Mitsubishi Electric Research Laboratory
MERL - Cambridge Research

Technical Report 2001-TR2001-14 March 27, 2001

A New Method For Numerical Constrained Optimization

Ronald N. Perry

Abstract

Numerical constrained optimization is an important tool with extensive applications in computer graphics and many other diverse fields. In computer animation, for example, constrained optimization has been used for finding initial conditions, smoothing or finishing motion paths, interpolating orientations, animating flexible bodies, self-assembly of objects, and finding boundaries.

An ideal problem for constrained optimization is one that has a single measure defining the quality of a solution plus some requirements upon that solution that must not be violated. The quality measure is called the *objective function*; the requirements are called *constraints*. A constrained optimization solver maximizes (or minimizes) the objective function while satisfying the constraints. In this report, we propose a new method for constraint handling that can be applied to established optimization algorithms and which significantly improves their ability to traverse through constrained space. To make the presentation concrete, we apply the new constraint method to the Nelder and Mead polytope algorithm. The resulting technique, called SPIDER, has shown great initial promise for solving difficult (e.g., nonlinear, nondifferentiable, noisy) constrained problems.

Presented at SIGGRAPH 2001 Conference Abstracts and Applications.

A New Method For Numerical Constrained Optimization

Ronald N. Perry
Mitsubishi Electric Research Laboratory

Introduction

Numerical constrained optimization is an important tool with extensive applications in computer graphics [3] and many other diverse fields [1]. In computer animation, for example, constrained optimization has been used for finding initial conditions, smoothing or finishing motion paths, interpolating orientations, animating flexible bodies, self-assembly of objects, and finding boundaries [3].

An ideal problem for constrained optimization is one that has a single measure defining the quality of a solution plus some requirements upon that solution that must not be violated. The quality measure is called the ***objective function*** (denoted as F below); the requirements are called ***constraints*** (denoted as C_i below). A constrained optimization solver maximizes (or minimizes) the objective function while satisfying the constraints. In this sketch, we propose a new method for constraint handling that can be applied to established optimization algorithms and which significantly improves their ability to traverse through constrained space. To make the presentation concrete, we apply the new constraint method to the Nelder and Mead polytope algorithm [2]. The resulting technique, called SPIDER, has shown great initial promise for solving difficult (e.g., nonlinear, nondifferentiable, noisy) constrained problems.

The Idea

Many constrained problems have optima that lie near constraint boundaries. Consequently, avoidance of constraints can hinder an algorithm's path to the answer. By allowing (and even encouraging) an optimization algorithm to move its vertices into constrained space, a more efficient and robust algorithm emerges. In the new method, constraints are partitioned into multiple levels. 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 (employing the insight stated above) have been shown to significantly aid motion along constraints toward an optimum. Note that the new approach uses no penalty function and thus does not warp the performance surface, thereby avoiding the possible ill-conditioning of the objective function typical in penalty methods.

Problem Statement

Maximize $F(\mathbf{x})$ for all $\mathbf{x} \in R^N$ such that $C_i(\mathbf{x}) \geq 0$ where $F:R^N \rightarrow R$, $C_i:R^N \rightarrow R$, and $i = [1 \ldots M]$. F or any C_i may be highly nonlinear, nondifferentiable, and/or noisy. A function (e.g., F) is considered ***noisy*** when repeated evaluation of that function with identical input yields different values.

SPIDER Algorithm

1. Pick a starting location $\mathbf{x_s} \in R^N$

2. Partition F and the constraints C_i into W levels $[L_1 \ldots L_W]$ with $L_W = \{ F \}$. We define the ***performance*** of a location $\mathbf{x} \in R^N$ 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. P and L are computed as follows:

   a. $P_1 \leftarrow \sum \min(C_k(\mathbf{x}), 0)$, for all C_k in L_1
   b. If (P_1 is nonzero) $P \leftarrow P_1$, $L \leftarrow 1$, Stop
   c. $P_2 \leftarrow \sum \min(C_k(\mathbf{x}), 0)$, for all C_k in L_2
   d. If (P_2 is nonzero) $P \leftarrow P_2$, $L \leftarrow 2$, Stop
   e. …
   f. …
   g. $P_W \leftarrow F(\mathbf{x})$, $P \leftarrow P_W$, $L \leftarrow W$, Stop

   The performances of two locations $\mathbf{x_1}$ ($<P_1,L_1>$) and $\mathbf{x_2}$ ($<P_2,L_2>$) are compared as follows:

   a. If ($L_1 == L_2$) if ($P_1 > P_2$) $\mathbf{x_1}$ is better else $\mathbf{x_2}$ is better, Stop
   b. If ($L_1 > L_2$) $\mathbf{x_1}$ is better else $\mathbf{x_2}$ is better

3. Determine the performance of the starting location $\mathbf{x_s}$ to form the starting vertex $\mathbf{v_s} = <\mathbf{x_s},P_s,L_s>$

4. Pick the scale of each dimension to form the ***size of space*** vector $\mathbf{s} \in R^N$

5. Generate an initial set of N+1 vertices $[\mathbf{v_1} \ldots \mathbf{v_{n+1}}]$ using $\mathbf{v_s}$ and **s**. For example (where N = 2 and vector indices begin at 1),

   $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3} \leftarrow \mathbf{v_s}$
   $\mathbf{v_2}.\mathbf{x}[1] \leftarrow \mathbf{v_2}.\mathbf{x}[1] + \text{random}() * \mathbf{s}[1]$
   $\mathbf{v_3}.\mathbf{x}[2] \leftarrow \mathbf{v_3}.\mathbf{x}[2] + \text{random}() * \mathbf{s}[2]$

6. Determine the performance of each vertex $[\mathbf{v_1} \ldots \mathbf{v_{n+1}}]$

7. Sort the vertices $[\mathbf{v_1} \ldots \mathbf{v_{n+1}}]$ from worst to best. Label the overall best vertex $\mathbf{B_{all}}$ and the overall worst vector $\mathbf{W_{all}}$.

8. For each vertex **v** in the set $[\mathbf{v_1} \ldots \mathbf{v_{n+1}}]$ from worst to best:

   a. Determine the centroid **c** of $[\mathbf{v_1} \ldots \mathbf{v_{n+1}}]$, excluding **v** and other vertices at a lower level than **v**. If there are insufficient vertices to compute a valid centroid (e.g., < 2), incrementally include other vertices at lower levels until there are a sufficient number.

   b. If **v** is not the best vertex:
      b1. $\mathbf{x_t} \leftarrow \mathbf{c} + (\mathbf{c} - \mathbf{v}.\mathbf{x}) * \text{expansionFactor}$ (e.g., 1.1)
      b2. $<P_t,L_t> \leftarrow \text{ComputePerf}(\mathbf{x_t})$
      b3. Form trial vertex $\mathbf{v_t}$: $\mathbf{v_t}.\mathbf{x} \leftarrow \mathbf{x_t}$, $\mathbf{v_t}.P \leftarrow P_t$, $\mathbf{v_t}.L \leftarrow L_t$
      b4. Either accept $\mathbf{v_t}$ or reject $\mathbf{v_t}$ (where accept ***means*** replace **v** with $\mathbf{v_t}$) using the following criteria: if $\mathbf{v_t}.L == \mathbf{v}.L$, accept $\mathbf{v_t}$ if $\mathbf{v_t}.P > \mathbf{v}.P$; if $\mathbf{v_t}.L > \mathbf{v}.L$, accept $\mathbf{v_t}$ if $\text{Perf}(\mathbf{v_t}) > \text{Perf}(\mathbf{B_{all}})$; if $\mathbf{v_t}.L < \mathbf{v}.L$, accept $\mathbf{v_t}$ if $\text{Perf}(\mathbf{v_t}) > \text{Perf}(\mathbf{W_{all}})$

   c. If **v** is the best vertex:
      c1. $\mathbf{x_t} \leftarrow \mathbf{c} - (\mathbf{c} - \mathbf{v}.\mathbf{x}) * \text{expansionFactor}$ (e.g., 1.1)
      c2. $<P_t,L_t> \leftarrow \text{ComputePerf}(\mathbf{x_t})$
      c3. Form trial vertex $\mathbf{v_t}$: $\mathbf{v_t}.\mathbf{x} \leftarrow \mathbf{x_t}$, $\mathbf{v_t}.P \leftarrow P_t$, $\mathbf{v_t}.L \leftarrow L_t$
      c4. Accept $\mathbf{v_t}$ if $\text{Perf}(\mathbf{v_t}) > \text{Perf}(\mathbf{v})$

   d. Relabel $\mathbf{B_{all}}$ and $\mathbf{W_{all}}$ if either has changed

9. If $\text{Perf}(\mathbf{B_{all}})$ has not improved, shrink the vertices at the same level as $\mathbf{B_{all}}$ toward $\mathbf{B_{all}}$, and flip (as well as shrink) vertices at lower levels over $\mathbf{B_{all}}$. The later rule helps to move legs (vertices) across a constraint boundary towards feasibility.

10. If the number of successive shrinks exceeds some threshold, rebuild the vertex set using $\mathbf{B_{all}}$ as $\mathbf{v_s}$ and the current size in each dimension as **s** (see step 5)

11. If there are more iterations to perform, repeat from step 7

Notes and Extensions

1. One effective classification of constraints is to place simple limits on **x** that are independent of F into level 1, all other constraints into level 2, and the objective function F into level 3. Many different strategies for classification are being explored.

2. SPIDER permits dynamic classification of constraints at anytime. This classification can be specified by a user (through observation of how SPIDER is moving) or by a classification algorithm which, for example, categorizes constraints into "active" and "inactive" levels.

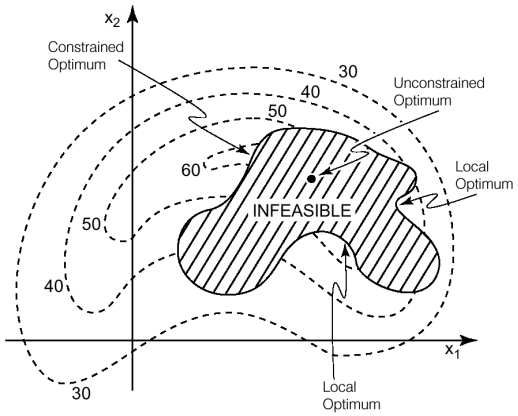
3. Other non-polytope optimization methods [2] can be modified to use the dynamic partitioned performances and corresponding rules introduced above, thereby improving their ability to traverse constrained space.

Results

Although we have tested SPIDER on some difficult problems with great initial success, much remains to prove the viability of this technique. In the talk we will demonstrate, through a series of contour plots, how SPIDER traverses constrained space and why the partitioned performances, along with the specified rules, enable an optimization algorithm to better navigate difficult terrain.

Summary

Partitioned performances permit many optimization algorithms, including SPIDER, to better traverse constrained space. The dynamic classification of constraints, either by a user or a classification algorithm, further enhances navigation through difficult terrain. Finally, the SPIDER algorithm, including the centroid computation, leg flipping when shrinking, and cycling through all legs before resorting, is a robust method for solving difficult constrained problems.



References

[1] R. Fletcher, Practical Methods of Optimization, Second Edition (John Wiley, Chichester and New York, 1987).
[2] Press, W., Flannery, B., Teukolsky, S., and Vetterling, W., Numerical Recipes, Cambridge University Press, New York, NY, 1986.
[3] Goldsmith, Jeff and Alan H. Barr, "Applying constrained optimization to computer graphics," SMPTE Journal, Vol. 102, No. 10, October 1993, pp. 910-912.