A New Method For Numerical Constrained Optimization

Motivation

• The applicability of optimization methods is widespread, reaching into almost every activity in which numerical information is processed

• For a summary of applications and theory
  - See Fletcher “Practical Methods of Optimization”

• For numerous applications in computer graphics
  - See Goldsmith and Barr “Applying constrained optimization to computer graphics”

• In this sketch, we describe a method and not its application
Informal Problem Statement

• An ideal problem for constrained optimization
  - has a single measure defining the quality of a solution (called the *objective function* $F$)
  - plus some requirements upon that solution that must not be violated (called the *constraints* $C_i$)

• A constrained optimization method maximizes (or minimizes) $F$ while satisfying the $C_i$’s

• Both $F$ and $C_i$’s are functions of $x \in \mathbb{R}^N$, the input parameters to be determined

Informal Problem Statement

• Many flavors of optimization
  - $x$ can be real-valued, integer, mixed
  - $F$ and $C_i$’s can be linear, quadratic, nonlinear
  - $F$ and $C_i$’s can be smooth (i.e., differentiable) or nonsmooth
  - $F$ and $C_i$’s can be noisy or noise-free
  - methods can be globally convergent or global

• Our focus
  - globally convergent methods
  - real-valued, nonlinear, potentially nonsmooth, potentially noisy, constrained problems
Our Contribution

• A new method for constraint handling, called *partitioned performances*, that
  - can be applied to established optimization algorithms
  - can improve their ability to traverse constrained space

• A new optimization method, called *SPIDER*, that
  - applies partitioned performances to a new variation of the Nelder and Mead polytope algorithm

An observation leads to an idea

• Observation
  - Many constrained problems have optima that lie near constraint boundaries
  - Consequently, avoidance (or approximations) of constraints can hinder an algorithm’s path to the answer

• Idea
  - By allowing (and even *encouraging*) an optimization algorithm to move its vertices into constrained space, a more efficient and robust algorithm emerges
The idea leads to a method

- Constraints are partitioned (i.e., grouped) into multiple levels (i.e., categories)
- A constrained performance, independent of the objective function, is defined for each level
- A set of rules, based on these partitioned performances, specify the ordering and movement of vertices as they straddle constraint boundaries
- These rules are non-greedy, permitting vertices at a higher (i.e., better) level to move to a lower (i.e., worse) level

Partitioned Performances (Advantages)

- Do not use a penalty function and thus do not warp the performance surface
  - this avoids the possible ill-conditioning of the objective function typical in penalty methods
- Do not linearize the constraints as do other methods (e.g., SQP)
- Assume very little about the problem form
  - $F$ and $C_i$’s can be nonsmooth (i.e., nondifferentiable) and highly nonlinear
Partitioning Constraints

• One effective partitioning of constraints
  - place simple limits on $x \in \mathbb{R}^N$ into level 1 (e.g., $x_1 \geq 0$)
  - place constraints which, when violated, produce singularities in $F$ into level 1
  - all other constraints into level 2
  - and the objective function $F$ into level 3

• Many different strategies for partitioning
  - just two levels: constrained and feasible
  - a level for every constraint, and a feasible level
  - dynamic partitioning (changing the level assignments during the search)

Computing Performance

• Assume a partitioning of $F$ and the $C_i$’s into $W$ levels $[L_1 \ldots L_w]$ with $L_w = \{ F \}$

• We define the \textit{partitioned performance} of a location $x \in \mathbb{R}^N$ as a 2-tuple $<P,L>$ consisting of a floating point scalar $P$ and an integer level indicator $L$. $P$ represents the “goodness” of $x$ at level $L$. 
Computing Performance

- To determine \(<P,L>\)
  - sum the constraint violations in each level
  - \(L\) is assigned to the first level, beginning at level 1, to have any violation and \(P\) is assigned the sum of the violations at \(L\)
  - if no violations occur, \(L \leftarrow W\) and \(P \leftarrow F(x)\)

Comparing Performances

- The partitioned performances of two locations \(x_1\) \((<P_1,L_1>)\) and \(x_2\) \((<P_2,L_2>)\) are compared as follows:
  - if \((L_1 = L_2)\)
    - if \((P_1 > P_2)\) \(x_1\) is better, otherwise \(x_2\) is better
  - if \((L_1 > L_2)\)
    - \(x_1\) is better
  - if \((L_2 > L_1)\)
    - \(x_2\) is better
SPIDER Method

- Applies partitioned performances to a new variation of the Nelder and Mead polytope algorithm
- Rules for ordering and movement using partitioned performances are demonstrated

What is a “SPIDER”? 

- Assuming we are maximizing an n-dimensional objective function $F$, SPIDER consists of $n+1$ “legs”, where
  - each leg contains its position in space
  - associated with each leg is a partitioned performance
What is a “SPIDER”? 

When \( n = 2 \), a triangle  
When \( n = 3 \), a tetrahedron

What does SPIDER do? 

- Crawl: each leg is at a known “elevation” on the performance “hill”, and it is SPIDER’s task to crawl up the hill and maximize performance.
How SPIDER walks

- By moving each leg through the centroid of the remaining legs

Before reflection and expansion

![Diagram of leg before and after reflection and expansion]

After reflection and expansion

How SPIDER walks

- Repeat N times
  - Sort legs of SPIDER, from worst to best. Label worst and best legs.
  - For each leg L, in worst to best order
    - Determine centroid
    - Compute position and performance of a trial leg, \( L_{\text{trial}} \)
      - if L is not the best leg, reflect and expand through centroid
      - if L is the best leg, reflect and expand away from centroid
    - If move successful, accept trial, relabel worst and best leg if required
  - EndFor
  - Shrink SPIDER if best leg has not improved
  - Rebuild SPIDER if successive shrinks exceed threshold
- EndRepeat
Rules for centroid computation

- Exclude leg being moved (L)
- Exclude legs at a lower level than L
  - this helps to give SPIDER a better sense of direction along constraint boundaries

Rules for moving a non-best leg

- Same level (level of L_{\text{trial}} = = \text{level of L})
  - accept trial leg if
    - P value of L_{\text{trial}} > P value of L
- Going down levels (level of L_{\text{trial}} < \text{level of L})
  - accept trial leg if its better than the worst leg
- Going up levels (level of L_{\text{trial}} > \text{level of L})
  - accept trial leg if its better than the best leg
Rules for moving the best leg

- It must improve in performance in order to move
- This gives SPIDER the ability to “straddle” and thus track along a constraint boundary

Rules for shrinking SPIDER

- Shrink the vertices at the same level as the best leg toward the best leg, and flip (as well as shrink) vertices at lower levels over the best leg
- Flipping helps to move legs across a constraint boundary towards feasibility
A Matlab Test Problem

• Sequential Quadratic Programming (SQP) methods represent the state-of-the-art in nonlinear constrained optimization

• SQP methods out perform every other tested method in terms of efficiency, accuracy, and percentage of successful solutions, over a large number of test problems

• On a Matlab test problem
  - Matlab SQP Implementation, 96 function calls
  - SPIDER, 108 function calls
The End