $\mathcal{X} = \{x + \tilde{\Delta}^{(i)} d^{(i)} : d^{(i)} \in \text{search pattern and inactive}\}$
- Trial point evaluation: Collect a nonempty set of evaluated point $\mathcal{Y}$
- Decision: If a point $y \in \mathcal{Y}$ is determined to be "better than" x, iteration is considered successful
- Successful: $x \leftarrow y$, reset $\Delta^{(i)} = \hat{\Delta} = \mathsf{max}(\mathsf{step(y)}, \Delta_{\mathsf{min}})$. Set $\epsilon = \mathsf{min}(\hat{\Delta}, \epsilon_{\mathsf{max}})$. New set of search direction $= \mathcal{T}(x, \epsilon)$. Note: One step-size $\Rightarrow$ one relevant tangent cone
- Unsuccessful: $\Delta^{(i)} \leftarrow .5\Delta^{(i)}$ for all direction indices corresponding to points in $\mathcal{Y}$. Append search directions if $\mathsf{min}(\epsilon_{\mathsf{max}}, \mathsf{min}_i\, \Delta^{(i)})$ has decreased to ensure search directions contain generators for

$$\bigcup_{\{i:\, \Delta^{(i)} \leq \epsilon_{\mathsf{max}}\}} \mathcal{T}(x, \Delta^{(i)}) \cup \mathcal{T}(x, \epsilon_{\mathsf{max}})$$

- Stop: if $\Delta^{(i)} \leq \Delta_{\mathsf{tol}}$ for all i

Linear constrained optimization demo

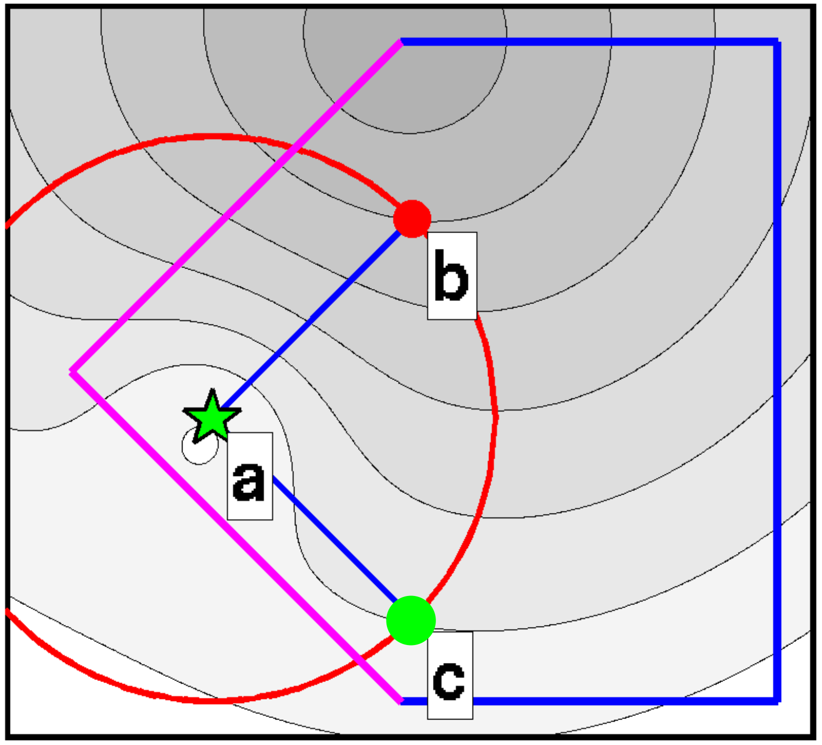


best: **a**

pending: **b c**

evaluated:

Linear constrained optimization demo

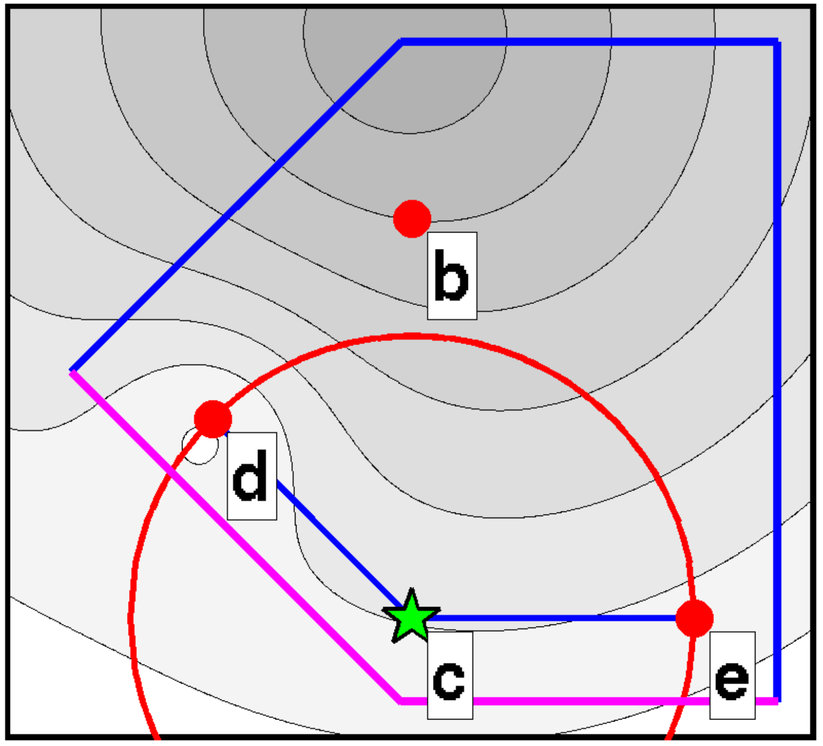


best: **a**

pending: **b**

evaluated: **c**

Linear constrained optimization demo

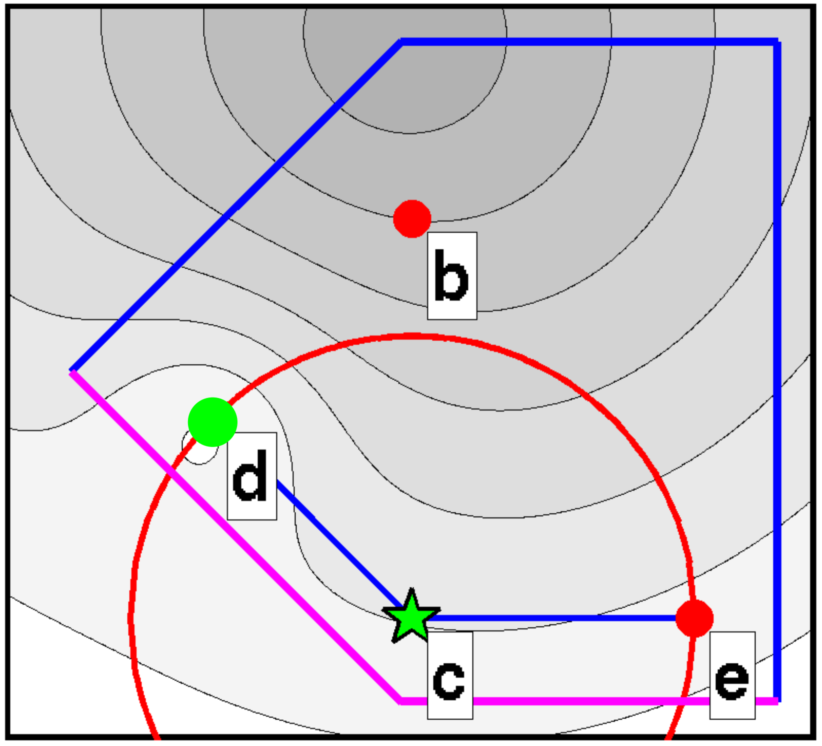


best: **c**

pending: **d e b**

evaluated:

Linear constrained optimization demo

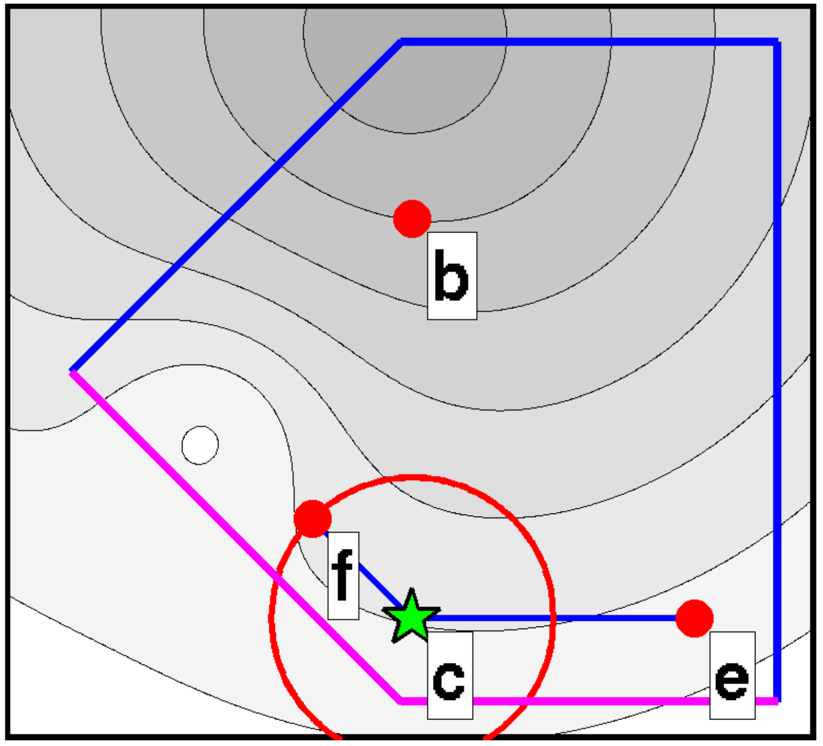


best: **c**

pending: **e b**

evaluated: d

Linear constrained optimization demo

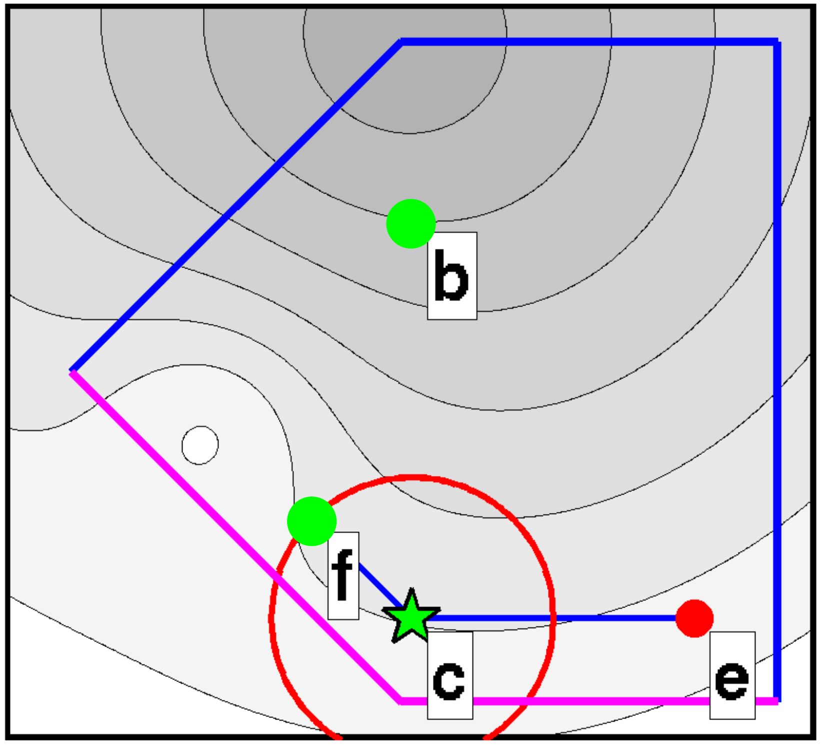


best: **c**

pending: **f e b**

evaluated:

Linear constrained optimization demo

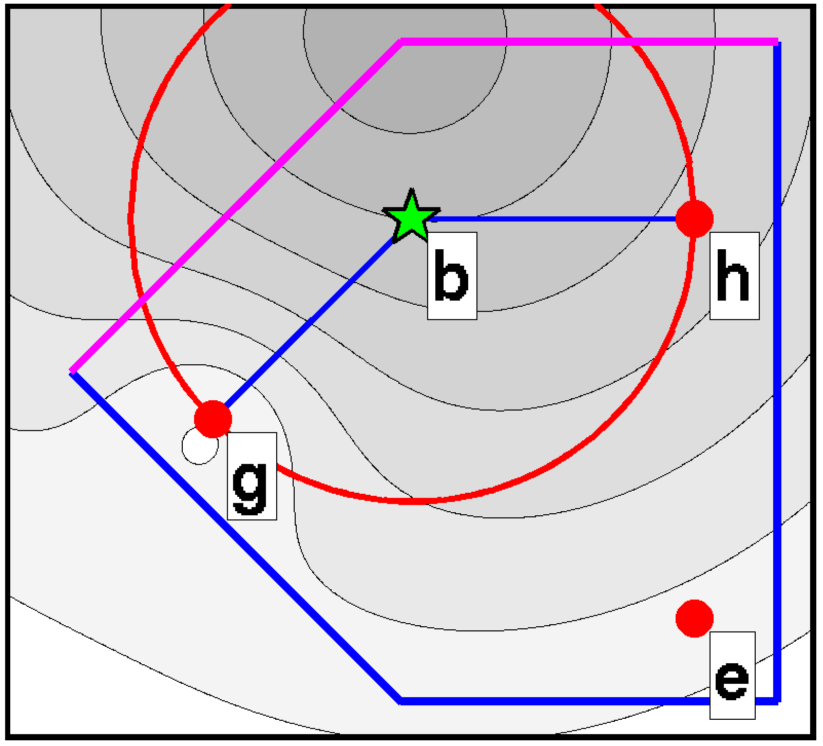


best: **c**

pending: **e**

evaluated: **f b**

Linear constrained optimization demo

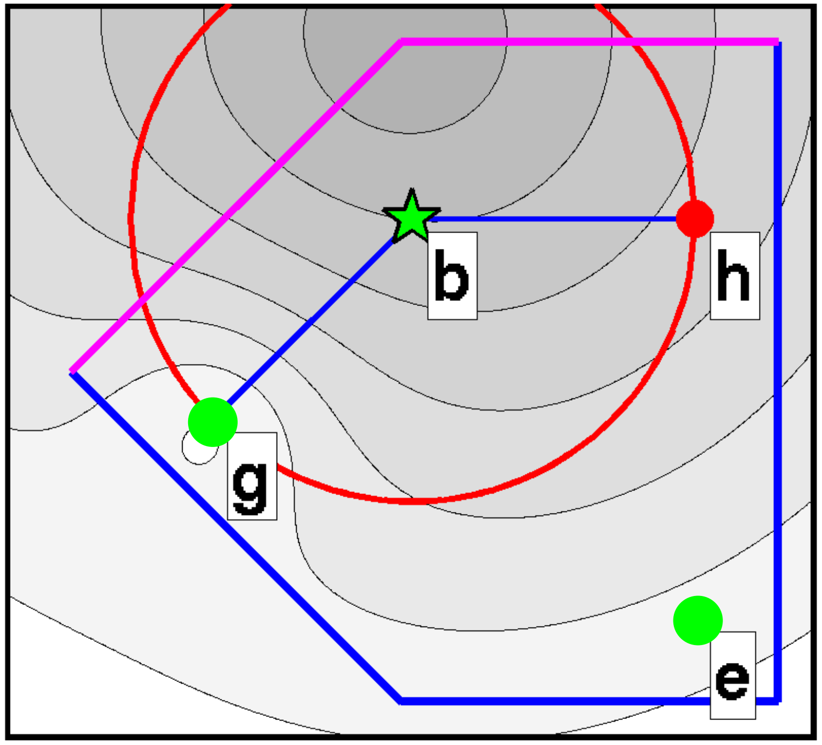


best: **b**

pending: **g h e**

evaluated:

Linear constrained optimization demo

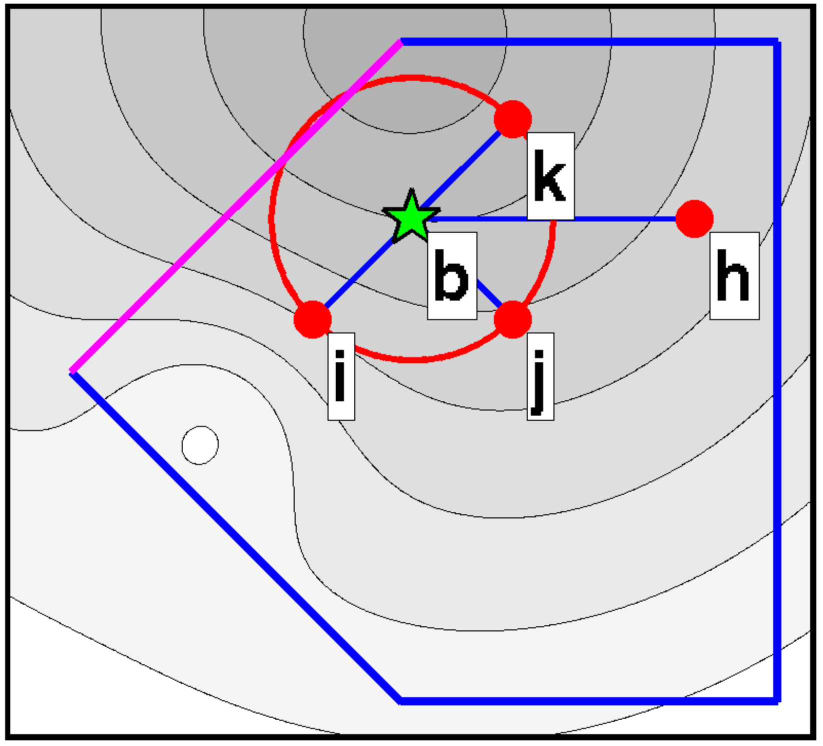


best: **b**

pending: **h**

evaluated: **g e**

Linear constrained optimization demo

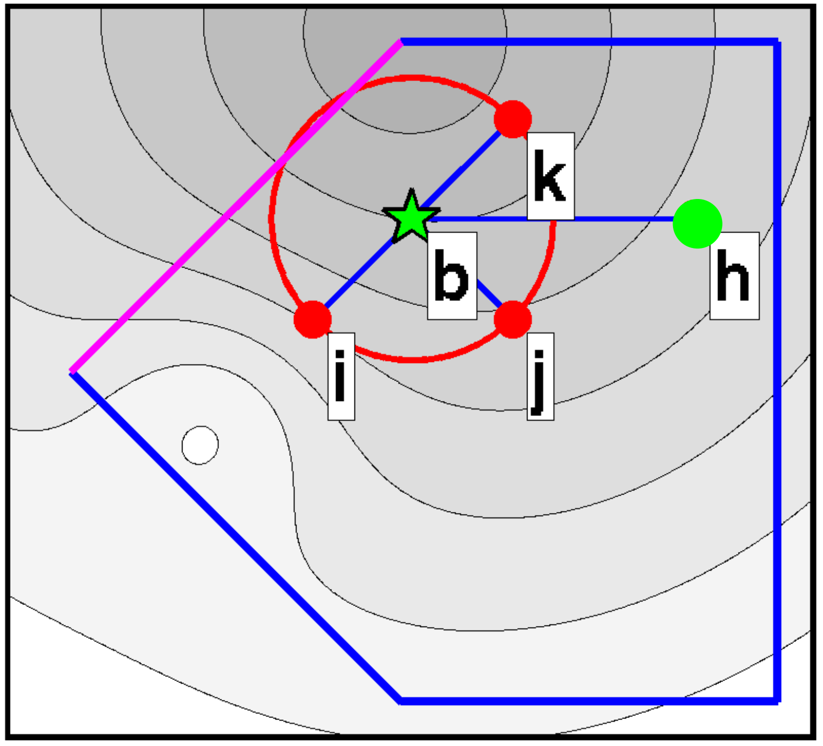


best: **b**

pending: **i j k h**

evaluated:

Linear constrained optimization demo

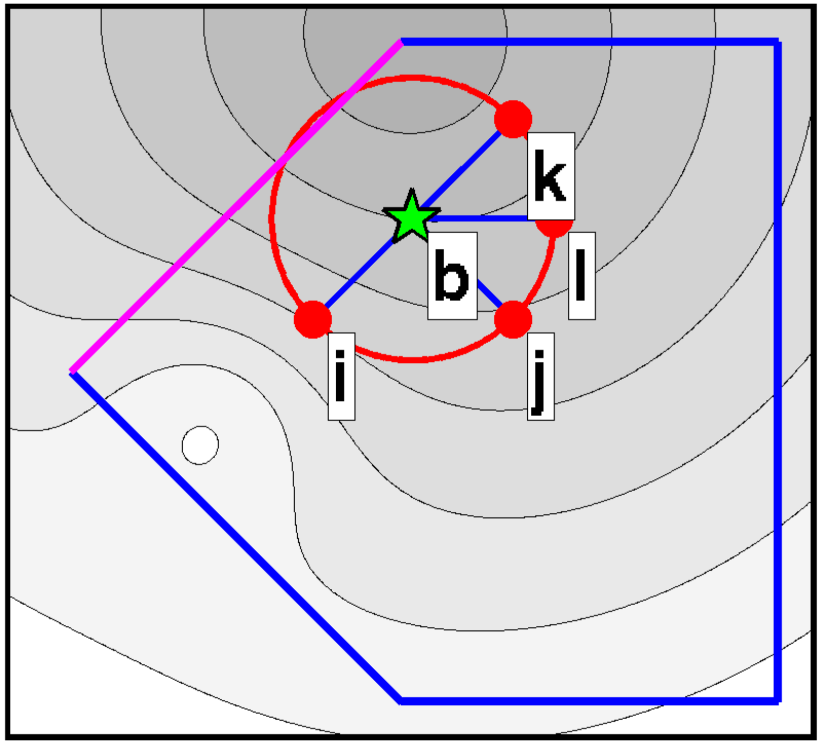


best: **b**

pending: **i j k**

evaluated: **h**

Linear constrained optimization demo



best: **b**

pending: **l i j k**

evaluated:

Asynchronous convergence theory

A useful measure of optimality

$$\chi(x) = \max_{\substack{x+\omega \in \Omega \\ \|w\| \leq 1}} -\nabla f(x)^T w.$$

Can show that $\chi(x) \geq 0$, $\chi(x)$ is continuous, and $\chi(x) = 0$ iff x is first-order optimal

Conn, Gould, Sartenaer, and Toint. (1996)

(a) Under assumptions always satisfied before APPSPACK terminates, we can show

$$\|P_{\mathcal{T}(x,\hat{\Delta})}(-\nabla f(x))\| \leq C_1 \hat{\Delta}$$

$$\chi(x) \leq C_2 \hat{\Delta}$$

where $\hat{\Delta}$ equals the current maximum step size

(b) $\liminf \hat{\Delta} = 0$

(a) and **(b)** together imply global convergence to a first-order optimal point

$P_{\mathcal{T}(x,\hat{\Delta})}(-\nabla f(x))$ denotes projection of $-\nabla f(x)$ onto local tangent cone $\mathcal{T}(x, \hat{\Delta})$

C_1 and C_2 depend on properties of f and A

APPSPACK numerical results for general linear constraints

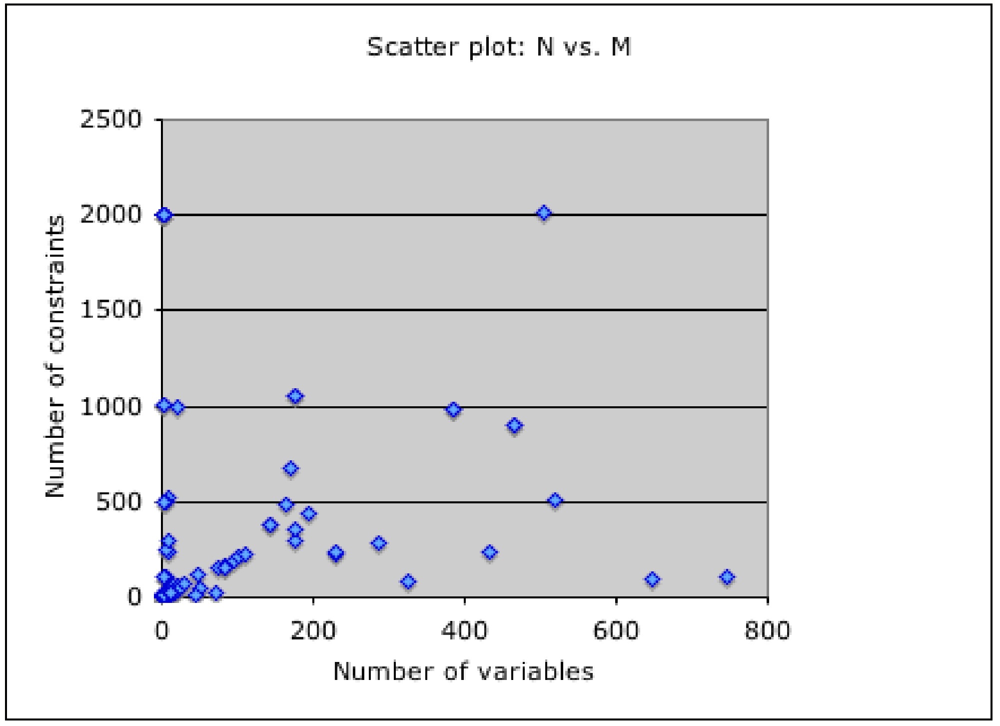
Details:

- Tested on linearly constrained CUTEr (Constrained and Unconstrained Testing Environment, revisited) (non-trivial) problems with $n \leq 1000$ variables
- All problems tested asynchronously in parallel on Sandia's Institutional Computing Cluster (ICC)
  - 20 proc for $n \leq 10$,
  - 40 proc for $10 < n \leq 100$
  - 60 proc for $100 < n \leq 1000$

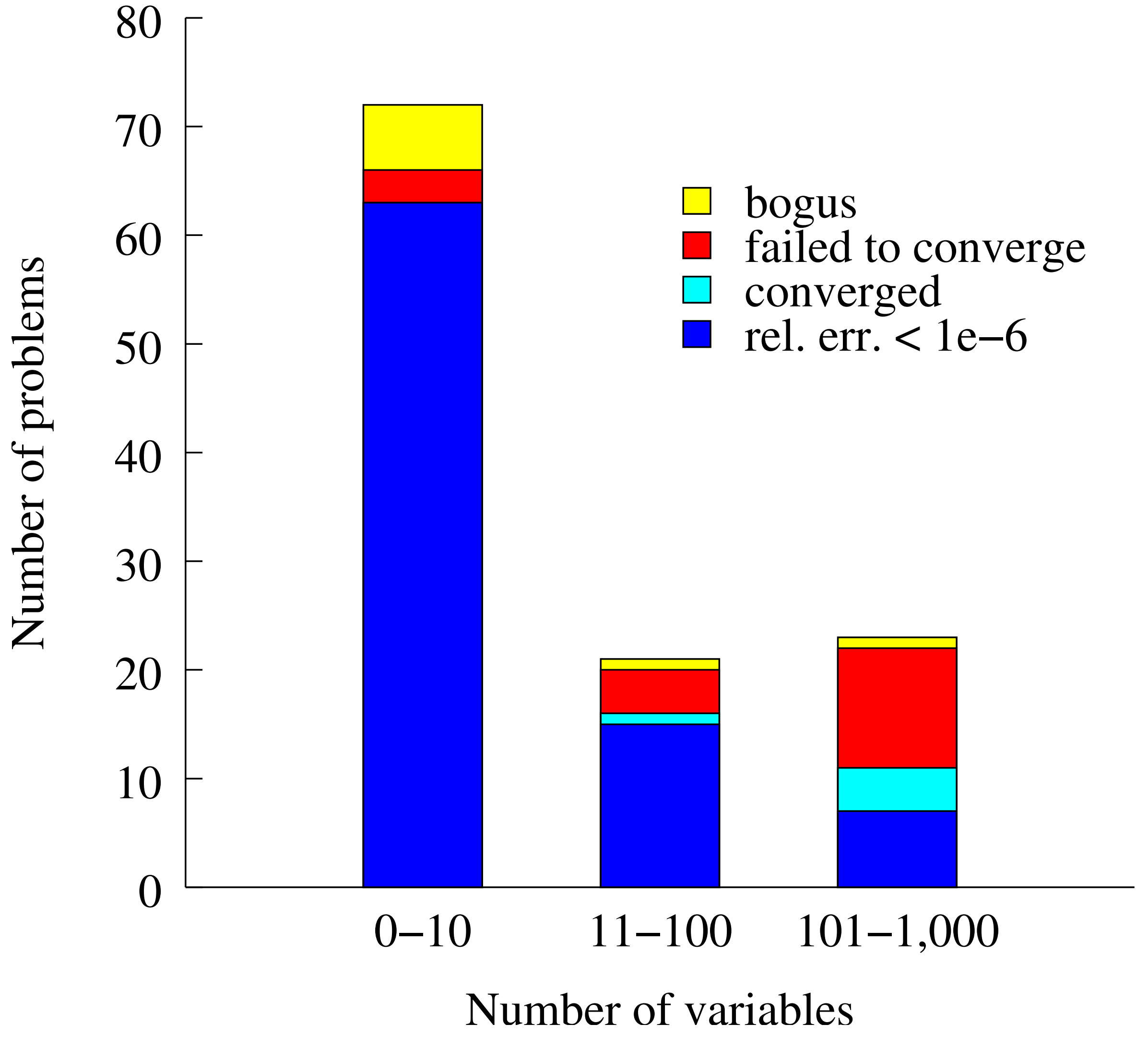
Motivation:

- Stress test APPSPACK's new linear constraint capabilities
  - CUTEr problem known to be difficult even for derivative-based methods
- Verify new asynchronous theory numerically
  - At risk of doing a large number of function evaluations, set stopping tolerance unusually high to see how well we could do

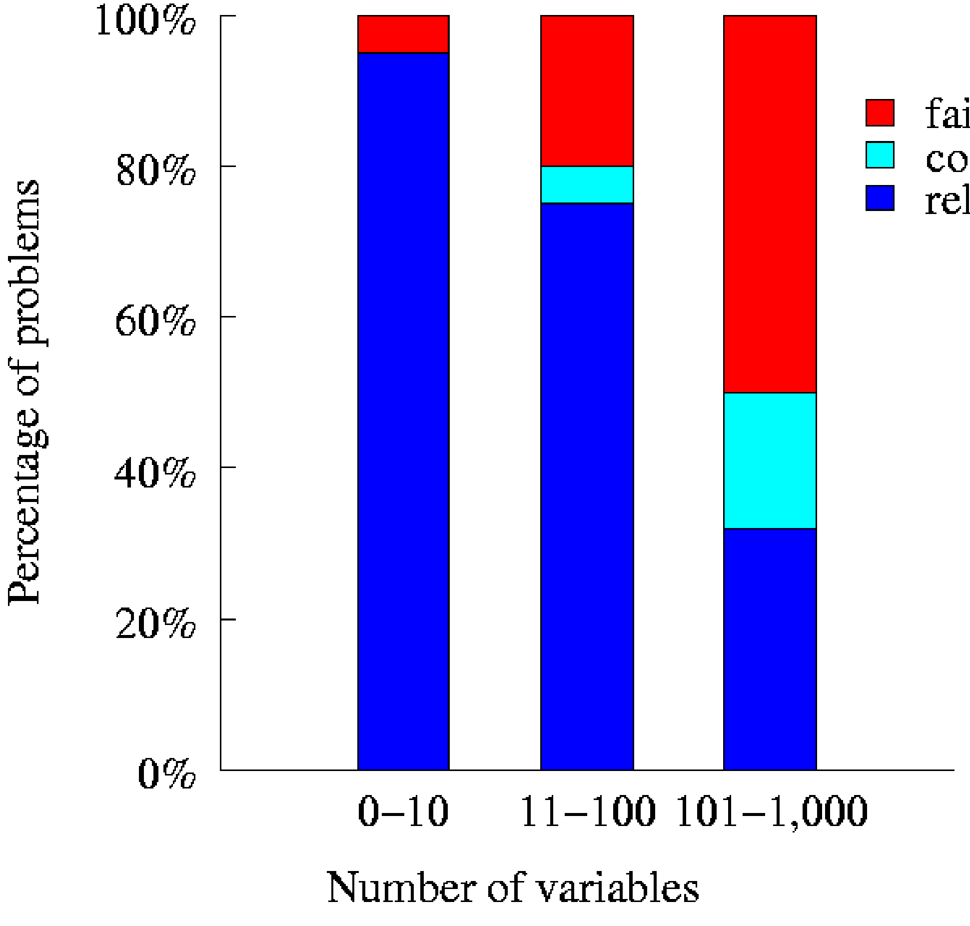
Numerical results: problem sizes



Numerical results: accuracy



Numerical results: accuracy



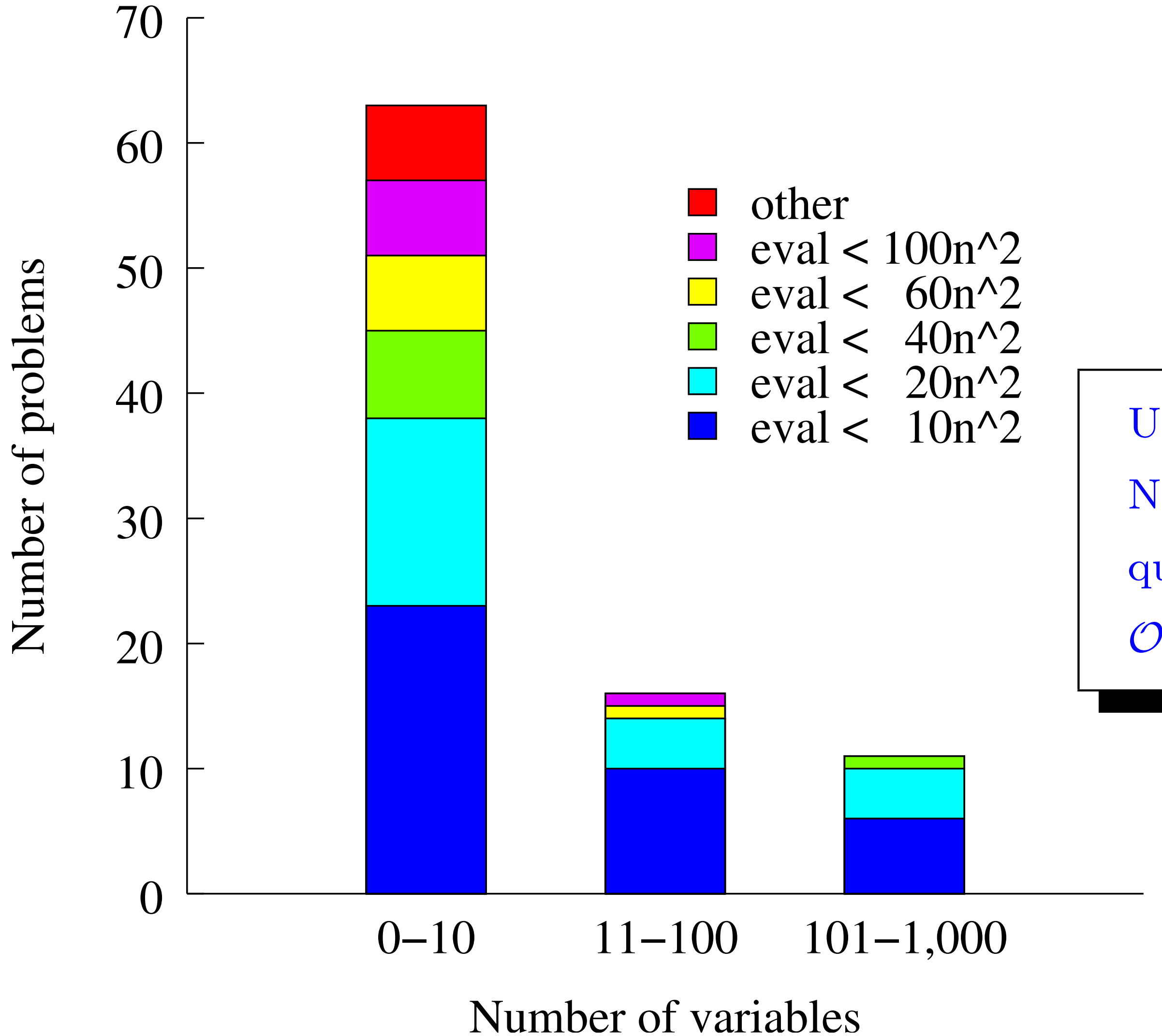
Largest problem solved:

505 variables,

1010 simple bounds, and

1008 constraints

Numerical results: function evaluations

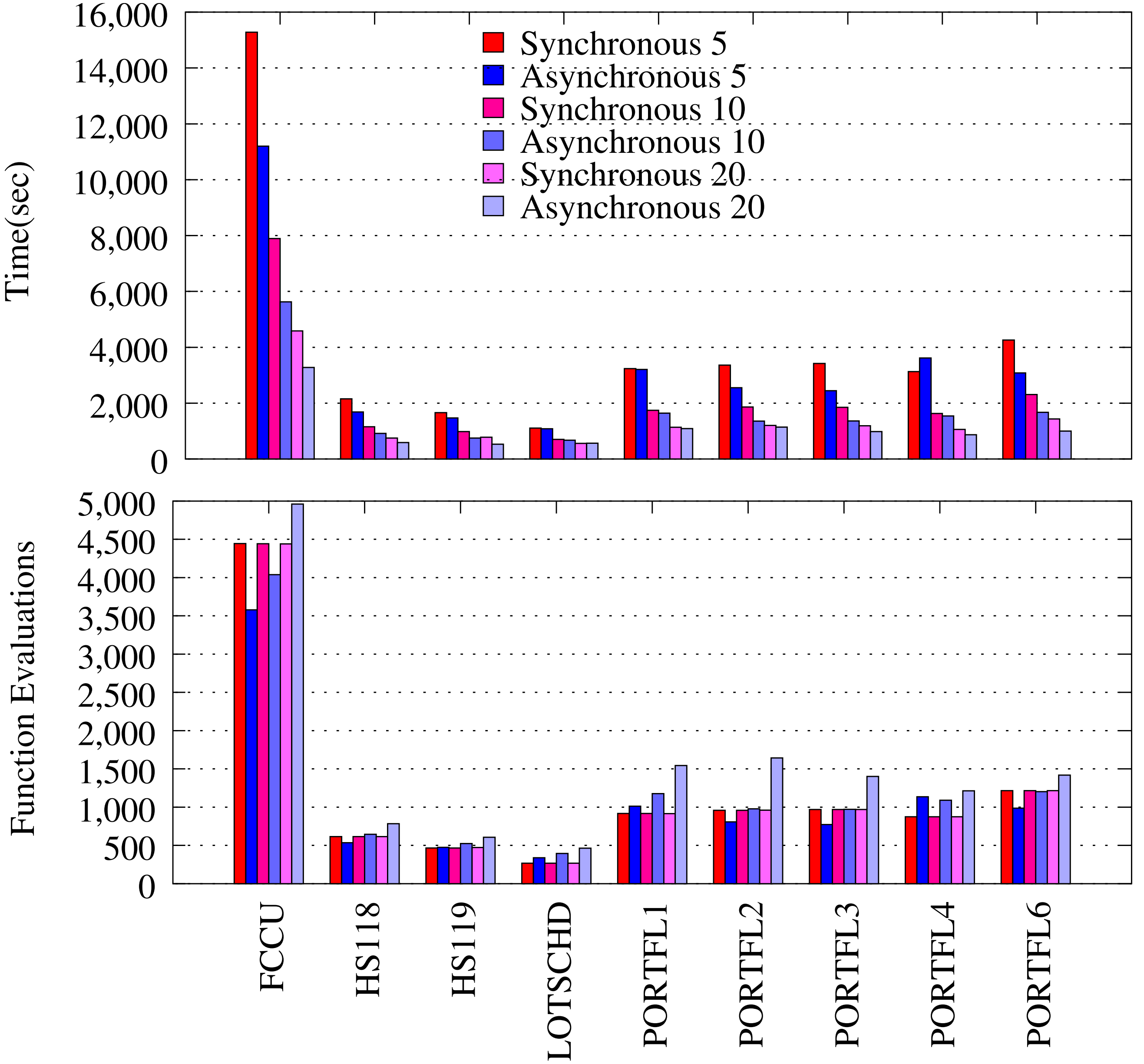


Using finite-difference Newton to minimize a convex quadratic one would expect $\mathcal{O}(n^2)$ evaluations.

Sync vs. Async

9 midrange problems selected. 5-15 seconds added randomly to each evaluation.

27 comparisons made



Handling nonlinear constraints

A sequence of linearly constrained problems

The subproblem

We solve a series of linearly constrained subproblems for λ_k, μ_k fixed:

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} & \Phi_k(x) \\ \text{subject to} & Ax \leq b \end{array}$$

where

$$\Phi_k(x) \triangleq f(x) + \lambda_k^T c(x) + \frac{1}{2\mu_k} \|c(x)\|^2$$

Each subproblem is solved approximately using APPSPACK.

Key feature: Algorithm can be shown to be globally convergent to first-order optimal points without accessing/estimating derivatives.

Conclusions

Conclusions and Summary

- APPSPACK with linear constraints:
  - Globally convergent to a KKT point.
  - Works well in practice.
  - Stable version currently available for download.
  - Corresponding paper "Asynchronous parallel generating set search for linearly-constrained optimization" to be submitted to SISC.
- APPSPACK with general equality constraints:
  - Globally convergent to a KKT point.
  - Software in place; currently fine tuning and debugging.
  - Stable release by end of next month.

Can download latest stable and developmental version here (LGPL license):

http://software.sandia.gov/appspack

Future work

- Categorical variables:

$$\begin{array}{ll} \underset{x_c \in \Omega, x_d \in \mathcal{S}}{\text{minimize}} & f(x_c, x_d) \\ \text{subject to} & \Omega \subset \mathbb{R}^n \\ & \mathcal{S} = \text{red, blue, green, etc.} \end{array}$$

- Nonlinear inequality constraints solved with slacks:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & \begin{array}{ll} h(x) \le 0, & \\ c(x) = 0, & \quad Ax \le b \end{array} \end{array}$$

- Globalization of APPSPACK
- Support for oracle points

Future work

- Categorical variables:

$$\begin{array}{ll} \underset{x_c \in \Omega, x_d \in \mathcal{S}}{\text{minimize}} & f(x_c, x_d) \\ \text{subject to} & \Omega \subset \mathbb{R}^n \\ & \mathcal{S} = \text{red}, \text{blue}, \text{green}, \text{etc.} \end{array}$$

- Nonlinear inequality constraints solved with slacks:

$$\begin{array}{ll} \underset{x, z}{\text{minimize}} & f(x) \\ \text{subject to} & \begin{array}{ll} h(x) + z = 0, & z \leq 0 \\ c(x) = 0, & Ax \leq b \end{array} \end{array}$$

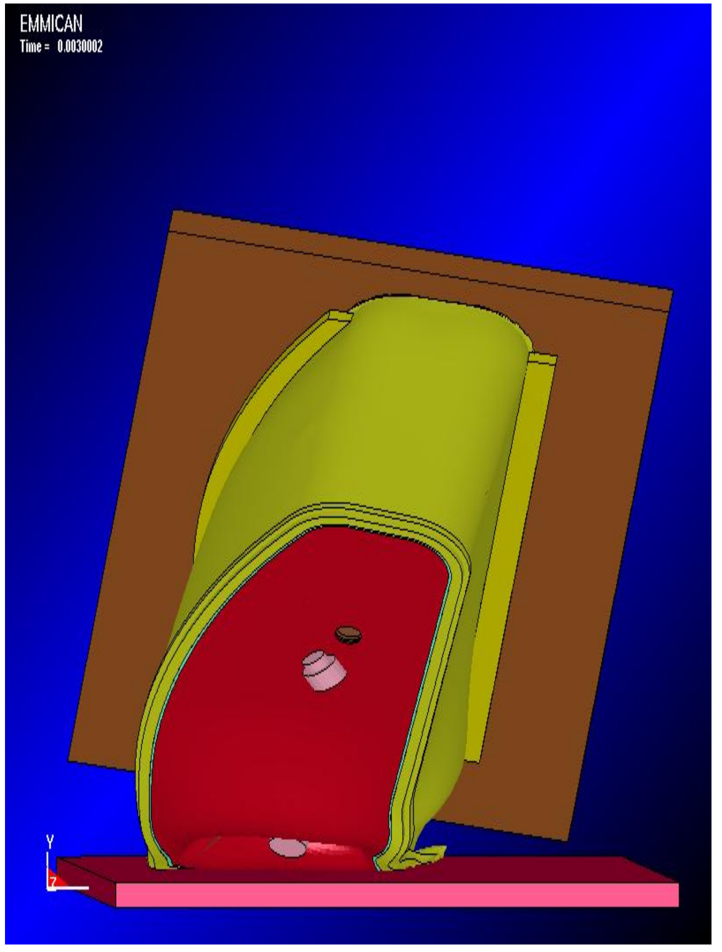
- Globalization of APPSPACK
- Support for oracle points

Why asynchronous?

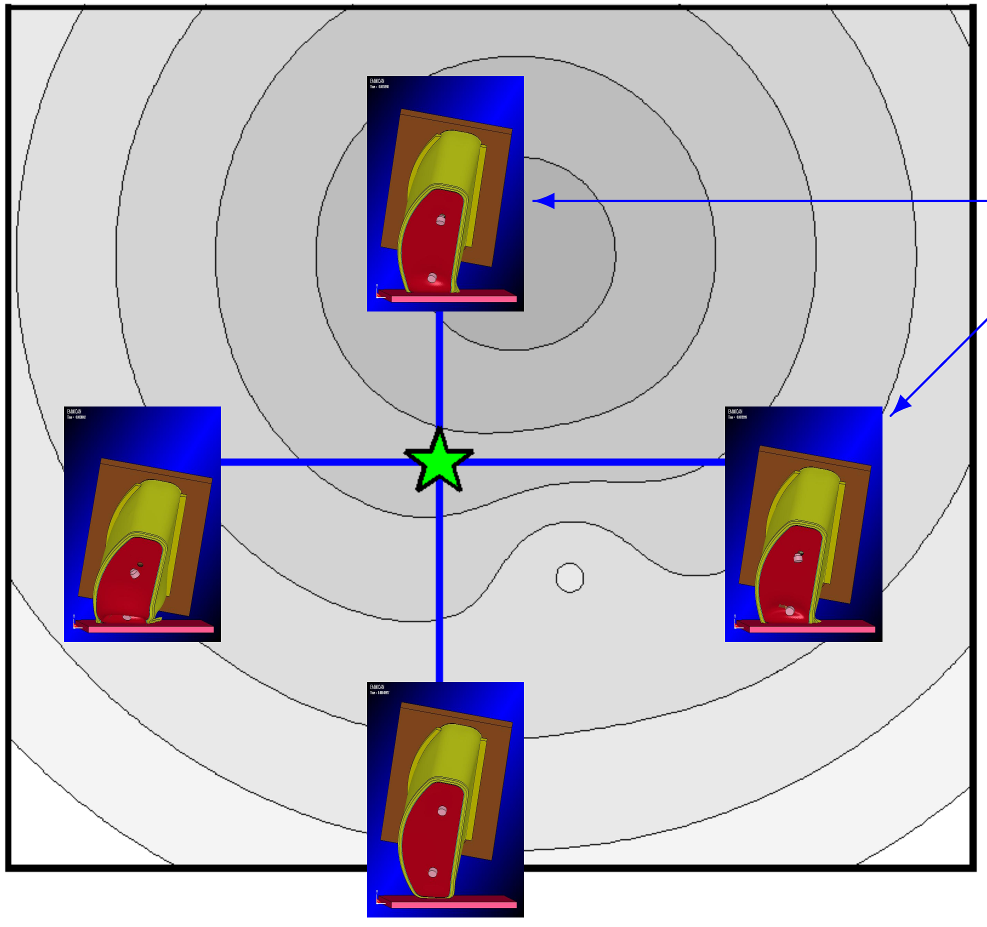
Sandia optimization problem (supporting nuclear safety studies)

Goal: *Determine if accidental drop could jeopardize integrity of internal components.*

1. Model developed to simulate drop from different angles.
2. Optimization problem: determine angle that maximizes damage.
3. Single function eval involves:
   - Rotating/remeshing: 2-5 min.
   - Simulating drop: 1 to 15 hrs.

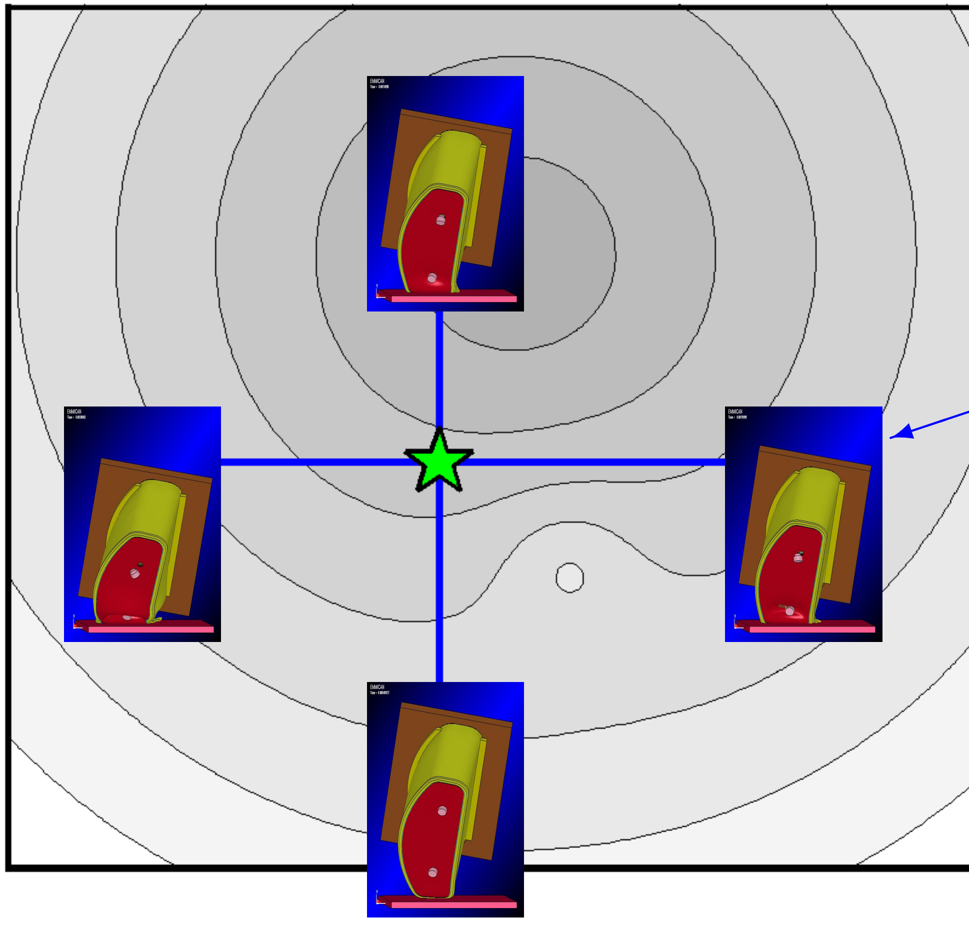


Sandia "Can Crush" problem configuration



Four evaluations performed in parallel.

Sandia "Can Crush" problem configuration

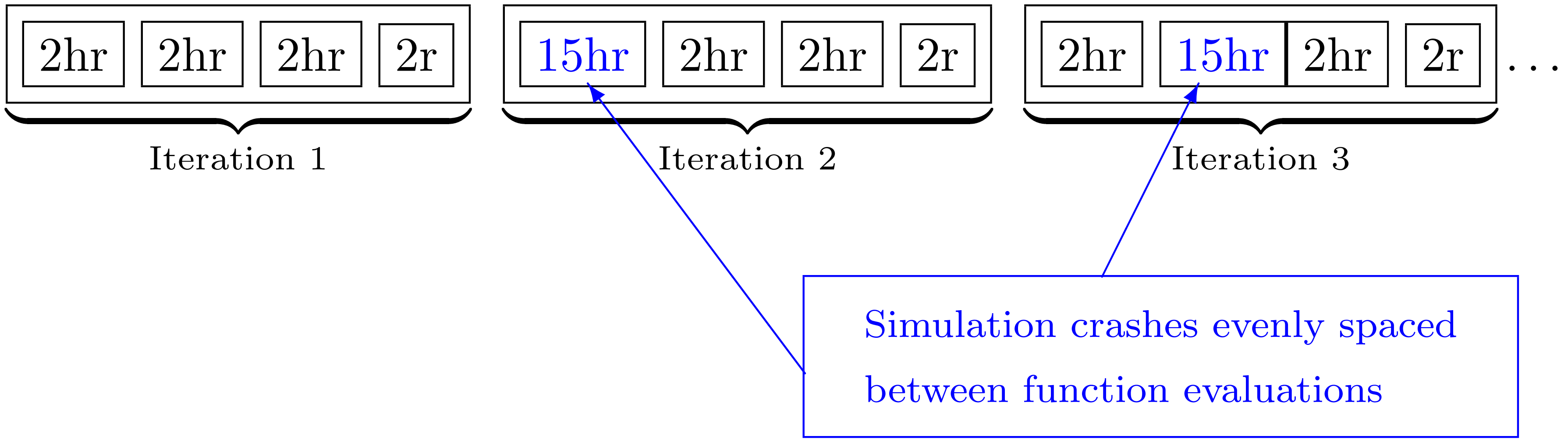


Each evaluation performed on 10 processors.

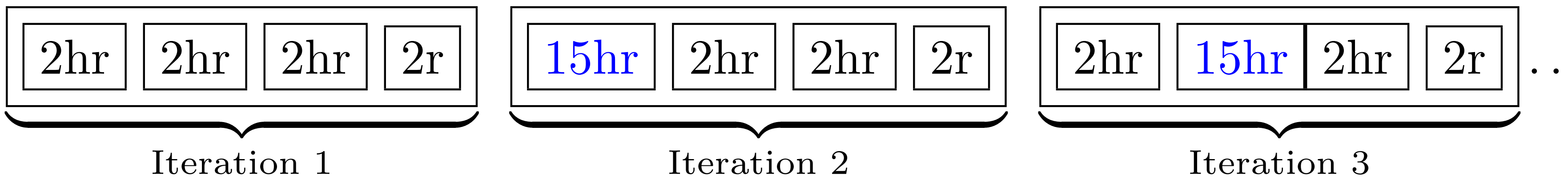
For each simulation

- For initial time step simulation could be unstable.
- Whenever simulation crashed, the time step was reduced and the simulation ran again.
- Approximately 1 out every 5 simulations crashed for initial time step
- With initial time step simulation takes 1-2 hours.
- With smaller time step simulation takes 10-15 hours.

Worse case scenario for synchronous case



Worse case scenario for synchronous case



Implication

- 4 out of 5 iterations take 15hrs.
- 1 out of 5 iterations takes 2hrs.
- 4 out of 5 iterations, 30 processors are left idle for 13 of the 15 hours.

Punchline Approximately 84% of clock-time, 75% of available processors are not being used!

Asynchronous algorithms can greatly reduced time processors spend idle

Handling nonlinear constraints

A sequence of linearly constrained problems

Nonlinearly constraints

Consider the following problem

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & f(x) \\ \text{subject to} & \begin{array}{rcl} Ax & \leq & b \\ c(x) & = & 0 \end{array} \end{array}$$

Implementation based upon

- Conn, Gould, and Toint. (1996)
- Lewis and Torczon. (2002)
- Kolda, Lewis, and Torczon . (Pending)

The subproblem

We solve a series of linearly constrained subproblems for λ_k, μ_k fixed:

$$\begin{array}{ll} \min\limits_{x \in \mathbb{R}^n} & \Phi_k(x) \\ \text{subject to} & Ax \leq b \end{array}$$

where

$$\Phi_k(x) \triangleq f(x) + \lambda_k^T c(x) + \frac{1}{2\mu_k} \|c(x)\|^2$$

Each subproblem is solved approximately using APPSPACK.

Key feature: Algorithm can be shown to be globally convergent to first-order optimal points without accessing/estimating derivatives.

Basic frame work with derivatives

while not converged **do**

Solve subproblem approximately until

$$\|P_{\mathcal{T}_k}(-\nabla_x \Phi_k(x))\| \leq C\omega_k$$

$P_{\mathcal{T}_k}(\cdot)$ denotes projection onto $\mathcal{T}(x, \omega_k)$.

Update λ_k, μ_k.

if $\|c(x_k)\| \leq \eta_k$, (infeasibility sufficiently reduced)

$$\lambda_{k+1} = \lambda_k + c(x_k)/\mu_k \text{ (Hestenes-Powell)}$$

otherwise $\mu_{k+1} = \tau\mu_k$. (increase penalty)

end

Conn, Gould, Sartenaer, Toint (1996).

Basic frame work with derivatives

while not converged **do**

Solve subproblem approximately until

$$\|P_{\mathcal{T}_k}(-\nabla_x \Phi_k(x))\| \leq C\omega_k$$

$P_{\mathcal{T}_k}(\cdot)$ denotes projection onto $\mathcal{T}(x, \omega_k)$.

Update λ_k, μ_k.

if $\|c(x_k)\| \leq \eta_k$, (infeasibility sufficiently reduced)

$\lambda_{k+1} = \lambda_k + c(x_k)/\mu_k$ (Hestenes-Powell)

otherwise $\mu_{k+1} = \tau \mu_k$. (increase penalty)

end

Main problem: no access to first derivatives.

Borrowing from linearly constrained optimization theory

We know that at unsuccessful iterations

$$\|P_{\mathcal{T}(x,\hat{\Delta})}(-\nabla_x \Phi_k)\| \leq C(\Phi_k, A)\hat{\Delta}$$

Recall we need a bound of the form

$$\|P_{\mathcal{T}(x,\omega_k)}(-\nabla_x \Phi_k)\| \leq C\omega_k$$

where C is independent of k. Dependence on k removed by normalizing wrt $\|\lambda_k\|$ and $1/\mu_k$:

$$\text{choose step tolerance} \leq \omega_k \frac{1}{1 + \|\lambda_k\| + 1/\mu_k}.$$

Preliminary numerical results

- Current test suite consists of 18 Hock and Schittkowski CUTEr problems that have nonlinear equality constraints and ≤ 10 variables
- Current implementation caches $f(x)$ and $c(x)$

Stopping criteria:

$$\Delta_{(k,tol)} \leq 10^{-4}$$

$$\|c(x)\| \leq 10^{-4}$$