Research Article

A Newton-Type Algorithm for Solving Problems of Search Theory

Liping Zhang

Department of Mathematical Sciences, Tsinghua University, Beijing 100084, China

Correspondence should be addressed to Liping Zhang; lzhang@mail.tsinghua.edu.cn

Received 9 March 2012; Accepted 15 December 2012

Academic Editor: Chandal Nahak

Copyright © 2013 Liping Zhang. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In the survey of the continuous nonlinear resource allocation problem, Patriksson pointed out that Newton-type algorithms have not been proposed for solving the problem of search theory in the theoretical perspective. In this paper, we propose a Newton-type algorithm to solve the problem. We prove that the proposed algorithm has global and superlinear convergence. Some numerical results indicate that the proposed algorithm is promising.

1. Introduction

We consider the problem

$$\max_{x \in X} \sum_{i=1}^{n} a_i \left(1 - \exp(-b_i x_i)\right),$$  \hspace{1cm} (1)

where $X = \{x \in \mathbb{R}_+^n \mid e^T x = c\}$, $a, b \in \mathbb{R}_+^n$, $c > 0$, and $e \in \mathbb{R}^n$ is the vector of ones. The problem described by (1) is called the theory of search by Koopman [1] and Patriksson [2]. It has the following interpretation: an object is inside box $i$ with probability $a_i$, and $-b_i$ is proportional to the difficulty of searching inside the box. If the searcher spends $x_i$ time units looking inside box $i$, then he/she will find the object with probability $1 - \exp(-b_i x_i)$. The problem described by (1) represents the optimum search strategy if the available time is limited to $c$ time units. Such problems in the form of (1) arise, for example, in searching for a lost object, in distribution of destructive effort such as a weapons allocation problem [3], in drilling for oil, and so forth [2]. Patriksson [2] surveyed the history and applications as well as algorithms of Problem (1); see [2, Sections 2.1.4, 2.1.5, and 3.1.2]. Patriksson pointed out that Newton-type algorithms have not been theoretically analyzed for the problem described by (1) in the references listed in [2].

Recently, related problems and methods were considered in many articles, for example, [4–6]. For example, a projected pegging algorithm was proposed in [5] for solving convex quadratic minimization. However, the question proposed by Patriksson [2] was not answered in the literature. In this paper, we design a Newton-type algorithm to solve the problem described by (1). We show that the proposed algorithm has global and superlinear convergence. According to the Fischer-Burmeister function [7], the problem described by (1) can be transformed to a semismooth equation. Based on the framework of the algorithms in [8, 9], a smoothing Newton-type algorithm is proposed to solve the semismooth equation. It is shown that the proposed algorithm can generate a bounded iteration sequence. Moreover, the iteration sequence superlinearly converges to an accumulation point which is a solution to the problem described by (1). Numerical results indicate that the proposed algorithm has good performance even for $n = 10000$.

The rest of this paper is organized as follows. The Newton-type algorithm is proposed in Section 2. The global and superlinear convergence is established in Section 3. Section 4 reports some numerical results. Finally, Section 5 gives some concluding remarks.

The following notation will be used throughout this paper. All vectors are column ones, the subscript $T$ denotes transpose, $\mathbb{R}_+^n$ (resp., $\mathbb{R}$) denotes the space of $n$-dimensional real column vectors (resp., real numbers), and $\mathbb{R}_+^n$ and $\mathbb{R}_+$ denote the nonnegative and positive orthants of $\mathbb{R}_+^n$ and $\mathbb{R}$, respectively. Let $\Phi'$ denote the derivative of the function $\Phi$. We define $N := \{1, 2, \ldots, n\}$. For any vector $x \in \mathbb{R}_+^n$, we denote
by \( \text{diag}\{x_i : i \in N\} \) the diagonal matrix whose \( i \)-th diagonal element is \( x_i \) and \( \text{vec}\{x_i : i \in N\} \) the vector \( x \). The symbol \( \| \cdot \| \) stands for the 2-norm. We denote by \( S \) the solution set of Problem (1). For any \( \alpha, \beta > 0 \), \( \alpha = O(\beta) \) (resp., \( \alpha = o(\beta) \)) means \( \alpha/\beta \) is uniformly bounded (resp., tends to zero) as \( \beta \to 0 \).

2. Algorithm Description

In this section, we formulate the problem described by (1) as a semismooth equation and develop a smoothing Newton-type algorithm to solve the semismooth equation.

We first briefly recall the concepts of NCP, semismooth and smoothing functions [10–12].

**Definition 1.** A function \( \phi : \mathbb{R}^2 \to \mathbb{R} \) is called an NCP function if
\[
\phi(u, v) = 0 \iff u \geq 0, \quad v \geq 0, \quad uv = 0. \tag{2}
\]

**Definition 2.** A locally Lipschitz function \( F : \mathbb{R}^n \to \mathbb{R}^m \) is called semismooth at \( x \in \mathbb{R}^n \) if \( F \) is directionally differentiable at \( x \) and for all \( Q \in \partial F(x + d) \) and \( d \to 0 \),
\[
F(x + d) - F(x) - Qd = O(d), \tag{3}
\]
where \( \partial F \) is the generalized Jacobian of \( F \) in the sense of Clarke [13].

\( F \) is called strongly semismooth at \( x \in \mathbb{R}^n \) if \( F \) is semismooth at \( x \) and for all \( Q \in \partial F(x + d) \) and \( d \to 0 \),
\[
F(x + d) - F(x) - Qd = O(\|d\|). \tag{4}
\]

**Definition 3.** Let \( \mu \neq 0 \) be a parameter. Function \( F_\mu(x) \) is called a smoothing function of a semismooth function \( F(x) \) if it is continuously differentiable everywhere and there is a constant \( \bar{c} > 0 \) independent of \( \mu \) such that
\[
\|F_\mu(x) - F(x)\| \leq \bar{c}\mu, \quad \forall x. \tag{5}
\]

The Fischer-Burmeister function [7] is one of the well-known NCP functions:
\[
\phi(u, v) = u + v - \sqrt{u^2 + v^2}. \tag{6}
\]

Clearly, the Fischer-Burmeister function defined by (6) is not smooth, but it is strongly semismooth [14]. Let \( \varphi : \mathbb{R}^3 \to \mathbb{R} \) be the perturbed Fischer-Burmeister function defined by
\[
\varphi(\mu, u, v) = u + v - \sqrt{u^2 + v^2 + \mu^2}. \tag{7}
\]

It is obvious that for any \( \mu > 0 \), \( \varphi \) is differentiable everywhere and for each \( \mu \geq 0 \), we have
\[
|\varphi(\mu, u, v) - \phi(u, v)| \leq \mu, \quad \forall (u, v) \in \mathbb{R}^2. \tag{8}
\]

In particular, \( \varphi(0, u, v) = \phi(u, v) \) for all \((u, v) \in \mathbb{R}^2\). Namely, \( \varphi \) defined by (7) is a smoothing function of \( \phi \) defined by (6).

According to Kuhn-Tucker theorem, the problem described by (1) can be transformed to
\[
e^T x = c,
\]
\[
x_i \geq 0, \quad s - a_i b_i \exp(-b_i x_i) \geq 0, \quad \forall i \in N, \tag{9}
\]
\[
x_i (s - a_i b_i \exp(-b_i x_i)) = 0, \quad \forall i \in N,
\]
where \( s \in \mathbb{R} \).

Define
\[
\Psi(s, x) := \left( \phi(x_i, s - a_i b_i \exp(-b_i x_i)) \right)_{i \in N}, \tag{10}
\]
\[
H(s, x) := \left( e^T x - c \right) / \Psi(s, x). \tag{11}
\]

Based on the perturbed Fischer-Burmeister function defined by (7), we obtain the following smooth equation:
\[
G(y) := G(\mu, s, x) := \left( e^T x - c \right) / \Phi(\mu, s, x) = 0 \tag{12}
\]
where
\[
\Phi(\mu, s, x) := \left( \varphi(\mu, x_i, s - a_i b_i \exp(-b_i x_i)) \right)_{i \in N}. \tag{13}
\]

Clearly, if \( y^* = (0, s^*, x^*) \) is a solution to (12) then \( x^* \) is an optimal solution to the problem described by (1).

We give some properties of the function \( G \) in the following lemma, which will be used in the sequel.

**Lemma 4.** Let \( G \) be defined by (12). Then \( G \) is semismooth on \( \mathbb{R}^{m+2} \) and continuously differentiable at any \( y = (\mu, s, x) \in \mathbb{R}_+ \times \mathbb{R}^{n+1} \) with its Jacobian
\[
G'(y) = \begin{pmatrix}
1 & 0 & 0 \\
0 & 0 & e^T \\
\Phi'_\mu & \Phi'_s & \Phi'_x
\end{pmatrix}, \tag{14}
\]
where
\[
\Phi'_\mu := \text{vec} \left\{ -\frac{\mu}{w_i} : i \in N \right\},
\]
\[
\Phi'_s := \text{vec} \left\{ 1 - \frac{s - a_i b_i \exp(-b_i x_i)}{w_i} : i \in N \right\},
\]
\[
\Phi'_x := \text{diag} \left\{ \left( 1 - \frac{x_i}{w_i} \right) + a_i b_i \exp(-b_i x_i) \right\}_{i \in N}. \tag{15}
\]
with \( u_i := \sqrt{(x_i^2 + (s - a_i b_i \exp(-b_i x_i))^2 + \mu^2)} \), \( i \in N \). Moreover, the matrix \( G'(y) \) is nonsingular on \( R^m + R^n \).

Proof. \( G \) is semismooth on \( R^{m+2} \) due to the strong semismoothness of \( \phi(u, v) \). \( G \) is continuously differentiable on \( R^m + R^n \). For any \( \mu > 0 \), (14) can be obtained by a straightforward calculation from (12). Clearly, we have for any \( \mu > 0 \),

\[
0 < 1 - \frac{x_i}{u_i} < 2, \quad 0 < (\Phi'_{sij})_i < 2,
\]

(16)

\[
a, b, c_i^2 \exp(-b_i x_i) > 0, \quad \forall i \in N,
\]

which implies that \( (\Phi'_{sij})_i > 0 \) for all \( i \in N \). Let

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & e^T & 0 \\
\Phi'_{v} & \Phi'_{s} & \Phi'_{x} \\
\end{pmatrix}
\begin{pmatrix}
\nu \\
v \\
z
\end{pmatrix} = 0.
\]

(17)

Then, we have \( u = 0 \) and

\[
\sum_{i=1}^{n} z_i = 0, \quad (\Phi'_{sij})_i u_i + (\Phi'_{sij})_j z_i = 0, \quad \forall i \in N.
\]

(18)

The second equality in (18) implies

\[
z_i = -\frac{(\Phi'_{sij})_j}{(\Phi'_{sij})_i} v_i, \quad i \in N.
\]

(19)

Since \( (\Phi'_{sij})_i > 0 \) and \( (\Phi'_{sij})_j > 0 \) for \( i \in N \), the first equality in (18) yields \( v = 0 \) and hence \( z = 0 \). Therefore, the matrix \( G'(y) \) defined by (14) is nonsingular for \( \mu > 0 \).

We now propose a smoothing Newton-type algorithm for solving the smooth equation in (12). It is a modified version of the smoothing Newton method proposed in [8]. The main difference is that we add a different perturbed item in Newton equation, which allows the algorithm to generate a bounded iteration sequence. Let \( y = (\mu, s, x) \in R^{m+2} \) and \( y \in (0, 1) \). Define a function \( \rho : R^{m+2} \rightarrow R^n \) by

\[
\rho(y) := y \|G'(y)\| \min \{1, \|G'(y)\|\}.
\]

(20)

Algorithm 5

Step 0. Choose \( \delta, \sigma \in (0, 1) \) and \( \mu^0 > 0 \). Let \( \mathbf{U} := (\mu^0, 0, 0) \in R^m + R \times R^n \). Let \( s^0 \in R \) and \( x^0 \in R^n \) be arbitrary points. Let \( y^0 := (\mu^0, s^0, x^0) \). Choose \( y \in (0, 1) \) such that \( y \|G'(y)\| < 1 \) and \( y \mu^0 < 1 \). Set \( k := 0 \).

Step 1. If \( G(y^k) = 0 \), stop. Otherwise, let \( \rho_k := \rho(y^k) \).

Step 2. Compute \( \Delta y^k := (\Delta \mu^k, \Delta s^k, \Delta x_k) \in R^{m+2} \) by

\[
G(y^k) + G'(y^k) \Delta y^k = \rho_k \mathbf{u}.
\]

(21)

Step 3. Let \( m_k \) be the smallest nonnegative integer such that

\[
\|G(y^k + \delta^{m_k} \Delta y^k)\| \leq [1 - \sigma (1 - y \mu^0)^2 \delta^{m_k}] \|G(y^k)\|.
\]

(22)

Let \( \lambda_k := \delta^{m_k} \).

Step 4. Set \( y^{k+1} := y^k + \lambda_k \Delta y^k \) and \( k := k + 1 \). Go to Step 1.

The following theorem proves that Algorithm 5 is well defined.

Theorem 5. Algorithm 5 is well defined. If it finitely terminates at \( k \)th iteration then \( x^k \) is an optimal solution to the problem described by (1). Otherwise, it generates an infinite sequence \( \{y^k = (\mu^k, s^k, x^k)\} \) with \( \mu^k > 0 \) and \( \mu^k \geq \rho_k \mu^0 \).

Proof. If \( \mu^k > 0 \) then Lemma 4 shows that the matrix \( G'(y^k) \) is nonsingular. Hence, Step 2 is well defined at the \( k \)th iteration.

For any \( 0 < \alpha \leq 1 \), define

\[
R(\alpha) := G(y^k + \alpha \Delta y^k) - G(y^k) - \alpha G'(y^k) \Delta y^k.
\]

(23)

It follows from (21) that

\[
\Delta \mu^k = -\mu^k + \rho_k \mu^0.
\]

(24)

Hence, for any \( 0 < \alpha \leq 1 \), we have

\[
\mu^k + \alpha \Delta \mu^k = (1 - \alpha) \mu^k + \alpha \rho_k \mu^0 > 0.
\]

(25)

From Lemma 4, \( G \) is continuously differentiable around \( y^k \). Thus, (23) implies that

\[
R(\alpha) = 0(\alpha).
\]

(26)

On the other hand, (20) yields

\[
\rho_k \leq \|G(y^k)\|, \quad \rho_k \leq \|G(y^k)\|^2.
\]

(27)

Therefore, for any sufficiently small \( \alpha \in (0, 1) \),

\[
\|G(y^k + \alpha \Delta y^k)\| \leq \|R(\alpha)\| + (1 - \alpha) \|G(y^k)\| + \alpha \rho_k \mu^0
\]

\[
\leq (1 - \alpha) \|G(y^k)\| + \alpha y \mu^0 \|G(y^k)\| + o(\alpha)
\]

\[
= \left[1 - (1 - y \mu^0) \alpha\right] \|G(y^k)\| + o(\alpha),
\]

(28)

where the first inequality follows from (21) and (23), and the second one follows from (26) and (27). Inequality in (28) implies that there exists a constant \( 0 < \delta \leq 1 \) such that

\[
\|G(y^k + \alpha \Delta y^k)\| \leq \left[1 - \sigma (1 - y \mu^0) \alpha\right] \|G(y^k)\|,
\]

(29)

\( \forall \alpha \in (0, \delta] \).

This inequality shows that Step 3 is well defined at the \( k \)th iteration. In addition, by (24), Steps 3 and 4 in Algorithm 5, we have

\[
\mu^{k+1} = (1 - \lambda_k) \mu^k + \lambda_k \rho_k \mu^0 > 0
\]

(30)

holds since \( 0 < \lambda_k \leq 1 \) and \( \mu^k > 0 \). Consequently, from \( \rho_k > 0 \) and the above statements, we obtain that Algorithm 5 is well defined.

It is obvious that if Algorithm 5 finitely terminates at \( k \)th iteration then \( G(y^k) = 0 \), which implies that \( \mu^k = 0 \) and
(s, x) satisfies (9). Hence, x is an optimal solution to the problem described by (1).

Subsequently, we assume that Algorithm 5 does not finitely terminate. Let \{y\} = (μ, s, x) be the sequence generated by the algorithm. It follows that \( \mu_k > 0 \). We want to prove that \{y\} satisfies \( \mu_k \geq \rho_k \mu^0 \) through the induction method. Clearly, \( \rho(y) \leq \gamma \|G(y)\| < 1 \), which yields \( y \in \Omega \). Assume that \( \mu_k \geq \rho_k \mu^0 \); then (24) yields

\[
\begin{align*}
\mu_k + \rho_k \mu^0 &= \mu_k + \lambda_k \Delta \mu_k \\
&= (1 - \lambda_k) \mu_k + \lambda_k \rho_k \mu^0 - \rho_k \mu^0 \\
&\geq (1 - \lambda_k) \rho_k \mu^0 + \lambda_k \rho_k \mu^0 - \rho_k \mu^0 \\
&\geq \mu^0 (\rho_k - \rho_{k+1}).
\end{align*}
\]

Clearly,

\[
\rho_k = \begin{cases}
\frac{\|G(y)\|^2}{\|G(y)\|}, & \text{if } \|G(y)\| < 1, \\
\gamma \|G(y)\|, & \text{otherwise}.
\end{cases}
\]

It follows from (20) and (22) that

\[
\|G(y^k+1)\| \leq \|G(y^k)\|,
\]

\[
\rho_k \leq \gamma \|G(y^k)\|,
\]

\[
\rho_k \leq \gamma \|G(y^k)\|^2.
\]

Hence, combining (31), (32), and (33), we obtain that \( \mu_k \geq \rho_k \mu^0 \) which gives the desired result.

### 3. Convergence Analysis

In this section we establish the convergence property for Algorithm 5. We show that the sequence \{y\} = (μ, s, x) generated by Algorithm 5 is bounded and its any accumulation point yields an optimal solution to the problem described by (1). Furthermore, we show that the sequence \{y\} is superlinearly convergent.

**Theorem 6.** The sequence \{y\} = (μ, s, x) generated by Algorithm 5 is bounded. Let \( y^* = (\mu^*, s^*, x^*) \) denote an accumulation point of \{y\}. Then \( \mu^* = 0 \), and \( x^* \) is the optimal solution of the problem described by (1).

**Proof.** By Theorem 5,

\[
\mu_k + \rho_k \mu^0 = (1 - \lambda_k) \mu_k + \lambda_k \rho_k \mu^0 \\
\leq (1 - \lambda_k) \mu_k + \lambda_k \mu^0 = \mu^k,
\]

which implies that \( \mu_k \) is monotonically decreasing and hence converges. It follows from (22) that

\[
\|G(y^k+1)\| \leq \|G(y^k)\|, \quad \forall k.
\]

Hence, \( \|G(y^k)\| \) also converges. Consequently, there exists a constant \( M > 0 \) such that \( \|G(y^k)\| \leq M \) for all k. This implies that \( \mu_k \) and \( \{x^k\} \) are bounded, and that for any \( k \) and \( i \in N \),

\[
\begin{align*}
\|x_k + (s_k - a_i b_i \exp(-b_i x_k)) \| &\leq \sqrt{(s_k)^2 + (s_k - a_i b_i \exp(-b_i x_k))^2} + (\mu_k^2) \\
&\leq M.
\end{align*}
\]

This shows that \{s\} is also bounded. Consequently, \{y\} is bounded. Let \( y^* = (\mu^*, s^*, x^*) \) be an accumulation point of \{y\}. We assume, without loss of generality, that \{y\} converges to \( y^* \). Then, we have

\[
\lim_{k \to \infty} \|G(y^k)\| = \|G(y^*)\|, \quad \lim_{k \to \infty} \mu_k = \mu^*.
\]

By (20),

\[
\lim_{k \to \infty} \rho_k = \rho_* := \gamma \|G(y^*)\| \min \{1, \|G(y^*)\|\}.
\]

Suppose \( \|G(y^*)\| > 0 \) by contradiction. Then \( \rho_* > 0 \). From Theorem 5 and (22), we have

\[
\mu^* \geq \mu^0 > 0, \quad \lim_{k \to \infty} \lambda_k = 0.
\]

Hence, from Step 3 of Algorithm 5, we obtain

\[
\|G(y_k + \frac{\lambda_k}{\delta} \Delta y_k)\| \geq \left[ 1 - \sigma \left(1 - \gamma \mu^0\right) \frac{\lambda_k}{\delta} \right] \|G(y_k)\|.
\]

Taking \( k \to \infty \) in (40) and then combining with (21), we have

\[
\|G(y^*)\| \geq \|G(y_k)\| \geq \|G(y^*)\| \geq \|G(y^*)\|,
\]

which yields

\[
\rho_* \mu^0 \geq \left[ 1 - \sigma \left(1 - \gamma \mu^0\right) \right] \|G(y^*)\|.
\]

Since \( \rho_* \leq \gamma \|G(y^*)\| \), (42) implies

\[
(1 - \sigma) \left(1 - \gamma \mu^0\right) \leq 0,
\]

which contradicts \( \sigma < 1 \) and \( \mu^0 < 1 \). Therefore, \( \|G(y^*)\| = 0 \) and then \( \mu^* = 0 \). Consequently, \( x^* \) is the optimal solution of the problem described by (1).
Proof. Define two index sets:
\[ N_1 := \{ i \in N : (x_i^*)^2 + (s^* - a_i b_i \exp(-b_i x_i^*))^2 \neq 0 \}, \]
\[ N_2 := \{ j \in N : x_j^* = 0, s^* = a_j b_j \}. \]
By a direct computation, we get
\[ Q = \begin{pmatrix} 0 & e^T \\ \alpha & A \\ \beta & B \end{pmatrix}, \]
where \( A \) is an \(|N_1| \times n\) matrix with all elements being zero except the \((i, i)\)-th as \( A_i \), for \( i \in N_1 \), \( B \) is an \(|N_2| \times n\) matrix with all elements being zero except the \((j, j)\)-th as \( B_j \) for \( j \in N_2 \), and
\[ \alpha = \text{vec} \left\{ 1 - \frac{s^* - a_i b_i \exp(-b_i x_i^*)}{\sqrt{(x_i^*)^2 + (s^* - a_i b_i \exp(-b_i x_i^*))^2}} : i \in N_1 \right\}, \]
\[ A_i = 1 - \frac{x_i^*}{\sqrt{(x_i^*)^2 + (s^* - a_i b_i \exp(-b_i x_i^*))^2}} \]
\[ + a_i b_i^2 \exp(-b_i x_i^*) \alpha_i, \quad i \in N_1, \]
\[ B_j = \{ (u, v + a_j b_j^2 u) : (u - 1)^2 + (v - 1)^2 \leq 1 \}, \]
\[ j \in N_2. \]

Obviously,
\[ 0 < \alpha_i < 2, \quad 0 < A_i < 2 \left( 1 + a_i b_i^2 \right), \quad i \in N_1, \]
\[ \beta_j \geq 0, \quad B_j > 0, \quad j \in N_2. \]
Let \( Q(w, x) = 0 \). Then we have
\[ \sum_{i \in N_1} z_i + \sum_{j \in N_2} z_j = 0; \]
\[ \alpha_i w + A_i z_i = 0, \quad i \in N_1; \]
\[ \beta_j w + B_j z_j = 0, \quad j \in N_2, \]
which implies
\[ z_i = -\frac{\alpha_i}{A_i} w, \quad i \in N_1; \]
\[ z_j = -\frac{\beta_j}{B_j} w, \quad j \in N_2. \]
Therefore,
\[ \left( \sum_{i \in N_1} \frac{\alpha_i}{A_i} + \sum_{j \in N_2} \frac{\beta_j}{B_j} \right) w = 0, \]
which, together with (47), yields \( w = 0 \). Thus, (49) implies \( z = 0 \), and hence the matrix \( Q \) is nonsingular.

\[ \text{Theorem 8. Let} \{ \gamma^k \} \text{ be the iteration sequence generated by Algorithm 5. Then} \{ \gamma^k \} \text{ superlinearly converges to} \gamma^*, \text{ that is,} \]
\[ \| \gamma^{k+1} - \gamma^* \| = o(\| \gamma^k - \gamma^* \|). \]
Proof. By Theorem 6, \( \| \gamma^* \| \) is bounded and then let \( \gamma^* = (\mu^*, s^*, x^*) \) be its any accumulation point. Hence, \( G(\gamma^*) = 0 \) and all matrices \( Q \in \partial G(\gamma^*) \) are nonsingular from Lemma 7. By Proposition 3.1 in [12],
\[ \| G'(\gamma^*)^{-1} \| = O(1) \]
for all \( \gamma^k \) sufficiently close to \( \gamma^* \). From Lemma 4, we know that \( G \) is semismooth at \( \gamma^* \). Hence, for all \( \gamma^k \) sufficiently close to \( \gamma^* \),
\[ \| G(\gamma^k) - G'(\gamma^*) (\gamma^k - \gamma^*) \| = o(\| \gamma^k - \gamma^* \|^2). \]
which implies that
\[ \rho_k \mu_0 \leq \gamma \mu^0 \| G(\gamma^k) \|^2 = O(\| G(\gamma^k) \|^2) = O(\| \gamma^k - \gamma^* \|^2). \]
This inequality, together with (51) and (52), yields
\[ \| \gamma^k + \Delta \gamma^k - \gamma^* \|
\[ \leq \| G'(\gamma^*)^{-1} (\| G(\gamma^k) - G'(\gamma^*) (\gamma^k - \gamma^*) \| + \rho_k \mu_0) \|
\[ = o(\| \gamma^k - \gamma^* \|). \]
Following the proof of Theorem 3.1 in [15], we obtain \( \| \gamma^k - \gamma^* \| = O(\| G(\gamma^k) \|) \) for all \( \gamma^k \) sufficiently close to \( \gamma^k \). This implies, from (55), that for all \( \gamma^k \) sufficiently close to \( \gamma^k \),
\[ \| G(\gamma^k + \Delta \gamma^k) - G(\gamma^k + \Delta \gamma^k - \gamma^*) \|
\[ = o(\| \gamma^k - \gamma^* \|). \]
Table 1: Problem (1) with \(a, b, c\) randomly generated in different intervals.

<table>
<thead>
<tr>
<th>DIM</th>
<th>Inter</th>
<th>Gval</th>
<th>CPU (sec.)</th>
<th>AT [5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
<td>4.0397e - 011</td>
<td>0.0016</td>
<td>0.0008</td>
</tr>
<tr>
<td>500</td>
<td>13.6</td>
<td>5.7544e - 010</td>
<td>0.0078</td>
<td>0.0021</td>
</tr>
<tr>
<td>1000</td>
<td>16.1</td>
<td>9.3739e - 010</td>
<td>0.0167</td>
<td>0.0290</td>
</tr>
<tr>
<td>5000</td>
<td>22.6</td>
<td>9.2203e - 010</td>
<td>0.0844</td>
<td>0.0980</td>
</tr>
<tr>
<td>10000</td>
<td>26</td>
<td>9.6499e - 010</td>
<td>0.2182</td>
<td>0.3946</td>
</tr>
</tbody>
</table>

Table 2: Problem (1) with \(a, b, c\) randomly generated in \([0, 1]\).

<table>
<thead>
<tr>
<th>DIM</th>
<th>Inter</th>
<th>Gval</th>
<th>CPU (sec.)</th>
<th>AT [5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>12.3</td>
<td>2.0675e - 009</td>
<td>0.006</td>
<td>0.0017</td>
</tr>
<tr>
<td>500</td>
<td>15.7</td>
<td>1.3929e - 009</td>
<td>0.015</td>
<td>0.0021</td>
</tr>
<tr>
<td>1000</td>
<td>16.8</td>
<td>1.2247e - 009</td>
<td>0.037</td>
<td>0.0524</td>
</tr>
<tr>
<td>5000</td>
<td>19.2</td>
<td>1.0316e - 009</td>
<td>0.119</td>
<td>0.1082</td>
</tr>
<tr>
<td>10000</td>
<td>21.6</td>
<td>1.9056e - 009</td>
<td>0.246</td>
<td>0.4693</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Activity</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(4 \times 10^6)</td>
<td>(3 \times 10^6)</td>
<td>(2 \times 10^6)</td>
<td>(10^5)</td>
</tr>
<tr>
<td>(b)</td>
<td>(2 \times 10^{-6})</td>
<td>(3 \times 10^{-6})</td>
<td>(10^{-6})</td>
<td>(10^{-5})</td>
</tr>
</tbody>
</table>

Table 4

<table>
<thead>
<tr>
<th>Region</th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>0.1013</td>
<td>0.3205</td>
<td>0.1323</td>
<td>0.2730</td>
<td>0.1730</td>
</tr>
<tr>
<td>(b)</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
</tr>
</tbody>
</table>

4. Computational Experiments

In this section, we report some numerical results to show the viability of Algorithm 5. First, we compare the numerical performance of Algorithm 5 and the algorithm in [5] on two randomly generated problems. Second, we apply Algorithm 5 to solve two real-world examples. Throughout the computational experiments, the parameters used in Algorithm 5 were \(\delta = 0.75, \sigma = 0.25, \mu^0 = 0.001,\) and \(\gamma = \min\{1/\|G(y^k)\|, 0.99\}\). In Step 1, we used \(\|G(y^k)\| \leq 10^{-8}\) as the stopping rule. The vector of ones is the initial starting point.

Firstly, problems in the form of (1) with 100, 500, 1000, 5000, and 10000 variables were computed. In the first randomly generated example, \(a_i\) was randomly generated in the interval \([10, 20]\), \(b_i\) was randomly generated in the interval \([1, 2]\) for each \(i \in N\), and \(c\) was randomly generated in the interval \([50, 51]\). In the second randomly generated example, \(a_i\) with \(\sum_{i=1}^{n} a_i = 1\) and \(b_i\) was randomly generated in the interval \([0, 1]\) for each \(i \in N\), and \(c\) was also randomly generated in the interval \([0, 1]\). Each problem was run 30 times. The numerical results are summarized in Tables 1 and 2, respectively. Here, Dim denotes the number of variable; AT [5] denotes the average run time in seconds used by the algorithm in [5]. In particular, we list more items for Algorithm 5. Inter denotes the average number of iterations, CPU (sec.) is the average run time, and Gval denotes the average values of \(\|G(y^k)\|\) at the final iteration.

The numerical results reported in Tables 1 and 2 show that the proposed algorithm solves the test problems much faster than the algorithm in [5] when the size of problem is large.

Secondly, we apply Algorithm 5 to solve two real-world problems. The first example is described in [16]. This problem is how to allocate a maximum amount of total effort, \(c\), among \(n\) independent activities, where \(a_i(1 - \exp(-b_i x_i))\) is the return from the \(i\)th activity, that is, effort \(x_i\), to yield the maximum total return. Note that here \(a_i\) is the potential attainable and \(b_i\) is the rate of attaining the potential from effort \(x_i\). When no effort is devoted to the \(i\)th activity, the value of \(x_i\) is zero. This example usually arises in the marketing field; the activities may correspond to different products, or the same product in different marketing areas, in different advertising media, and so forth. In this example we wish to allocate one million dollars among four activities with values of \(a_i\) and \(b_i\) as in Table 3.

For this problem Algorithm 5 obtained the maximum total return \(4.8952 \times 10^6\) after 25 iterations and elapsing 0.0625 second CPU time. The total effort one million dollars was allocated, 0.5764 million and 0.4236 million to activities \(A\) and \(B\), respectively.

The second example is to search an object in 5 regions, where \(a_i\) is the prior probability with \(\sum_{i=1}^{n} a_i = 1\) of an object of search being in the \(i\)th region and \(1 - \exp(-b_i x_i)\) is the probability of finding an object known to be in the \(i\)th region with \(x_i\) time units. The data of this example, listed in Table 4, come from Professor R. Jiang of Jiaozhou Bureau of Water.
Conservancy: the available time is $c = 30$ time unite, and after 16 iterations and elapsing 0.0781 second CPU time Algorithm 5 computed the result $x = (0, 16.6037, 0, 13.3963, 0)$. This shows that we will spend 17 time units in Region II and 13 time units in Region IV to find water.

5. Conclusions

In this paper we have proposed a Newton-type algorithm to solve the problem of search theory. We have shown that the proposed algorithm has global and superlinear convergence. Some randomly generated problems and two real world problems have been solved by the algorithm. The numerical results indicate that the proposed algorithm is promising.

Acknowledgment

This work was supported by NSFC Grant (11271221).

References
