Paper

Gradient-Based Algorithms in the Brachistochrone Problem Having a Black-Box Represented Mathematical Model

Roman Dębski

Department of Computer Science, AGH University of Science and Technology, Kraków, Poland

Abstract—Trajectory optimization problems with black-box represented objective functions are often solved with the use of some meta-heuristic algorithms. The aim of this paper is to show that gradient-based algorithms, when applied correctly, can be effective for such problems as well. One of the key aspects of successful application is choosing, in the search space, a basis appropriate for the problem. In an experiment to demonstrate this, three simple adaptations of gradient-based algorithms were executed in the forty-dimensional search space to solve the brachistochrone problem having a black-box represented mathematical model. This experiment was repeated for two different bases spanning the search space. The best of the algorithms, despite its very basic implementation, needed only about 100 iterations to find very accurate solutions. 100 iterations means about 2000 objective functional evaluations (simulations). This corresponds to about 20 iterations of a typical evolutionary algorithm, e.g. $ES(\mu,\lambda)$.

Keywords—black-box optimization, brachistochrone problem, optimal control, trajectory optimization.

1. Introduction

The brachistochrone (i.e. the curve of fastest descent) problem was posed by Johann Bernoulli in *Acta Eruditorum* in June 1696. Its original wording was, *"Given two points A and B in a vertical plane, what is the curve traced out by a point acted on only by gravity, which starts at A and reaches B in the shortest time"*. The first who found the solution were: Johann Bernoulli, Johan's brother Jakob, Newton, Leibniz and l'Hôspital [1]. Since then the problem has been studied by mathematicians, physicists and engineers. This is a consequence of the fact that apart form being a classic problem in the calculus of variations it also plays an important role in the trajectory optimization, mainly because some of minimum-time trajectory planning tasks can be reduced to one of generalizations of the brachistochrone problem.

The original problem, which assumes that the particle is falling on a vertical plane in a uniform gravitational field, has an analytical solution, e.g. [2]. So do some of the original problem generalizations – for instance, an introduction of the Coulomb friction force [3]–[6] or the drag force proportional to velocity [5], taking into account a nonuniform gravitational field [7], a motion on surfaces different from a vertical plane [8] or relativistic effects [9]–[11]. Yet many engineering problems (related to trajectory optimization) are too complex to be solved analytically – either by the use of classic calculus of variations methods or, when the problem is put into the optimal control context, the Pontryagin maximum principle [2], [12]–[14]. In such cases other methods have to be used [15]–[19]. At the implementation level, each of these methods is usually based on non-linear programming (the family of gradient/sub-gradient methods [15]), dynamic programming [20] or some meta-heuristics, e.g., evolutionary algorithms, simulated annealing, particle swarm optimization, tabu search.

A special group of trajectory optimization problems consists of those with *black-box represented* objective functions [21], [22]. This is the case, for instance, when values of the objective function (performance measure) are received from simulation. In such situation most of the classic optimization methods cannot be used (at least not directly) and a common practice is to base the optimization process on one of the meta-heuristics[1] [23]–[26]. Although this approach has some drawbacks, e.g. [15], [25], especially when applied to trajectory optimization problems, only a few studies of alternative methods have been carried out, e.g. [27]).

This paper addresses this by showing that gradient-based methods, when applied correctly, can also be effective for trajectory optimization problems having black-box represented mathematical models. An important aspect of the successful application is choosing, in the search space, a basis appropriate for the problem. In an experiment to demonstrate this, three simple adaptations of gradient-based algorithms were executed to solve the brachistochrone problem in search spaces spanned by two different bases. The first one – the natural basis in $\mathbb{R}^n$ – was selected to demonstrate some pitfalls of overly direct application of gradient-based methods to variational problems. This knowledge can be useful both while implementing custom-made trajectory

[1]Because they are usually "derivative-free", i.e. do not use derivatives of the objective function.

optimization software[2] and while using any of the trajectory optimization tools available on the market (MATLAB, OTIS or libraries Trajopt or NTG [28]).

This paper is organized as follows. In Section 2 the optimization problem is presented. Section 3 describes the solution methods proposed – six algorithms derived from non-linear programming. In Section 4 optimization results are discussed. Section 5 contains conclusions of the study. In the last part, which is Appendix, the simulation-based trajectory evaluator used in the experiments is described.

2. Problem Formulation

A black-box optimization occurs when the explicit formula of the objective function (performance measure) is unknown, i.e. it is "opaque" or black-boxed to the optimization routine. A typical example of this situation is when objective function values are taken from a computer simulation. In such problems, derivative-related information is not available and, as a consequence, gradient-based algorithms cannot be applied. What is commonly used instead, is one of the derivative-free (DFO) algorithms (e.g. [22], [27]) or (meta-)heuristics. Another possible approach, which is presented in this paper, is to use approximate values of partial derivatives, e.g. by finite differences in a gradient-based algorithm and, in case of non-convex problems, combine it with a multi-start method.

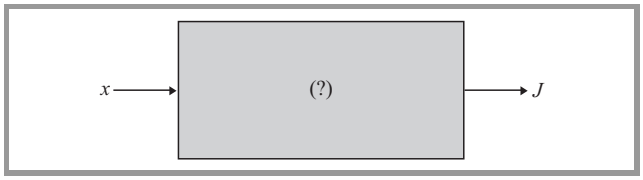

Fig. 1. A black-box functional – the value of J that corresponds to the input can be found only through simulation.

The brachistochrone problem analyzed in this paper covers cases with arbitrarily complex but continuous, black-box represented mathematical models. In this context, the performance measure is expressed by a black-box functional shown in Fig. 1. The input vector $\boldsymbol{x}$ represents a trajectory (encoded in some way) and J – the time of the corresponding displacement. The optimization task is to find x^* corresponding to the minimum value of J, or more formally:

$$\underset{\boldsymbol{x}}{\text{minimize}}\ J(\boldsymbol{x}),\ \text{subject to: } x_0 = A \text{ and } x_{n+1} = B. \qquad (1)$$

A given trajectory represented as a sequence of points

$$(x_0, x_1, \ldots, x_{n+1}) = (P^{(0)}, P^{(1)}, \ldots, P^{(n+1)})$$

in coordinate system $\xi_1 - \xi_2$ is shown in Fig. 2.

[2]It can be necessary e.g. because of some missing functionality in the available tools, their license constrains or the target platform limitations.

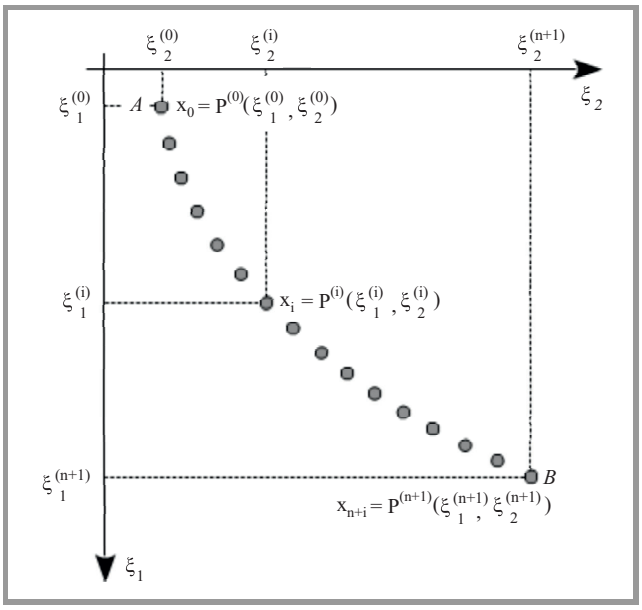

Fig. 2. Trajectory representation.

Note that this representation does not assume anything about the shape of trajectory segments, i.e. $\overset{\frown}{P^{(i-1)}P^{(i)}}, i = 1, \ldots, n+1$.

3. Solution Methods

The optimization algorithms presented in this section are simulation-based. The simulation is represented by function Evaluate, which is shown as Algorithm 1. This function is referenced in the algorithms' pseudo-code.

Algorithm 1 Trajectory evaluation

```
1: function Evaluate(ξ)
2:     //...evaluate (through simulation) ξ, i.e. calc. J(ξ)
3:     return J(ξ)
4: end function
```

The optimization process was based on a series of evaluations of subsequent (admissible) trajectories represented as a series of points (Fig. 2)

$$\boldsymbol{\xi} = \left(\left(\xi_1^{(0)}, \xi_2^{(0)}\right), \left(\xi_1^{(1)}, \xi_2^{(1)}\right), \ldots, \left(\xi_1^{(n+1)}, \xi_2^{(n+1)}\right)\right)^{\top}, \qquad (2)$$

where

$$\left(\xi_1^{(0)}, \xi_2^{(0)}\right) = A\left(\xi_{1A}, \xi_{2A}\right) \qquad (3)$$

and

$$\left(\xi_1^{(n+1)}, \xi_2^{(n+1)}\right) = B\left(\xi_{1B}, \xi_{2B}\right). \qquad (4)$$

The optimization assumed that only ξ_2 components were variated (perturbed) and ξ_1 were fixed in the following regular mesh

$$\xi_1^{(i)} = \xi_{1A} + i\,\frac{\xi_{1B} - \xi_{1A}}{n+1}, \quad i = 0, 1, \ldots, n+1. \qquad (5)$$

Taking into account the boundary conditions and Eqs. 3–4, only points with indexes $1 \ldots n$ were variated (see also Figs. 3–4).

The algorithms discussed in the next subsections contain references to the following symbols:

$\boldsymbol{\xi_0}$ – initial guess trajectory, $\boldsymbol{\xi_0} = \boldsymbol{\xi_{20}} = \left(\xi_2^{(1)}, \ldots, \xi_2^{(n)}\right)^\top$,

e – step size multiplier (assumed to be constant),

δ_0 – stop condition parameter.

In all these algorithms a finite difference based approximation of partial derivatives and gradients was applied. The approximation formulas will be given in each case separately.

3.1. Algorithms in the Search Space Spanned by the Natural Basis

The natural basis in $\mathbb{R}^n$ is usually the first candidate considered in gradient-based optimization tasks. This basis, put into the trajectory optimization context, is shown in Fig. 3 (note the way of representing a n-dimensional space).

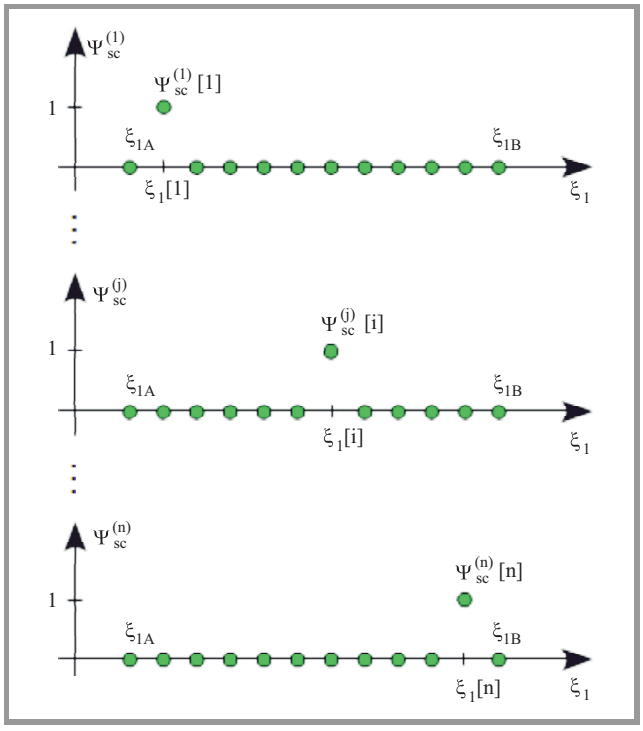

Fig. 3. ψ_{sc}-basis.

In this paper this basis is defined in the following way[3]

$$\left\{\boldsymbol{\psi}_{sc}^{(j)}, \quad j = 1, \ldots, n\right\}, \tag{6}$$

where

$$\boldsymbol{\psi}_{sc}^{(j)}[i] = \delta_{ij} \quad i, j = 1, \ldots, n \tag{7}$$

and δ_{ij} is the Kronecker delta.

In the next part of this section the algorithms defined in this search space are discussed.

[3]Letters $(.)_{sc}$ in the lower index were taken from **s**ingle **c**omponent.

Algorithm SC-FD-SimpGrad

This algorithm uses a forward finite difference (fd) approximation of directional derivative (in the direction of $\boldsymbol{\psi}_{sc}^{(j)}$), expressed in the following way

$$\Delta_{\boldsymbol{\psi}_{sc}^{(j)}}^{(fd)} J = \frac{J\left(\boldsymbol{\xi_2} + \varepsilon \boldsymbol{\psi}_{sc}^{(j)}\right) - J(\boldsymbol{\xi_2})}{\varepsilon}, \quad j = 1, \ldots, n. \tag{8}$$

Having defined the formula for directional derivatives can be calculated the approximation of the gradient vector

$$\hat{\boldsymbol{\nabla}}_{\boldsymbol{\psi}_{sc}}^{(fd)} J = \left(\Delta_{\boldsymbol{\psi}_{sc}^{(1)}}^{(fd)} J, \ldots, \Delta_{\boldsymbol{\psi}_{sc}^{(j)}}^{(fd)} J, \ldots, \Delta_{\boldsymbol{\psi}_{sc}^{(n)}}^{(fd)} J\right)^\top. \tag{9}$$

The algorithm pseudo-code, shown as Algorithm 2, is divided into two parts, with function Grad-Approx being a helper method used to calculate the approximation of the gradient vector.

Algorithm 2 Forward difference based (simple) gradient descent (ψ_{sc}-Basis)

function Grad-Approx($\boldsymbol{\xi}, J_\xi$)
  for $j \leftarrow 1, n$ **do**
    $J_{\xi+\varepsilon} \leftarrow$ Evaluate($\boldsymbol{\xi} + \varepsilon \boldsymbol{\psi}_{sc}^{(j)}$)
    $\hat{\boldsymbol{\nabla}}_{\boldsymbol{\psi}_{sc}}^{(fd)} J[j] \leftarrow \frac{J_{\xi+\varepsilon} - J_\xi}{\varepsilon}$ ▷ see Eq. (8)
  end for
  return $\hat{\boldsymbol{\nabla}}_{\boldsymbol{\psi}_{sc}}^{(fd)} J$
end function

function SC-FD-SimpGrad($\boldsymbol{\xi_0}, e, \delta_0$)
  $J_0 \leftarrow$ Evaluate($\boldsymbol{\xi_0}$)
  $\hat{\boldsymbol{g}}_{\boldsymbol{0}} \leftarrow$ Grad-Approx($\boldsymbol{\xi_0}, J_0$)
  while *true* **do**
    $\boldsymbol{h_0} = -\hat{\boldsymbol{g}}_{\boldsymbol{0}}$
    $\boldsymbol{\xi_1} \leftarrow \boldsymbol{\xi_0} + e\, \boldsymbol{h_0}$
    $J_1 \leftarrow$ Evaluate($\boldsymbol{\xi_1}$)
    $\hat{\boldsymbol{g}}_{\boldsymbol{1}} \leftarrow$ Grad-Approx($\boldsymbol{\xi_1}, J_1$)
    if $J_1 > J_0$ **then** ▷ check stop conditions
      return $(\boldsymbol{\xi_0}, J_0)$
    else if $\frac{|J_0 - J_1|}{J_0} < \delta_0$ **then**
      return $(\boldsymbol{\xi_1}, J_1)$
    end if
    $\boldsymbol{\xi_0} \leftarrow \boldsymbol{\xi_1}$
    $J_0 \leftarrow J_1$
    $\hat{\boldsymbol{g}}_{\boldsymbol{0}} \leftarrow \hat{\boldsymbol{g}}_{\boldsymbol{1}}$
  end while
end function

The algorithm performs $(k+1)(n+1)$ simulations and uses $\Theta(n)$ memory, where k is the total number of iterations in the main optimization routine, and n is the size of the vector representing a trajectory.

Algorithm SC-CD-SimpGrad

This algorithm uses a central finite difference (cd) approximation of directional derivative (in the direction of $\boldsymbol{\psi}_{sc}^{(j)}$), expressed in the following way

$$\Delta_{\boldsymbol{\psi}_{sc}^{(j)}}^{(cd)} J = \frac{J\left(\boldsymbol{\xi}_2 + \varepsilon \boldsymbol{\psi}_{sc}^{(j)}\right) - J\left(\boldsymbol{\xi}_2 - \varepsilon \boldsymbol{\psi}_{sc}^{(j)}\right)}{2\varepsilon}, \quad j = 1, \ldots, n. \tag{10}$$

The approximation of the gradient vector is equal to

$$\hat{\boldsymbol{\nabla}}_{\boldsymbol{\psi}_{sc}}^{(cd)} J = \left(\Delta_{\boldsymbol{\psi}_{sc}^{(1)}}^{(cd)} J, \ldots, \Delta_{\boldsymbol{\psi}_{sc}^{(j)}}^{(cd)} J, \ldots, \Delta_{\boldsymbol{\psi}_{sc}^{(n)}}^{(cd)} J\right)^{\top}. \tag{11}$$

The algorithm pseudo-code, shown as Algorithm 3, is again divided into two parts. The main optimization routine is the same as in Algorithm 2, so is not repeated here.

Algorithm 3 Central difference based (simple) gradient descent (ψ_{sc}-Basis)

```
function Grad-Approx(ξ)
    for j ← 1,n do
        J_{ξ+ε} ← Evaluate(ξ + ε ψ_sc^(j))
        J_{ξ−ε} ← Evaluate(ξ − ε ψ_sc^(j))
        ∇̂^(cd)_{ψ_sc} J[j] ← (J_{ξ+ε} − J_{ξ−ε}) / (2ε)        ▷ see Eq. (10)
    end for
    return ∇̂^(cd)_{ψ_sc} J
end function

function SC-CD-SimpGrad(ξ_0, e, δ_0)
    //...                                                    ▷ see Algorithm 2
end function
```

This algorithm performs $(k+1)(2n+1)$ simulations and uses $\Theta(n)$ memory, where k and n are defined in the same way as in Algorithm 2.

3.2. Algorithms in the Search Space Spanned by the Modified Basis

The basis introduced in this section is more complex and non-orthogonal. It was chosen as an example of a non-standard basis. Its definition[4] is as follows (see Fig. 4, note the way of representing an n-dimensional space)

$$\left\{\boldsymbol{\psi}_{mc}^{(j)}, j = 1, \ldots, n\right\}, \tag{12}$$

where

$$\boldsymbol{\psi}_{mc}^{(j)}[i] = \begin{cases} \frac{\xi_1[i] - \xi_{1A}}{\xi_1[j] - \xi_{1A}}, & 1 \le i \le j, \\ \frac{\xi_{1B} - \xi_1[i]}{\xi_{1B} - \xi_1[j]}, & j < i \le n. \end{cases} \tag{13}$$

In the next paragraphs the algorithms defined in the search space spanned by $\boldsymbol{\psi}_{mc}$ basis are discussed.

[4]Letters $(.)_{mc}$ in the lower index were taken from **m**ulti **c**omponent.

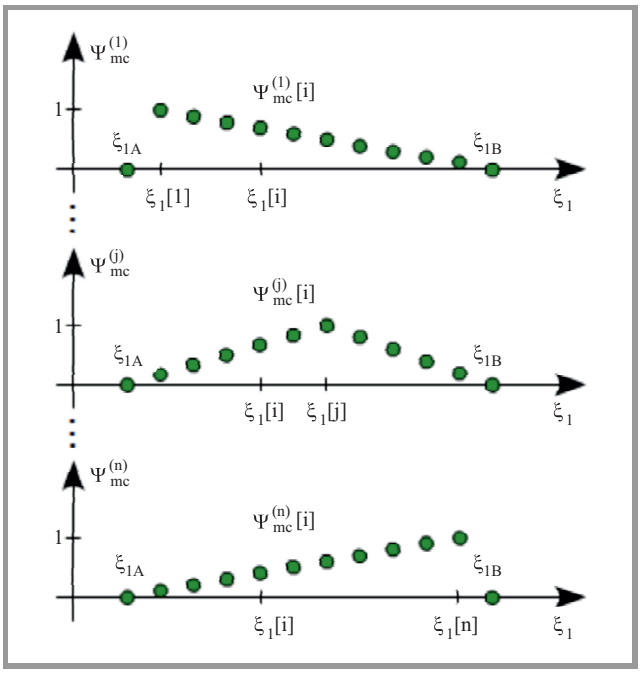

Fig. 4. ψ_{mc}-basis.

Algorithm MC-FD-SimpGrad

In the new basis a forward finite difference (fd) approximation of directional derivative (in the direction of $\boldsymbol{\psi}_{mc}^{(j)}$) is defined as follows

$$\Delta_{\boldsymbol{\psi}_{mc}^{(j)}}^{(fd)} J = \frac{J\left(\boldsymbol{\xi}_2 + \varepsilon \boldsymbol{\psi}_{mc}^{(j)}\right) - J\left(\boldsymbol{\xi}_2\right)}{\varepsilon \|\boldsymbol{\psi}_{mc}^{(j)}\|}, \quad j = 1, \ldots, n, \tag{14}$$

whilst the approximation of the gradient vector is equal to

$$\hat{\boldsymbol{\nabla}}_{\boldsymbol{\psi}_{mc}}^{(fd)} J = \left(\Delta_{\boldsymbol{\psi}_{mc}^{(1)}}^{(fd)} J, \ldots, \Delta_{\boldsymbol{\psi}_{mc}^{(j)}}^{(fd)} J, \ldots, \Delta_{\boldsymbol{\psi}_{mc}^{(n)}}^{(fd)} J\right)^{\top}. \tag{15}$$

Note: $\|.\|$ in the denominator is the L^2-norm.

Algorithm 4 Forward difference based (simple) gradient descent (ψ_{mc}-Basis)

```
function Grad-Approx(ξ, J_ξ)
    for j ← 1,n do
        J_{ξ+ε} ← evaluate(ξ + ε ψ_mc^(j))
        ∇̂^(fd)_{ψ_mc} J[j] ← (J_{ξ+ε} − J_ξ) / (ε ||ψ_mc^(j)||)   ▷ see Eq. (14)
    end for
    return ∇̂^(fd)_{ψ_mc} J
end function

function MC-FD-SimpGrad(ξ_0, e, δ_0)
    //...                                                    ▷ see Algorithm 2
end function
```

MC-FD-SimpGrad algorithm pseudo-code is shown as Algorithm 4 and again, the main optimization routine is the same as in Algorithm 2. This algorithm performs

$(k+1)(n+1)$ simulations and uses $\Theta(n)$ memory, where k and n are defined in the same way as in Algorithm 2.

Central finite difference approximation based algorithms

In the new basis the central finite difference (cd) approximation of directional derivative in the direction of $\psi_{mc}^{(j)}$ can be written as follows:

$$\Delta_{\psi_{mc}^{(j)}}^{(cd)} J = \frac{J\left(\boldsymbol{\xi_2} + \varepsilon \boldsymbol{\psi}_{mc}^{(j)}\right) - J\left(\boldsymbol{\xi_2} - \varepsilon \boldsymbol{\psi}_{mc}^{(j)}\right)}{2\varepsilon \|\boldsymbol{\psi}_{mc}^{(j)}\|}, \quad j = 1, \ldots, n \tag{16}$$

and, as a consequence, the approximation of the gradient vector is equal to

$$\hat{\boldsymbol{\nabla}}_{\psi_{mc}}^{(cd)} J = \left(\Delta_{\psi_{mc}^{(1)}}^{(cd)} J, \ldots, \Delta_{\psi_{mc}^{(j)}}^{(cd)} J, \ldots, \Delta_{\psi_{mc}^{(n)}}^{(cd)} J\right)^{\top}. \tag{17}$$

The above formulas remain the same for the three algorithms presented below.

Algorithm MC-CD-SimpGrad

The algorithm pseudo-code is shown as Algorithm 5 (as before, the main optimization routine is the same as in Algorithm 2). This algorithm performs $(k+1)(2n+1)$ simulations and uses $\Theta(n)$ memory (k and n are defined in the same way as in Algorithm 2).

Algorithm 5 Central difference based (simple) gradient descent (ψ_{mc}-Basis)

```
1: function Grad-Approx(ξ)
2:     for j ← 1,n do
3:         J_{ξ+ε} ← Evaluate(ξ + ε ψ_mc^(j))
4:         J_{ξ−ε} ← Evaluate(ξ − ε ψ_mc^(j))
5:         ∇̂^(cd)_{ψ_mc} J[j] ← (J_{ξ+ε} − J_{ξ−ε}) / (2ε ||ψ_mc^(j)||)   ▷ see Eq. (16)
6:     end for
7:     return ∇̂^(cd)_{ψ_mc} J
8: end function

9: function MC-CD-SimpGrad(ξ_0, e, δ_0)
10:    //...                                                  ▷ see Algorithm 2
11: end function
```

Algorithm MC-CD-SteepDsc

The adaptation of steepest descent algorithm [29] to the brachistochrone problem is shown as Algorithm 6.

This algorithm in each iteration of its main loop performs a minimization along the line

$$\lambda_{min} \leftarrow \underset{\lambda>0}{\operatorname{argmin}} \operatorname{Evaluate}(\boldsymbol{\xi_0} + \lambda \boldsymbol{h_0}) \tag{18}$$

extending from point ξ_0 in the direction of $\boldsymbol{h_0} = -\hat{\boldsymbol{g}}_{\boldsymbol{0}}$ (i.e. minus the local gradient approximate).

Algorithm 6 Central difference based steepest descent (ψ_{mc}-Basis)

```
1: function Grad-Approx(ξ)
2:     //...                                                  ▷ see Algorithm 5
3: end function

4: function MC-CD-SteepDsc(ξ_0, δ_0)
5:     J_0 ← Evaluate(ξ_0)
6:     ĝ_0 ← Grad-Approx(ξ_0)
7:     while true do
8:         h_0 = −ĝ_0
9:         λ_min ← argmin_{λ>0} Evaluate(ξ_0 + λ h_0)
10:        ξ_1 ← ξ_0 + λ_min h_0
11:        J_1 ← Evaluate(ξ_1)
12:        ĝ_1 ← Grad-Approx(ξ_1)
13:        if J_1 > J_0 then                                  ▷ check stop conditions
14:            return (ξ_0, J_0)
15:        else if |J_0 − J_1| / J_0 < δ_0 then
16:            return (ξ_1, J_1)
17:        end if
18:        ξ_0 ← ξ_1
19:        J_0 ← J_1
20:        ĝ_0 ← ĝ_1
21:    end while
22: end function
```

The steepest descent algorithm (see Algorithm 6) performs $(k+1)(2n+1)+l$ simulations and uses $\Theta(n)$ memory, where l is the total number of simulations corresponding to the solution of Eq. (18) and k and n are defined in the same way as in Algorithm 2.

Algorithm MC-CD-ConjGrad

A simple adaptation of the conjugate gradient algorithm [30] to the brachistochrone problem is shown as Algorithm 7. It is one of the most popular and efficient methods in non-linear programming.

The algorithm performs $(1+kn)(1+2n)+l$ simulations and uses $\Theta(n)$ memory, where l, k and n are defined in the same way as in Algorithm 6.

From the simple analysis of Algorithm 7 one can see that the calculation of the approximate gradient is performed at least n times (see the external and internal loops) and therefore this single calculation performs $2n$ simulations. This total number of simulations can seem unexpected, because of the n^2 term, when compared to the previous algorithms, but in the conjugate gradient algorithm k is expected to be very small, often equals 1.

4. Experimental Results

The algorithms discussed in Section 3 were executed using the simulator described in Appendix 1, with combinations of the coefficients μ and k (see Eqs. (20)–(21) in

Algorithm 7 Central difference based conjugated gradient (ψ_{mc}-Basis)

```
1: function Grad-Approx(ξ)
2:     //...                                   ▷ see Algorithm 5
3: end function

4: function MC-CD-ConjGrad(ξ_0, δ_0)           ▷ (see [30])
5:     J_0 ← Evaluate(ξ_0)
6:     ĝ_0 ← Grad-Approx(ξ_0)
7:     while true do
8:         h_0 = −ĝ_0
9:         for j ← 1, n do
10:            λ_min ← argmin_{λ>0} Evaluate(ξ_0 + λ h_0)
11:            ξ_1 ← ξ_0 + λ_min h_0
12:            J_1 ← Evaluate(ξ_1)
13:            ĝ_1 ← Grad-Approx(ξ_1)
14:            if J_1 > J_0 then               ▷ check stop conditions
15:                return (ξ_0, J_0)
16:            else if |J_0 − J_1| / J_0 < δ_0 then
17:                return (ξ_1, J_1)
18:            end if
19:            β ← (ĝ_1 ĝ_1) / (ĝ_0 ĝ_0)
20:            h_1 = −ĝ_1 + β h_0
21:            ξ_0 ← ξ_1
22:            J_0 ← J_1
23:            ĝ_0 ← ĝ_1
24:            h_0 ← h_1
25:        end for
26:    end while
27: end function
```

the Appendix) listed in Table 1. In all cases the initial guess trajectory $\boldsymbol{\xi_0}$ was the straight line between points A and B. The arc AB was approximated by a piecewise-linear function with forty linear segments, so the search space was forty-dimensional. $\xi_1 - \xi_2$ axes were on the surface of the ski slope and the slope angle was assumed to be α. The first experiment setup corresponds to the classic brachistochrone problem. This experiment was performed as a test to check the accuracy of the final solutions obtained from all six algorithms by comparison to the exact solution. These solutions are shown from two perspectives in Figs. 5–6.

Table 1
Experiment setups

Exper. no.	Point A	Point B	α	μ	k
1	(0,0)	(10,10)	15°	**0.00**	**0.00**
2				**0.12**	**0.00**
3				**0.00**	**0.05**
4				**0.12**	**0.05**

Figure 5 shows six trajectories received as a result of the experiment and also, for reference, the straight line AB (as

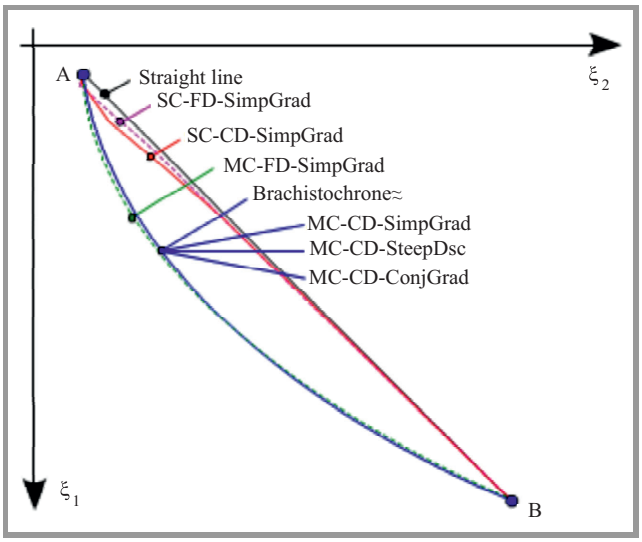

Fig. 5. Simulation results (trajectories) for the classic brachistochrone problem (no friction and no drag, i.e. $\mu = 0.00, k = 0.00$).

the initial guess trajectory, i.e., the start point of each algorithm) and the brachistochrone (i.e. the exact solution). Three results, obtained from MC-CD-SimpGrad, MC-CD-SteepDsc and MC-CD-ConjGrad, were very close to the exact solution. Errors related to the final times were smaller than 0.1% (Fig. 6) and so they are drawn as a single line. On the other hand, the trajectories obtained from SC-FD-SimpGrad and SC-CD-SimpGrad, were very far from the exact solution. The search space for these two algorithms was spanned by $\boldsymbol{\psi}_{sc}$ basis (Fig. 3). These two algorithms will be referenced in this section as sc-algorithms, whilst the other four, defined in the context of ψ_{mc} basis (Fig. 4), as mc-algorithms.

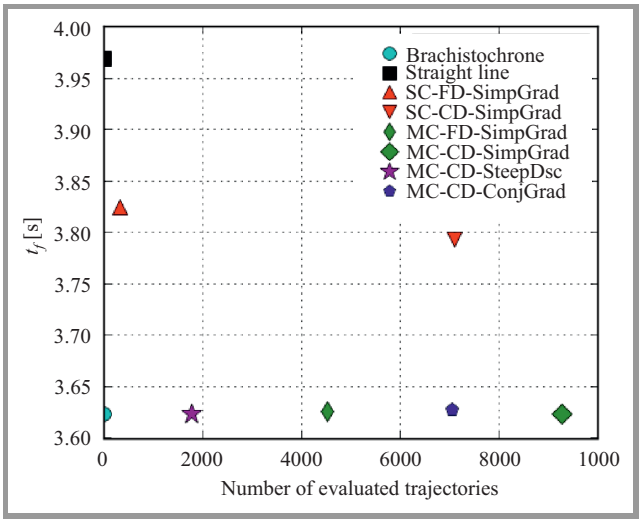

Fig. 6. Simulation results for $\mu = 0.00, k = 0.00$.

Figure 6 shows the experimental results from a different point of view – the algorithms' efficiency and accuracy. Each point represents the final result as a pair of number of evaluated trajectories, and t_f is the final time (to-

tal time of displacement) corresponding to the optimal trajectory. All mc-algorithms performed much better than sc-algorithms. The best of the mc-algorithms was MC-CD-SteepDsc. It needed 1776 evaluations (of different trajectories) to find the solution with the total time of dispacement equal to $t_f = 3.6238$ seconds. The relative error was smaller than 0.01%. MC-CD-ConjGrad performed 7050 evaluations because it did not converge during the first iteration of its external loop as it is often expected to and the algorithm time complexity depends on n^2, see Algorithm 7. Both sc-algorithms performed significantly worse. They were able to improve the initial guess trajectory only by about 3%.

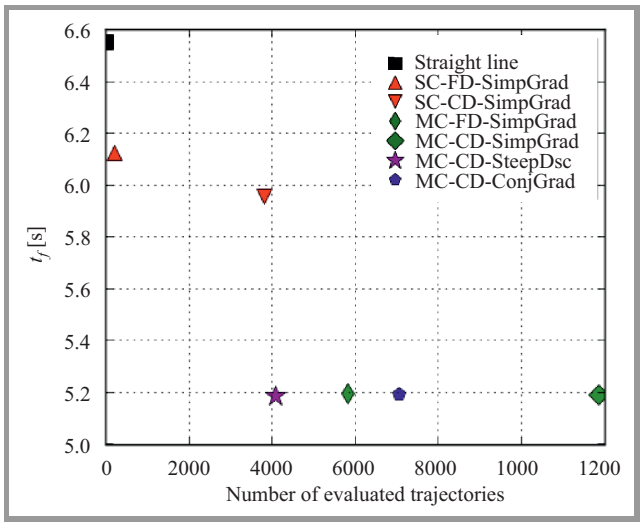

Fig. 7. Simulation results for $\mu = 0.12, k = 0.00$.

Figure 7 presents the results of the second experiment (motion with friction but no drag, i.e. $\mu = 0.12,\ k = 0.00$). Again, the mc-algorithms performed much better than the sc-algorithms and the most efficient was MC-CD-SteepDsc, but this time its advantage was not so significant.

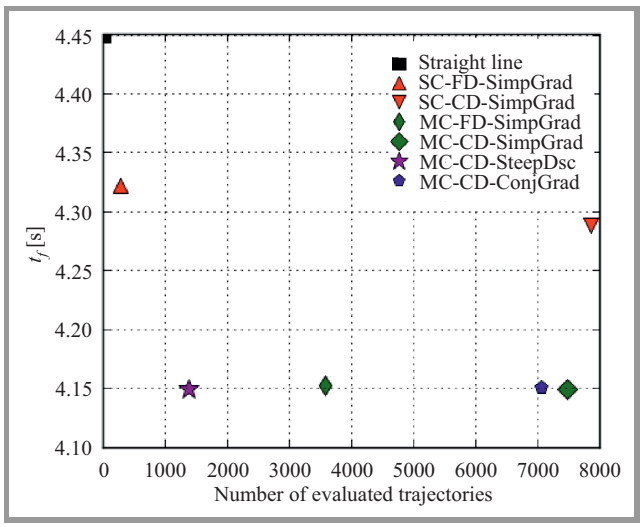

Fig. 8. Simulation results for $\mu = 0.00, k = 0.05$.

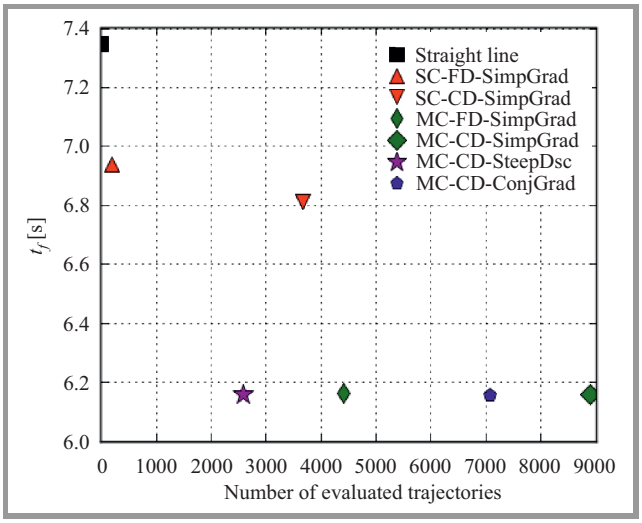

Fig. 9. Simulation results for $\mu = 0.12, k = 0.05$.

Figures 8 and 9 show results from the last two experiments for $\mu = 0.00,\ k = 0.05$ and $\mu = 0.12,\ k = 0.05$. The same pattern can be seen – the most efficient algorithm was again MC-CD-SteepDsc and, in general, the mc-algorithms performed much better than the sc-algorithms.

5. Conclusion

The application of six gradient-based algorithms to the brachistochrone problem having a black-box represented mathematical model has been studied. The main part of this model was a simulation-based trajectory evaluator. As an example of this problem, trajectory optimization in alpine ski racing was chosen.

Each of the six algorithms has been presented in detail (pseudo-code, time and memory complexity). These algorithms were divided into two groups depending on the basis used for spanning their search spaces.

The experimental results have shown that gradient-based algorithms, when applied correctly, can be effective for simulation-based (continuous) trajectory optimization problems. The best of the algorithms, despite its very basic implementation, needed only about 100 iterations corresponding to about 2000 objective function evaluations to find very accurate solutions in a 40-dimensional search space.

Future work could concentrate on experimenting with different bases for the search space. For instance, an orthogonal versus non-orthogonal bases comparison could be carried out. Another area of research could be related to combining the methods presented in this paper with multi-start or memetic algorithms. And finally, the presented algorithms could be verified in an *augmented cloud* environment [31]–[33].

It is worth noting that the presented approach could also be applied to more general variational problems like ”piecewise optimization” of complex trajectories or optimal shape design.

Appendix 1

Simulation-based Trajectory Evaluator (the Black-box Simulator)

Let's consider a skier (modeled as a material point of mass m) going down a slope with angle α from point A to point B. The arc AB is approximated by a piecewise-linear function. This modification simplifies the problem significantly – instead of one (complex) two-dimensional problem, we have a series of (simple) one-dimensional ones. Each of the sub-problems is related to one segment only (Fig.10, note a local coordinate system $\zeta\eta$, set for each segment).

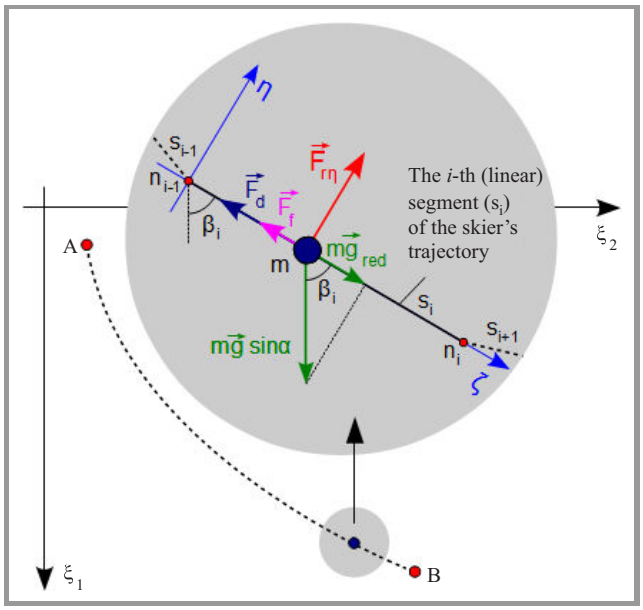

Fig. 10. The forces acting on a skier going down a slope with angle α (one-dimensional approximation model). All forces are reduced to the skier's center of mass and to the surface of the ski slope (i.e. $\xi_1 - \xi_2$).

Equations of motion

The equations of motion for each segment can be written in the following way

$$\begin{cases} m\ddot{\zeta} = mg_{red} - (F_f + F_d) \\ 0 = -F_{r\eta} - mg\sin\alpha\sin\beta \end{cases}, \tag{19}$$

where:

$$F_f = \mu mg\cos\alpha\,, \tag{20}$$

$$F_d = k_1 v^2 = mk\dot{\zeta}^2\,, \tag{21}$$

represent snow resistance (friction) and air resistance (drag), respectively, and

$$g_{red} = g\sin\alpha\cos\beta\,, \tag{22}$$

can be considered as "reduced gravitational acceleration" to the slope plane ($g\sin\alpha$) and to the current linear segment direction ($g\sin\alpha\cos\beta$). Only the first equation is important in the simulation; the second one expresses the condition of equilibrium in the normal direction to the trajectory. After dividing both sides of the first of the Eq. (19) by m and simplifying the expression, for the i^{th} segment we get

$$\ddot{\zeta}_i + k\dot{\zeta}_i^{\,2} = g\,(\sin\alpha\ \cos\beta_i - \mu\cos\alpha)\,. \tag{23}$$

Boundary conditions

The arc $\overset{\frown}{AB}$ is approximated by a piecewise-linear function. We assume that its first segment starts at $A(\xi_{1A}, \xi_{2A})$, and the last one ends at $B(\xi_{1B}, \xi_{2B})$ (Fig. 10). The boundary conditions have to be written now for each segment. An additional assumption has to be introduced into the model – the speed remains constant at the boundary of each pair of subsequent segments, i.e.

$$|\boldsymbol{v}_s^{(i)}| = |\boldsymbol{v}_f^{(i-1)}|, \tag{24}$$

where $\boldsymbol{v}_s^{(i)}$ is the initial speed for the $(i)^{\text{th}}$ segment, and $\boldsymbol{v}_f^{(i-1)}$ is the final speed for the $(i-1)^{\text{th}}$ segment.

Performance measure

In order to find the total time of displacement a series of simulations (one for each segment – s) has to be performed

$$J = t_f = \sum_s t_f^{(s)}. \tag{25}$$

References

[1] J. Babb and J. Currie, "The brachistochrone problem: Mathematics for a broad audience via a large context problem", *The Montana Mathem. Enthus.*, vol. 5, pp. 169–184, 2008.

[2] H. J. Sussmann and J. C. Willems, "300 years of optimal control: from the brachystochrone to the maximum principle", *Control Sys., IEEE*, vol. 17, no. 3, pp. 32–44, 1997.

[3] N. Ashby, W. E. Brittin, W. F. Love, and W. Wyss, "Brachistochrone with coulomb friction", *American J. Phys.*, vol. 43, p. 902, 1975.

[4] J. C. Hayen, "Brachistochrone with coulomb friction", *Int. J. Non-Linear Mechan.*, vol. 40, no. 8, pp. 1057–1075, 2005.

[5] A. S. Parnovsky, "Some generalisations of brachistochrone problem", *Acta Phys. Polonica – Ser. A Gen. Phys.*, vol. 93, pp. 55–64, 1998.

[6] A. M. A. Van der Heijden and J. D. Diepstraten, "On the brachystochrone with dry friction", *In. J. Non-Linear Mechan.*, vol. 10, no. 2, pp. 97–112, 1975.

[7] J. Gemmer, R. Umble, and M. Nolan, "Generalizations of the brachistochrone problem", *arXiv preprint math-ph/0612052*, 2006.

[8] V. Čović and M. Vesković, "Brachistochrone on a surface with coulomb friction", *Int. J. Non-Linear Mechan.*, vol. 43, no. 5, pp. 437–450, 2008.

[9] C. Farina, "Bernoulli's method for relativistic brachistochrones", *J. Phys. A: Mathem. and General*, vol. 20, no. 2, pp. L57–59, 1987.

[10] H. F. Goldstein and C. M. Bender, "Relativistic brachistochrone", *J. Mathem. Phys.*, vol. 27, p. 507, 1986.

[11] G. Mingari Scarpello and D. Ritelli, "Relativistic brachistochrones under electric or gravitational uniform fields", *ZAMM-J. Appl. Mathem. and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 86, no. 9, pp. 736–743, 2006.

[12] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K: Cambridge University Press, 2006 [Online]. Available: http://planning.cs.uiuc.edu/

[13] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mischenko, *The Mathematical Theory of Optimal Processes*. New York-London: Wiley, 1962.

[14] H. J. Sussmann and J. C. Willems, "The brachistochrone problem and modern control theory", in *Contemporary Trends in Nonlinear Geometric Control Theory and its Applications*, A. Anzaldo-Meneses, B. Bonnard, J.-P. Gauthier, and F. Monroy-Perez, Eds. Singapore: World Scientific Publishers, 2002, pp. 113–165.

[15] J. T. Betts, "Survey of numerical methods for trajectory optimization", *J. Guidance Contr. Dynam.*, vol. 21, no. 2, pp. 193–207, 1998.

[16] W. A. Golfetto and S. da Silva Fernandes, "A review of gradient algorithms for numerical computation of optimal trajectories", *J. Aerosp. Technol. Manag.*, vol. 4, no. 2, pp. 131–143, 2012.

[17] A. Jameson and J. Vassberg, "Studies of alternate numerical optimization methods applied to the brachistochrone problem", in *Proc. OptiCON '99 Conf.*, Newport Beach, CA, USA, pp. 281–296, 1999.

[18] E. Olvovsky, "Novel gradient-type optimization algorithms for extremely large-scale nonsmooth convex optimization", PhD thesis, Technion-Israel Institute of Technology, 2005.

[19] A. V. Rao, "A survey of numerical methods for optimal control", *Adv. Astronaut. Sci.*, vol. 135, no. 1, pp. 497–528, 2009.

[20] R. Bellman, "The theory of dynamic programming", Tech. rep., DTIC Document, 1954.

[21] P. Pošik and W. Huyer, "Restarted local search algorithms for continuous black box optimization", *Evolut. Comput.*, vol. 20, no. 4, pp. 575–607, 2012.

[22] P. Pošik, W. Huyer, and L. Pál, "A comparison of global search algorithms for continuous black box optimization", *Evolut. Comput.*, vol. 20, no. 4, pp. 1–32, 2012.

[23] M. Ceriotti and M. Vasile, "MGA trajectory planning with an ACO-inspired algorithm", *Acta Astronautica*, vol. 67, no. 9–10, pp. 1202–1217, 2010.

[24] M. Vasile and M. Locatelli, "A hybrid multiagent approach for global trajectory optimization", *J. Global Optimiz.*, vol. 44, no. 4, pp. 461–479, 2009.

[25] M. Vasile, L. Summerer, and P. De Pascale, "Design of earth–mars transfer trajectories using evolutionary-branching technique", *Acta Astronaut.*, vol. 56, no. 8, pp. 705–720, 2005.

[26] N. Yokoyama, "Trajectory optimization of space plane using genetic algorithm combined with gradient method", in *Proc. 23rd Int. Congr. Aerospace Sciences*, Toronto, Canada, 2002.

[27] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions", *J. Global Optimiz.*, vol. 13, no. 4, pp. 455–492, 1998.

[28] M. B. Milam, "Real-time optimal trajectory generation for constrained dynamical systems", PhD thesis, California Institute of Technology, 2003.

[29] A.-L. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées", *Comp. Rend. Sci. Paris*, vol. serie A, no. 25, pp. 536–538, 1847.

[30] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients", *The Comp. J.*, vol. 7, no. 2, pp. 149–154, 1964.

[31] A. Byrski, R. Dębski, and M. Kisiel-Dorohinicki, "Agent-based computing in an augmented cloud environment", *Comput. Syst. Sci. Eng.*, vol. 21, no. 1, pp. 7–18, 2012.

[32] R. Dębski, A. Byrski, and M. Kisiel-Dorohinicki, "Towards an agent-based augmented cloud", *J. Telecommun. Inform. Technol.*, no. 1, pp. 16–22, 2012.

[33] R. Dębski, T. Krupa, and P. Majewski, ComcuteJS: A web browser based platform for large-scale computations", *Comp. Sci.*, vol. 14, no. 1, pp. 143–152, 2013.

[34] J. Stillwell, "Mathematics and its history", *The Australian Mathem. Soc.*, p. 168, 2002.

[35] G. Galilei, H. Crew, and A. De Salvio, *Dialogues Concerning Two New Sciences*. Charleston: BiblioBazaar, 2010.

Roman Dębski works as an Assistant Professor at the Department of Computer Science at AGH University of Science and Technology. He obtained his M.Sc. in Mechanics (1997) and Computer Science (2002) and Ph.D. specializing in Computational Mechanics (2002). Before joining the University he worked in the IT industry for over 15 years. His current interests include mathematical modeling and computer simulations, parallel processing, heterogeneous computing and trajectory optimization.
E-mail: rdebski@agh.edu.pl
Department of Computer Science
AGH University of Science and Technology
Al. Mickiewicza 30
30-059 Krakow, Poland