Branch and Bound based methods to minimize the energy consumed by an electrical vehicle on long travels with slopes

Riadh Omheni* Frédéric Messine† Abdelkader Merakeb‡

September 2, 2016

Abstract

We consider the problem of minimization of the energy consumed by an electrical vehicle performing quite long travels with slopes. The model we address here, takes into account the electrical and mechanical differential equations of the vehicle. This yields a mixed-integer optimal control problem that can be approximated, using a methodology based on some decomposition steps, by an integer nonlinear global optimization problem. In [8], an efficient Branch and Bound algorithm was proposed to solve this problem but only for flat and short travels ($100m$). In this paper, we show that this algorithm can be extended to solve more operational cases. Firstly, we consider the case of long travels and propose a new Branch and Bound based method for which two powerful heuristics for computing bounds have been developed. Secondly, we deal with routes presenting slopes and develop a new algorithm that allows to efficiently deal with this kind of situations. Note that the main difficulty lies in determining where the changes of slopes occur. A particular attention was devoted to the add of constraints about the acceleration of the vehicle to the optimal control problem in order to provide realistic optimal solutions. Numerous numerical results validating our operational approach are reported.

Keywords: Discretization techniques, optimal control, discrete switch control, Branch and Bound, electrical vehicle, energy consumption.

1 Introduction

Nowadays, the electrical vehicle energy management is an important topic and a lot of researches has been carried out in this field concerning the optimization of the batteries and the optimization of the electrical motors for example. A lot of other works deals with the energy consumed by hybrid vehicle performing a displacement. However, some of these works are dedicated to the optimal control problem of the minimization of the energy consumed by an electrical vehicle performing a fixed displacement. In [4, 11, 5, 12, 9, 1, 16, 2], this subject is discussed but only concerning hybrid vehicles or some simplified models where the electrical equations coming from the motor are not directly taken into account; only the power is considered as a parameter of these models. These simplifications own some advantages to deal with long travels with slopes.

In [7, 8], a more precise model involving the electrical and mechanical differential equations was proposed. This yields to the study of a mixed-integer optimal control problem. A first methodology of resolution based on `MatLab` was introduced in [8] involving a sophisticated

*SAS Institute, Paris, France (`Riadh.Omheni@sas.com`).

†Université de Toulouse, LAPLACE, ENSEEIHT, France (`Frederic.Messine@n7.fr`).

‡Université Mouloud Mammeri de Tizi-Ouzou, Algérie (`merakeb_kader@yahoo.fr`).

Branch and Bound algorithm. More recently, in [15] the same problem was solved close to the global optimality. Indeed, by using direct and indirect shooting numerical techniques [14, 13], upper bounds on the mixed-integer optimal control problem were obtained. Moreover, by using the methods of moments [3, 6] precise lower bounds were provided, proving so the global optimality (with a very small ϵ) of the solutions found by the shooting methods. The solutions found in [15] show that the operational Branch and Bound method presented in [8] provides correct results very close to the best ones; the differences are due to the discretization steps of these two distinct approaches (the shooting techniques are more precise).

In this work, starting from the Branch and Bound algorithm presented in [8], we propose some extensions to deal with more operational cases involving long travels and slopes. Thus, a `Fortran` code is developed and its efficiency is shown in comparison to the `MatLab` one (which was developed and used in [8]). Apart from this, the contributions of the present paper are threefold. First, we propose two new powerful heuristics for computing bounds. These two new methods are proposed based on a study of the properties of the problem leading to an important gain in terms of efficiency and reliability compared to those provided in [8]. This has been illustrated by comparing all our heuristic bounding methods on more than 100 problems. Second, we propose an extension of the Branch and Bound algorithm to deal with long travels. Careful consideration has been given to the acceleration (starting) and deceleration (stopping) phases. Numerical results are provided for travels of $1km$. Third, we propose an extension of the optimal control problem to deal with more operationel case, namely when the route presents slopes. Based on some numerical examples, we show the efficiency of this algorithm and also the fact that constraints about the acceleration of the vehicle have to be inserted into the optimal control problem to provide realistic optimal solutions. We also show on one example that our algorithm can easily deal with long travel presenting slopes.

The paper is organized as follows. In Section 2, we recall the mixed-integer optimal control that we studied in [8, 15]. We also recall the discretized method that we introduced in [8] yielding the approached global optimization problem that we solved so far. In Section 3, we recall the Branch and Bound algorithm introduced in [8]. However, in this paper this algorithm is implemented in `Fortran 90` in place of `MatLab`. Some numerical tests showed the enormous benefit of this work. We propose thereafter, in Section 4, two new efficient heuristics for computing bounds. In Section 5, the Branch and Bound algorithm is extended to deal with long travels. In Section 6, an extension of the optimal control problem is proposed and a new Branch and Bound algorithm is presented in the case where slopes are taken into account. In Section 7, we conclude the paper and discuss about long displacements with slopes by showing on a numerical example that our methodology can efficiently solve those operational problems.

2 Model

The mixed-integer optimal control problem presented in [8, 15] is the following:

$$
\left\{
\begin{array}{ll}
\min\limits_{x,u} & E(x_1,u) = \int_0^{t_f} u(t)x_1(t)V_{alim} + R_{bat}u^2(t)x_1^2(t)dt \\
s.t. & \\
& \dot{x}_1(t) = \frac{u(t)V_{alim} - R_m x_1(t) - K_m x_2(t)}{L_m} \\
& \dot{x}_2(t) = \frac{1}{J}\left(K_m x_1(t) - \frac{r}{K_r}\left(MgK_f + \frac{1}{2}\rho S C_x \left(\frac{x_2(t)r}{K_r}\right)^2\right)\right) \\
& \dot{x}_3(t) = \frac{x_2(t)r}{K_r} \\
& \\
& |x_1(t)| \leq 150 \\
& u(t) \in \{-1, 1\} \\
& \\
& (x_1(0), x_2(0), x_3(0)) = (x_1^0, x_2^0, x_3^0) \in I\!R^3 \\
& (x_1(t_f), x_2(t_f), x_3(t_f)) \in \mathcal{T} \subseteq I\!R^3
\end{array}
\right. \tag{1}
$$

where E represents the electrical energy consumed during the displacement on the cycle $[0, t_f]$ (where t_f denotes the final time). The state variables are: (i) x_1 is the current inside the motor; (ii) x_2 is the angular speed; (iii) x_3 is the position of the car. The control u is in $\{-1, 1\}$ (the physical system can switch in $10^{-9}s$). The other terms are the parameters of the electrical and mechanical parts of the vehicle, which are explained in [8].

First, remark that if we discretize all the interval of time $[0, t_f]$ by fixing the value of the control u, it is necessary to have very small steps about 10^{-3}. Otherwise, the value of the current will change too roughly.

An idea proposed in [8], which directly comes from the use of this car, is to impose during some short laps of time the value of the current inside the electrical motor of the vehicle. This is possible using the control parameter $u(t)$. Thus, if we impose a reference current $iref$, if $x_1(t) > iref + \frac{\Delta}{2}$ then $u(t) := -1$ and if $x_1(t) < iref - \frac{\Delta}{2}$ then $u(t) := 1$. Here, Δ is a predefined tolerance; for the numerical experiments $\Delta = 1$. This technique is just a way to construct a regulator of current which is a first step before making a speed regulator for an electrical car. Hence, the differential system of equations that we consider is the following:

$$
VS(iref, t_0, t_f) := \left\{
\begin{array}{l}
\dot{x}_0(t) = u(t)x_1(t)V_{alim} + R_{bat}u^2(t)x_1^2(t) \\
\dot{x}_1(t) = \frac{u(t)V_{alim} - R_m x_1(t) - K_m x_2(t)}{L_m} \\
\dot{x}_2(t) = \frac{1}{J}\left(K_m x_1(t) - \frac{r}{K_r}\left(MgK_f + \frac{1}{2}\rho S C_x \left(\frac{x_2(t)r}{K_r}\right)^2\right)\right) \\
\dot{x}_3(t) = \frac{x_2(t)r}{K_r} \\
u(t) := \left\{
\begin{array}{l}
-1 \text{ if } x_1(t) > iref + \frac{\Delta}{2} \\
+1 \text{ if } x_1(t) < iref - \frac{\Delta}{2} \\
u(t) \text{ else.}
\end{array}
\right. \\
(x_0(t_0), x_1(t_0), x_2(t_0), x_3(t_0)) = (x_0^{(t_0)}, x_1^{(t_0)}, x_2^{(t_0)}, x_3^{(t_0)}) \in I\!R^4 \\
u(t_0) := 1;
\end{array}
\right. \tag{2}
$$

where t_0 is the initial time which is not necessarily equal to 0 and x_0 is a new state variable linked to E; for given x_1 and u, one has $x_0(t_f) = E(x_1, u)$. This system of differentiable equations can be efficiently solved using a classical differentiable integrator such as for example *Euler, RK2, RK4* with a step of time less than 10^{-3}. The function $VS(iref, t_0, t_f)$ computes in theory all the values for $x_0(t)$, $x_1(t)$, $x_2(t)$ and $x_3(t)$ for all $t \in [t_0, t_f]$ but in practice only values for a discretized time $t_i \in [t_0, t_f]$ are available. Here, we are interested by the final

values of the state variables, hence we define the following function:

$$VSF(iref, t_0, t_f) := (x_0(t_f), x_1(t_f), x_2(t_f), x_3(t_f)) \in I\!R^4, \tag{3}$$

where all the computations are performed using the function VS which makes it possible to solve the system of differential equations (2).

The main idea of this approach is to subdivide the cycle of time $[0, t_f]$ into P subintervals. In each step of time $[t_{p-1}, t_p]$ with $p \in \{1, \cdots, P\}$ ($t_p = p \times \frac{t_f}{P}$), we apply a reference current $iref_p$ which takes values in $[-150, 150]$ in order to directly satisfy the constraint on the state variable x_1 of problem (1). This yields the following continuous global optimization problem:

$$\begin{cases} \min_{iref \in [-150,150]^P} & \sum_{k=1}^{P} x_0^{(k)} \\ s.t. & \\ & (x_0^{(k)}, x_1^{(k)}, x_2^{(k)}, x_3^{(k)}) := VSF(iref_k, t_{k-1}, t_k) \\ & (x_0(0), x_1(0), x_2(0), x_3(0)) = (x_0^{(0)}, x_1^{(0)}, x_2^{(0)}, x_3^{(0)}) \in I\!R^4 \\ & (x_1^{(P)}, x_2^{(P)}, x_3^{(P)}) \in \mathcal{T} \subseteq I\!R^3 \end{cases} \tag{4}$$

Problem (4) is a good approximation of the initial problem (1) which generates just a few number of variables: P. In fact, we use a current regulator system to control the vehicle. This is also interesting in itself for a future implementation of this kind of regulator system in the car.

For later references, we define three matrices: M_{x_0}, M_{x_2} and M_{x_3} where the columns correspond to values where $iref$ is fixed with $x_1^{t_0} = iref$ and where the rows provide values for the entities when a velocity x_2 is given at the beginning of the sample time (we discretize also the possible values of the velocity). In Table 1, we represent the elements of the three matrices, where st_x_2 is a discretization step of the velocity values in km/h. Note that we do not need to compute matrices for x_1 because it will be fixed to $iref$.

	$iref = -150 + (j-1)s$
	$\vdots$
$x_2^{t_{k-1}} = (i-1)st\_x_2$	$\cdots \quad M_{x_k}(i,j)$

Table 1: Elements of the matrices M_{x_0}, M_{x_2} and M_{x_3}.

Using the definition of the function VSF given by (3), it is easy to see that the elements of each matrix are given by the following formula:

$$M_{x_l}(i,j) = \langle VSF(iref, t_{k-1}, t_k),\ e_l \rangle, \quad l \in \{0, 2, 3\} \tag{5}$$

where $e_0 = (1,0,0,0)$, $e_2 = (0,0,1,0)$ and $e_3 = (0,0,0,1)$ and $\langle ., . \rangle$ denotes the standard scalar product.

The electrical vehicle that we consider in this paper is the same as in [8, 15] and has the following characteristics: $R_{\text{bat}} = 0.05\ \Omega, V_{\text{alim}} = 150\ V, R_m = 0.03\ \Omega, K_m = 0.27, L_m = 0.05, r = 0.33m, K_r = 10, M = 250\ kg, g = 9.81, K_f = 0.03, \rho = 1.293\ kg/m^3, S = 2m^2$ and $C_x = 0.4$.

3 Branch and Bound Algorithm: BBA_SC

First, notice that we are not actually able to solve exactly the global optimization problem (4). Thus, we have to discretize the reference current: $iref \in D^P = \{-150, -150+s, -150+2\times s, \cdots, 150\}^P$; we have to take integer values for s which divide exactly the interval $[-150, 150]$. Therefore, the set of solutions becomes finite. In [8], we developed a Branch and Bound algorithm, named BBA_SC, which makes it possible to not explore all the finite large set of solutions. This algorithm, recalled hereafter and referred as Algorithm 1, uses the following classical four stages: (i) extraction, (ii) division, (iii) analysis and (iv) insertion.

Algorithm 1 BBA_SC: Branch-and-Bound algorithm for switch controls

1: Initialization: Let $\mathcal{L}$:= the initial domain D^P in which the minimum is searched, $x_0^{min} := \infty$ the upper bound of the minimum, $st\_Iref := s$ the discretization step of reference current, $x_1^{min} := 0$ the initial solution, $st\_x_2 :=$ the discretization step of the velocity, $x_3^{min} := 0$ the initial position, x_3^f the final position, $st\_tf := \frac{P}{t_f}$ denotes the length of the sample time.
2: Compute three matrices: M_{x_0}, M_{x_2} and M_{x_3}.
3: **while** $\mathcal{L} \neq \emptyset$ **do**
4: Extract an element X of $\mathcal{L}$.
5: Bisect X into two discrete sub-boxes $X^{(1)}$ and $X^{(2)}$ following the largest edge of the convex hull of X.
6: **for** $i = 1$ to 2 **do**
7: Compute (exactly or heuristically) lower bounds x_0^l, x_2^l and x_3^l and upper bounds x_2^u and x_3^u of $X^{(i)}$.
8: **if** $(x_3^u \geq x_3^f)$ and $(x_3^l \leq x_3^f)$ and $(x_0^l < x_0^{min})$ **then**
9: $midi :=$ midpoint of the sub-box $X^{(i)}$
10: $sol := [midi/st\_Iref] \times st\_Iref$
11: Compute trial solutions x_0^{sol}, x_2^{sol} and x_3^{sol}.
12: **if** $(x_3^{sol} \geq x_3^f)$ and $(x_0^{sol} < x_0^{min})$ **then**
13: $x_0^{min} := x_0^{sol}$, $x_1^{min} := sol$, $x_3^{min} := x_3^{sol}$.
14: **end if**
15: **if** $X^{(i)}$ is not reduced to a point **then**
16: Insert $X^{(i)}$ in $\mathcal{L}$.
17: **end if**
18: **end if**
19: **end for**
20: **end while**
21: Results : x_0^{min}, x_1^{min}, x_2^{sol} and x_3^{min}.

After the initialization phase, Algorithm 1 computes at line 2 three matrices: M_{x_0}, M_{x_2} and M_{x_3} using formula (5). This corresponds to a preprocessing step whose the results are used by Algorithm 1 throughout the minimization process. The time for computing these matrices depends on the sample time $t_k - t_{k-1} = \frac{t_f}{P}$, on st_x_2, on the integer value of s, on the integrator step of time (Euler, RK2, RK4, etc.) and on Δ. If we fix the integrator step to 10^{-4}, $st\_x_2 = 0.1$ and $\Delta = 1$, the preprocessing time to compute matrices is proportional to $(t_k - t_{k-1})$ and inversely proportional to s.

The **for** loop in line 6 of Algorithm 1 is the main part of a Branch-and-Bound algorithm. At line 5, the technique to bisect X into two discrete sub-boxes proceeds by a selection of

the component which has the largest width of the convex hull of X and by bisecting it by the middle of this component. At line 7, we compute lower and upper bounds on the state variables that will be used to decide whether or not to reduce the size of the search space. In our case, when a discrete box X is under study, we can compute bounds for x_0, x_2 and x_3 by calculating the integer sets of indices $I = \{i_1, ..., i_n\}$ and $J = \{j_1, ..., j_m\}$ corresponding to the possible values of the velocity at the end of the previous sample and the possible values of *iref*. Thus, we have to compute the bounds which correspond to the minimal and maximal values of $M_{x_0}(i,j), M_{x_2}(i,j), M_{x_3}(i,j)$ for all $(i,j) \in I \times J$. This method is called EM for exact method. In order to obtain the final value for x_0 and x_3, we have to sum all the lower and upper bounds for each sample time $[t_{k-1}, t_k]$.

The exact method EM for computing bounds is time consuming. This leads to a significant increase in the computing time required to explore all the branches of the search tree to ensure that the obtained solution is a global optimum. That is why, we introduced in [8] four heuristics, named $H1$, $H2$, $H3$ and $H4$, which makes it possible to perform rapidly quite efficient bounds. For the sake of completness, we recall hereafter these four heuristics.

- Heuristic $H1$: the values of each sub-matrices at the first row and first column $M_{x_0}(i_1, j_1)$, $M_{x_2}(i_1, j_1)$, $M_{x_3}(i_1, j_1)$ are taken as lower bounds. For upper bounds, we take the values corresponding to the last rows and last columns $M_{x_0}(i_n, j_m)$, $M_{x_2}(i_n, j_m)$, $M_{x_3}(i_n, j_m)$.

- Heuristic $H2$: we keep the same bounds as heuristic $H1$ for position. However, considering the energy and the velocity, we compute the minimum value on the first row for lower bounds and, as upper bounds, the maximum value on the last row:

  - Lower bounds: $\min_{j \in J} M_{x_0}(i_1, j), \min_{j \in J} M_{x_2}(i_1, j), M_{x_3}(i_1, j_1)$.
  - Upper bounds: $\max_{j \in J} M_{x_0}(i_n, j), \max_{j \in J} M_{x_2}(i_n, j), M_{x_3}(i_n, j_m)$.

- Heuristic $H3$: we keep the same bounds as heuristic $H1$ for position and velocity. However, for the energy, the value of the sub-matrix at the last row and the first column $M_{x_0}(i_n, j_1)$ is taken as lower bound and respectively $M_{x_0}(i_1, j_m)$ as upper bound.

- Heuristic $H4$: we keep the same bounds as heuristic $H2$ for position and velocity. Nevertheless, for the energy, as lower bound, we compute the minimum value on the last row and as upper bound, the maximum value on the first row:

  - Lower bounds: $\min_{j \in J} M_{x_0}(i_n, j), \min_{j \in J} M_{x_2}(i_1, j), M_{x_3}(i_1, j_1)$.
  - Upper bounds: $\max_{j \in J} M_{x_0}(i_1, j), \max_{j \in J} M_{x_2}(i_n, j), M_{x_3}(i_n, j_m)$.

The bounds computed exactly or heuristically at line 7 of Algorithm 1 are used at line 8 to decide whether or not it is necessary to look for the global minimum inside the discrete sub-box $X^{(i)}$. More precisely, if the condition of line 8 holds, then the global minimum may occur in $X^{(i)}$ which justifies its insertion into the list in line 16. In this case, we choose to evaluate the objective function at the midpoint of $X^{(i)}$ in order to get a feasible solution. Note that we use the symbol [.] in line 10 to denote the closest integer value of a real number. If the obtained trial solution produce a better solution than the best one found so far by the algorithm, we update the current solution (line 13). The process already described is iterated until convergence, this is ensured when the whole list $\mathcal{L}$ is empty (line 3).

In [8], we also proposed a refinement process that aims to improve the accuracy and the quality of the solution obtained by Algorithm 1. This is done by choosing the best attained

approximation of the global optimum as a starting point and then successively increasing P and decreasing s until the wished precision is reached. The steps of this process are given in Algorithm 2.

Algorithm 2 A refinement process to the expected continuous global optimum

1: Solve problem (4) using parameter P quite low and parameter s quite high. In [8] we used at this first step $P = 5$ and $s = 10$.
2: Take the solution previously found (at step 1 or step 4 depending on the context), and define a zone Z around this solution with a reduced range. Generally, we use a range about 40 amps around the solution but always inside $[-150, 150]$.
3: Find a more precise solution of problem (4) in Z by increasing P and decreasing s.
4: Return to step 2, until the wished precision is reached; generally $P = 10$ and $s = 1$.

In [8], `BBA_SC` was coded in `MatLab`. For this paper, `BBA_SC` is coded in `Fortran90`; `gfortran` from the `gnu` library is used. To have a discussion about the choice of `Fortran90` instead of `C++` and comparisons with distinct implementation of this code, see the technical report [10].

To compare the `MatLab` and `Fortran90` codes, we take the same first examples as in [8, 15] where an electrical vehicle with an initial state $(x_1^0, x_2^0, x_3^0) = (0, 0, 0)$ has to reach a final position of 100 meters (m) in 10 seconds (s). Thus, using `BBA_SC` on a 2.33 GHz Intel Xeon (quad core) processor with 16GB of RAM and Linux Operating System, we obtain the following solutions provided in Table 2. Note that E_{min} corresponds to the minimum energy returned by both implementations of `BBA_SC`.

Instance	*iref* (*amps*)	E_{min} (J)	*range* (*amps*)	*CPU* (s)			*Iter.*	
				Fortran	*MatLab*		*Fortran*	*MatLab*
					no mcc	mcc		
$P = 5$ $s = 10$	(150, 90, 30, -30, -110)	23,272	300	0.05	3.30	3.20	23,374	23,069
$P = 10$ $s = 10$	(150, 140, 110, 70, 30, 20, -10, -30, -90, -130)	22,813	40	0.89	40.31	40.97	264,374	264,406
$P = 10$ $s = 5$	(150, 145, 105, 65, 20, 20, 0, -20, -80, -135)	22,698	20	2.37	121.88	115.01	702,154	702,222
$P = 10$ $s = 1$	(150, 147, 104, 63, 21, 20, 0, -21, -80, -136)	22,648	4	4.30	250.77	243.32	1,405,661	1,410,307

Table 2: Table of refined solutions: free final velocity.

In Table 2, we remark a significant gain in CPU-time using `Fortran90` compared to the `Matlab` code. Indeed, the CPU-time is divided by **38** in the worst case and **58** in the best case, although the number of iterations is of the same order. The last refined solution (last row of Table 2) allows to obtain a gain of 2.69% on the quality of the global optimum for $7.56s$ of additional CPU-time. The gain is about a factor of 54 on the cumulative time. In order to speed-up the `MatLab` code, we converted it into a `C` code by using the `mcc complier of MatLab`. However, we did not observe empirical evidence of a gain of performance in terms of CPU-time. Using the `mcc compiler,` the highest gain is about $7s$ and is obtained in the last line of Table 2. This behaviour is mainly due to the structure of the `MatLab` code. In fact, each step of this latter is organized in four stages (extraction, division, analysis, insertion), as previously mentioned. Regarding these steps, the `MatLab` code cannot be vectorized and then no performance benefit can be obtained.

Now, if we try to solve directly problem (4) with $P = 10$ and $s = 1$ and by using heuristic $H3$, the `MatLab` code does not succeed to obtain a solution. However, the `Fortran90` implementation of `BBA_SC` provides the following solution $iref = (150, 140, 110, 70, 30, 20, -10, -40, -70, -140)$, which corresponds to 2,130,045,196 iterations of the Branch and Bound code in about 3 hours and 45 minutes of CPU-time; the corresponding minimum energy is equal to $22{,}772J$ and the final position is equal to $100.08m$. Thus, in the rest of this paper, the implementation in `Fortran90` of the `BBA_SC` code will be always used.

4 Two new heuristics for computing bounds

Some evaluations of `BBA_SC` for a displacement of 100 meters with different conditions on the final velocity and a cycle of time of 10 seconds ($t_f = 10s$) show that the percentage of success on the solution of the heuristics $H1$, $H2$, $H3$ and $H4$ depends on the studied instances. In Table 3, we show this strong dependency. For example, heuristic $H2$ seems very efficient in the case where the final position ($x_3(t_f)$) is fixed to $100m$, the final velocity ($x_2(t_f)$ converted in km/h) is equal to $50km/h$ and the state velocity ($x_2(t)$) is less or equal than $50km/h$ during all the displacement. However, considering the free final velocity case, $H2$ provides solutions which are sometimes not the best ones and may generate a small error about 1.5%.

In this paper, two new heuristics are proposed, they are more robust than the four previous ones as shown in Table 3. These two new heuristics were developed after studying the behavior of the energy, the velocity and the position matrices, see the corresponding curves on Figures 1, 2 and 3.

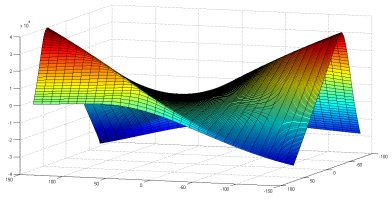

Figure 1: Energy: M_{x_0}

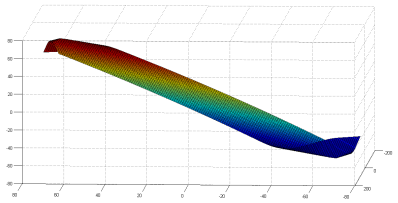

Figure 2: Velocity: M_{x_2}

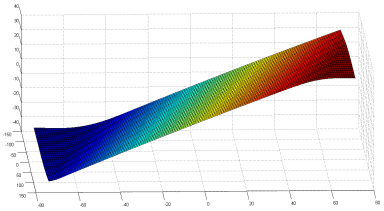

Figure 3: Position: M_{x_3}

First, note that the values of the position in the matrix are in the increasing order. Hence, it is the reason why we keep the same bounds for the position in our four heuristics $H1$-$H4$. Now, considering the energy, note that for `BBA_SC` it is just required to find a lower bound because it is the criterion to be minimized, and looking at Figure 1 which represents the energy matrix, we can see that two concavities appears. Thus, a better heuristic idea is to choose as lower bound the minimum value of the energy obtained only at the four extremal

vertices.

One of the reasons of the "failure" of the heuristics previously developed ($H1$-$H4$) is its vagueness in the computation of the velocity bounds. Indeed, the structure of the velocity presents three parts: a almost linear part and two other ones more complex. Therefore, we need first to explore the whole sub-matrix to find the exact bounds on the velocity as it is provided with the new heuristic $H5$. By using the preliminary numerical results, we remark that we can look for bounds on the velocity only on some parts of the matrix M_{x_2} and this yields the new heuristic $H6$. The two new heuristics $H5$ and $H6$ are given below. Recall that the integer sets of indices $I = \{i_1, ..., i_n\}$ and $J = \{j_1, ..., j_m\}$ correspond to the possible values of the velocity at the end of the previous sample time and the possible values of iref.

- Heuristic $H5$: for the position, we keep the same bounds as for heuristic $H1$. As lower bound for the energy, we compute the minimum value of the 4 elements located on the vertices of the induced sub-matrix. To compute the lower (resp. upper) bound for the velocity, we consider the minimum (resp. the maximum) over all the sub-matrix.

  - Lower bounds: $\min\{M_{x_0}(i_1, j_1),\ M_{x_0}(i_1, j_m),\ M_{x_0}(i_n, j_1),\ M_{x_0}(i_n, j_m)\}$; $\min\limits_{i\in I, j\in J} M_{x_2}(i, j)$; $M_{x_3}(i_1, j_1)$.
  - Upper bounds: $\max\limits_{i\in I, j\in J} M_{x_2}(i, j)$; $M_{x_3}(i_n, j_m)$.

- Heuristic $H6$: we keep the same bounds as heuristic $H5$ for the position and the energy. However, for the velocity, we compute as lower bound the minimum value on the first column and as upper bound, the value corresponding to the last row and last column of the induced sub-matrix.

  - Lower bounds: $\min\{M_{x_0}(i_1, j_1),\ M_{x_0}(i_1, j_m),\ M_{x_0}(i_n, j_1),\ M_{x_0}(i_n, j_m)\}$; $\min\limits_{i\in I} M_{x_2}(i, j_1)$; $M_{x_3}(i_1, j_1)$.
  - Upper bounds: $M_{x_2}(i_n, j_m)$; $M_{x_3}(i_n, j_m)$.

In order to test the efficiency of these two new heuristics, we performed 101 tests in which the final position varies from $20m$ to $120m$ by increment of $1m$. In Table 3, we present the percentage of success of different heuristics. $H5$ and $H6$ are completely reliable on these tests. Indeed, these heuristics are able to give the same solution as using the exact method EM independently of the studied instance without generating any error contrary to the other heuristics.

$Profile$	$H1$	$H2$	$H3$	$H4$	$H5$	$H6$
$x_2(t_f)$ free	0%	67%	97%	100%	100%	100%
$x_2(t_f) = 0$	33%	98%	94%	94%	100%	100%
$x_2(t_f) = 50$ $x_2(t) \leq 50$	76%	100%	77%	78%	100%	100%

Table 3: Percentage of success of different heuristics with respect to the studied instance.

Moreover, to validate the use of heuristics $H5$ and $H6$, we compare their performances in terms of CPU time, as given in Table 4.

$Profile$	$H5$	$H6$	EM
$x_2(t_f)$ free	$0.5s$	$0.13s$	$2.04s$
$x_2(t_f) = 0$	$0.22s$	$0.05s$	$1.03s$
$x_2(t_f) = 50$ $x_2(t) \leq 50$	$0.24s$	$0.05s$	$1.04s$

Table 4: Performance of heuristics $H5$ and $H6$ in CPU-time.

From Table 4, heuristic $H6$ appears to be faster than $H5$. In fact, considering for example the null final velocity case, comparing to EM and using $H6$, the average CPU-time is divided by **20**. However, it is only divided by **4** when $H5$ is used. This important gain in time for $H6$ compared to EM is also clear from the scheme given in Figure 4.

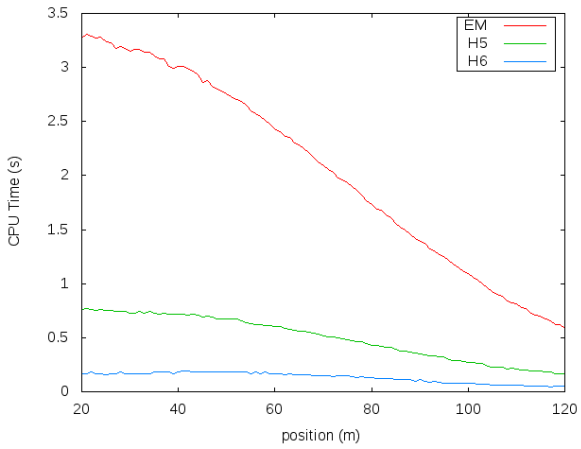

Figure 4: 101 test problems with a free final velocity. In ordinate, there is the corresponding CPU-time, and in abscise the final position. The parameters of `BBA_SC` are $P = 5$; $s = 10$.

For the case of $20m$, we remark that the exact method EM is very slow comparing to $H5$ and $H6$. This is due to the fact that the set of possible solutions is very large which is not the case when the final distance is $120m$. Thus, heuristic $H6$ will be used throughout the rest of the paper.

5 Algorithm for long travels

To have a good approximation of the global optimum of problem (4), we have to discretize the interval of time $[0, t_f]$ into small samples. If the final time t_f is large, the finite set of possible solutions becomes huge. In order to avoid this problem, the idea is to use variable steps of discretization instead of constant ones. More precisely, we choose to concentrate on the acceleration (starting) and deceleration (stopping) phases and discretize them using small steps of $2s$, while the remainder part are handled with uniform steps of $10s$. This idea is summarized in Algorithm 3. The efficiency and reliability of this algorithm are assessed on three test cases, as presented in Subsections 5.1-5.3.

Algorithm 3 Management of long travels

1: Discretize the sample of time into P equal intervals.
2: Solve problem (4) using `BBA_SC` with s fixed and quite big.
3: Define reduced zones around the so-obtained solution.
4: Discretize more precisely some important parts of the sample of time: the beginning and the end for example. This provides distinct steps of time and that increases P.
5: Solve problem (4) with this discretized set (in step 4) and by decreasing s.
6: Iterate step 5 by decreasing s.

5.1 Case without constraint on velocity

We evaluate our method for a displacement of 1,000m and a final time $t_f = 70s$: $(x_1(0), x_2(0), x_3(0)) = (0, 0, 0)$; $(x_1(t_f), x_2(t_f), x_3(t_f)) \in \mathcal{T} = \mathbb{R} \times \mathbb{R} \times \{1{,}000\}$. Testing our algorithm over sampling intervals of $10s$, we obtain the exact solution $iref = (140, 10, 30,\ 20, 30, 20, -20)$. By decomposing the beginning and the end in steps of $2s$, we have the equivalent solution (140, 140, 140, 140, 140, $10, 30, 20, 30, 20, -20, -20, -20, -20, -20$) (the two values 140 and -20 are reused 5 times at the beginning and at the end yielding 15 variables, see the above algorithm). Spreading it on a range of 20 amps, we generate a new box $IREF = [130, 150] \times [130, 150] \times [130, 150] \times [130, 150] \times [130, 150] \times [0, 20] \times [20, 40] \times [10, 30] \times [20, 40] \times [10, 30] \times [-30, -10] \times [-30, -10] \times [-30, -10] \times [-30, -10] \times [-30, -10]$ where the solution will be searched, see Figure 5.

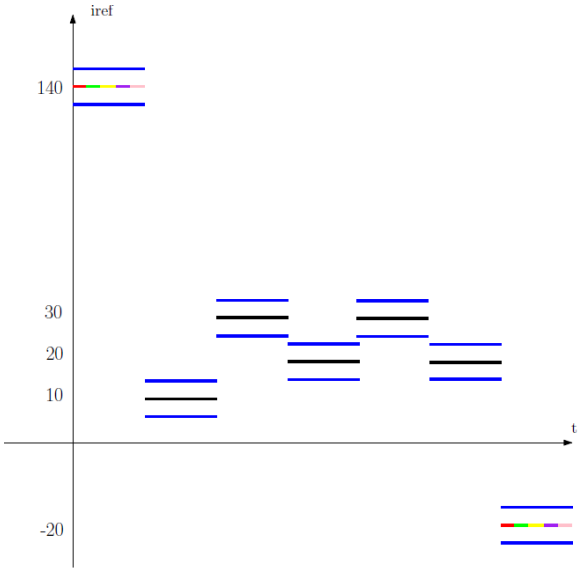

Figure 5: Refinement by decomposition in subdomains. In black, this is the reference solution, in blue the domain where the next global optimum will be searched and in color at the beginning and at the end of the sample, we add new variables by subdividing again these two areas. On this example this yields a global optimization problem with 15 variables.

By repeating this process (iteration of the algorithm) with different range to refine the solutions, we obtain the results presented in Table 5.

With the improved solutions, we obtain a gain of 1.45% on the quality of the optimum for 3,476 seconds of additional CPU-time. The curves on Figure 6 are drawn using the latest

Instance	*iref*	E_{min} (J)	$x_3(t_f)$ (m)	$x_2(t_f)$ (km/h)	*CPU* (s)	*range* $(amps)$	*Iter.*
$P = 7$ $s = 10$	(140, 10, 30, 20, 30, 20, -20)	206,945	1,000.20	12.92	11.69	300	3,354,323
$P = 15$ $s = 10$	(140, 130, 130, 130, 150, 20, 20, 20, 30, 20, -10, -10, -10, -30, -30)	204,987	1,000.18	13.61	4.25	20	683,753
$P = 15$ $s = 1$	(141, 128, 128, 128, 149, 18, 21, 21, 28, 22, -10, -8, -8, -29, -32)	203,934	1,000.02	14.87	3,529	4	627,133,367

Table 5: Refined solutions: free final velocity.

refined solution (last row of Table 5). The computation of this solution, by simulating it using $RK4$, provides an energy $E_{min} = 207{,}352J$ and a position $x_3(t_f) = 1{,}006.6m$. The error is about 1.65% on the energy and 0.65% on the position.

The current x_1 takes its maximum values in the first ten seconds to ensure the starting phase, decreases as far as reaching a steady-state value around 21 amps and ends with negative values which corresponds to a phase of recovery. In fact, the decreasing of the curve of the energy at the end of the cycle (last $10s$) corresponds to this phase. The current x_1 remains trapped around $iref$ with respect to the tolerance Δ. The values of the control u switch many times between -1 and +1; this is due to the fact that the current varies too quickly in the motor (about 3 amps every 10^{-3} seconds).

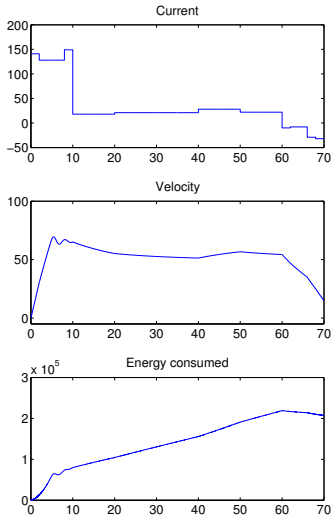

Figure 6: Case with a free final velocity: $P = 15$; $s = 1$; $x_3(t_f) = 1{,}000m$.

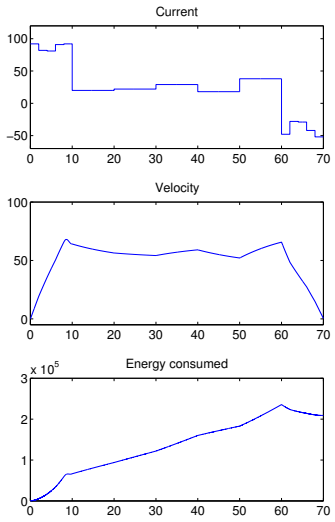

Figure 7: Case with a final velocity equals to 0: $P = 15$; $s = 1$; $x_3(t_f) = 1{,}000m$; $x_2(t_f) = 0km/h$.

5.2 Case with null final velocity

The final velocity on Figure 6 is not null. Nevertheless, our method with successive refinement can take into account this final state constraint. To compare solutions, we simulate last

instances with a constraint on the final velocity; i.e., a displacement of 1,000m and a cycle of time of 70 seconds ($t_f = 70s$) with a null final velocity: $(x_1(0), x_2(0), x_3(0)) = (0, 0, 0)$; $(x_1(t_f), x_2(t_f), x_3(t_f)) \in \mathcal{T} = \mathbb{R} \times \{0\} \times \{1{,}000\}$. The corresponding refined solutions are presented in Table 6.

Instance	*iref*	E_{min} (J)	$x_3(t_f)$ (m)	$x_2(t_f)$ (km/h)	CPU (s)	*range* *(amps)*	*Iter.*
$P = 7$ $s = 10$	(80, 20, 20, 30, 20, 40, -40)	211,515	1,000.42	0.23	3.72	300	1,094,326
$P = 15$ $s = 10$	(90, 80, 80, 90, 90, 20 20, 30, 20, 40, -50, -30, -30, -40, -50)	209,715	1,000.20	0.31	3.88	20	588,992
$P = 15$ $s = 1$	(92, 82, 81, 91, 92, 20 22, 29, 18, 38, -48, -28, -29, -42, -52)	208,632	1,000.01	0.44	3,604	4	660,735,488

Table 6: Refined solutions with null final velocity.

The last refined solution is obtained using a total CPU-time about 3,522s. Using the numerical integrator $RK4$, the calculation error is about 0.2% for the energy ($E_{min} = 209{,}059J$) and 0.03% for the position ($x_3(t_f) = 999.74m$). This solution is plotted on Figure 7.

Because the vehicle starts and ends with a velocity which is equal to zero, it necessarily goes through a phase of deceleration corresponding to the recovery phase of electrical energy. The current is at a high value at the beginning and ends with a law value to be able to stop the vehicle (null final velocity).

5.3 Case with constraints on the velocity state variable and on the final velocity

If we test our algorithm with constraints on the velocity state $(x_2(t)(\text{in } km/h) \leq 50km/h)$[1] and on the final velocity $(x_2(t_f) = 50km/h)$, our simulations show that it is not possible to perform the displacement of $1km$ within a final time equals to $70s$. That is why, we limited the final distance to $800m$. The solutions obtained using successive runs and refinement of `BBA_SC` code extended for long travels, are reported in Table 7 and drawn in Figure 8.

Instance	*iref*	E_{min} (J)	$x_3(t_f)$ (m)	$x_2(t_f)$ (km/h)	CPU (s)	*range* *(amps)*	*Iter.*
$P = 7$ $s = 10$	(50, 20, 10, 20, 20, 20, 40)	167,522	803.22	63.59	0.06	300	13,793
$P = 15$ $s = 10$	(50, 50, 60, 60, 50, 20, 20, 20,10, 10, 50, 30, 30, 30, 30)	150,905	801.61	50.25	0.22	20	36,523
$P = 15$ $s = 1$	(52, 52, 61, 58, 48, 18, 19, 21, 12, 10, 48, 28, 29, 29, 28)	148,432	800.02	49.53	3,155	4	632,196,840

Table 7: Refined solutions: constraints on the velocity state $(x_2(t) \leq 50\ km/h)$ and the final velocity $(x_2(t_f) = 50km/h)$.

To validate the solution obtained in Table 7, we use directly $RK4$. This provides an energy $E_{min} = 148{,}169J$, a position $x_3(t_f) = 798.82m$ and a final velocity $x_2(t_f) = 49.46km/h$. The error is about 0.18% for the energy, 0.15% for the position and $\pm 0.07km/h$ for the velocity.

[1] where $x_2(t)$ is converted into km/h by multiplying by the constant: $\frac{3.6r}{K_r}$

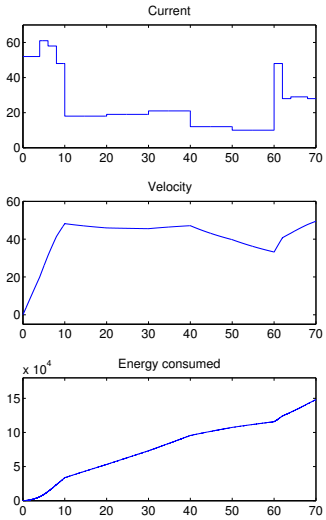

Figure 8: $Case$: $P = 15$; $s = 1$; $x_2(t_f) = 800m$; $x_2(t) \leq 50km/h$; $x_2(t_f) = 50km/h$.

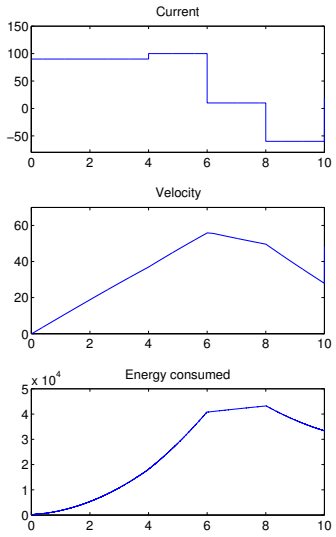

Figure 9: Free final velocity case with a slope of $+3°$, $\beta = 0.3$.

The curve of the current on Figure 8 is at its maximum value at the beginning, then decreases to maintain the velocity at its desired value and finally, increases in order to make the vehicle able to finish with a velocity of $50km/h$. In this profile, there is no recovery of the electrical energy which can be seen on the energy curve.

Based on the numerical tests already done, we can remark that:

i) Using the algorithm described at the beginning of this section, we obtain an important gain on the quality of the global optimum which needs a very long CPU-time.

ii) The refined solutions obtained with the different studied cases have an identical behavior at the beginning of the interval of time; i.e., x_1 takes its maximum value early in the cycle to ensure the starting phase. This is not the case at the end of the cycle. Indeed, considering the free or null final velocity case, the curve of current x_1 ends with negative currents which correspond to a recovery phase. However, considering a non-zero final velocity and a limit on the speed, x_1 increases in the last ten seconds to maintain the final velocity and in this case, there is no recovery phase.

iii) It is very expensive in terms of CPU-time to reach a precision of 1 amps on the solution ($s = 1$); it needs about 3,000 seconds. However, the quality of the solution is not improved so far: the gain is always less than 1% on the value of the energy. Thus, it seems sufficient to provide solutions using $s = 10$.

6 Algorithm to manage slopes

Thus, an important operational point is to take into account slopes in the displacement that a vehicle has to perform. However, the model of the problem has to include it in the differential

equation of the velocity as follows:

$$\dot{x}_2(t) = \frac{1}{J}\left(K_m x_1(t) - \frac{r}{K_r}\left(MgK_f + \frac{1}{2}\rho S C_x \left(\frac{x_2(t)r}{K_r}\right)^2 + Mg\ sin(\frac{\pi}{180}\theta)\right)\right), \quad (6)$$

where θ refers to the angle of the slope in degree. This is the same model as (1) but $-\frac{1}{J}\frac{r}{K_r}Mg\ sin(\frac{\pi}{180}\theta)$ is added to penalize the velocity if some slopes in degree have to be performed.

The difficulty to solve problem (1) taking into account equation (6) using our algorithm comes essentially from the computation of bounds. In fact, we are not able to determine exactly the optimal time t_* when the vehicle reaches the position where the slope changes. That is why, we just try to determine the sample time in which the distance when a change of slope occurs, is reached ($50m$ in our following example). Once this distance is found, we start from the beginning of this sample which will be again subdivided using smallest step (of $0.5s$ in this paper for the numerical tests) in order to obtain a greater accuracy. These computations require additional pre-processing for computing matrices over a sample time of length $0.5s$ which have to take into account the different slopes encountered in our travel.

6.1 Algorithm

The different steps described above are summarized in the following algorithm. Using these steps and including them at line 11 of Algorithm 1, we will be able to manage travels including slopes.

Algorithm 4 Management of slopes

1: Search the sample time in which the vehicle reaches the position of the slope change.
2: Return to the beginning of this sample time.
3: Reinitialize the state variables by taking their values in the previous sample.
4: Subdivide it with a step of $0.5s$.
5: Research the sample time of length $0.5m$ (included in the one found in step 1) in which the vehicle reaches the position when the slope changes.

6.2 Numerical tests

We test our algorithm for a displacement of $100m$ including a slope of $+3°$ after $50m$ using the "classical" method for computing bounds. We notice that we may have some solutions which present important stalls. For example, considering a free final velocity case, we obtain the solution $iref = (150, 150, -150, 100, -70)$ producing an consumed energy of $24{,}338J$. In order to avoid such situations, the idea will be to add a constraint on the acceleration. To perform this, we consider also the acceleration A:

$$A = \frac{r}{K_r}\ \dot{x}_2.$$

This acceleration is measured in m/s^2. It can be converted into "g" (the gravity of the Earth) if we prefer to compare it to the acceleration of gravity which is $1g = 9.81m/s^2$. Indeed, $1g$ means that an object of $1kg$ accelerates, in a crash for example, with $9.81m/s$ per second (we say $9.81m.s^{-2}$) or also, we add every second about $35.3km/h$ to our initial velocity, i.e., we reach the velocity of $105.9km/h$ after the first three seconds, which can be unbearable by the person inside the electrical vehicle. Thus, the objective here is to be able to control

and determine the acceleration that is supported by the vehicle and its driver. Based on the above remarks, we can add this new constraint:

$$|A| \leq \beta \; g, \tag{7}$$

where $\beta \in]0, 1]$.

To take into account this constraint, we need to compute a new matrix concerning the acceleration. It should be noticed that the smallest is β, the smoothest is the solution, as presented in Table 8. In Table 8, we present the solutions of problem (6) by taking into

$coefficient$	$iref$	E_{min} (J)	$x_3(t_f)$ (m)	$x_2(t_f)$ (km/h)	CPU (s)	$Iter.$
$\beta = 1$	(150, 150, -150, 100, -70)	24,338	100.74	16.62	49.93	1,527,119
$\beta = 0.5$	(150, 150, -120, 70, -90)	25,223	101.40	11.55	50.50	1,527,197
$\beta = 0.4$	(100, 80, 130, -50, -90)	25,989	100.09	17.08	50.56	1,527,268
$\beta = 0.3$	(90, 90, 100, 10, -60)	32,601	100.39	30.69	50.89	1,527,337

Table 8: Influence of the choice of β on the quality of solutions.

account the new constraint (7). These numerical results correspond to a distance of $100m$ including a slope of $+3°$ (at $50m$) with a free final velocity, see Figure 9.

Although the choice of $\beta = 1$ gives the minimum value for the energy consumed by an electrical vehicle compared to other choices, we will not consider this solution for the reasons previously discussed. It should be noticed that this solution is also the absolute one which is obtained when constraint (7) is removed. However, the maximum acceleration (resp. minimum) corresponding to the choice of $\beta = 0.3$ is about $3m.s^{-2}$ (resp. $-3m.s^{-2}$) which is equivalent to reach a speed of $34.77km/h$ in 3 seconds which can be accepted in a driving located at the limit of "quiet" on dry ground.

For the case of a null final velocity, the numerical simulations show that $\beta = 0.5$ is the best choice. However, when there is a constraint on the velocity state and on the final velocity, we find that constraint (7) can be omitted without affecting the quality of the solution.

Testing our algorithm for a displacement of $100m$ including a slope of $+3°$ (at $50m$) with different constraints on the state velocity and on the final speed, we find the results presented in Table 9 and Figure 9.

$Instance$	$iref$	E_{min} (J)	$x_3(t_f)$ (m)	$x_2(t_f)$ (km/h)	CPU (s)	$Iter.$
$x_3(t_f) = 100$ $x_2(t_f)$ free	(90, 90, 100, 10, -60)	32,601	100.39	30.69	50.89	1,527,337
$x_3(t_f) = 100$ $x_2(t_f) = 0$	(140, 140, 40, -100, -100)	31,283	100.32	0.37	2.91	14,667
$x_3(t_f) = 100$ $x_2(t_f) = 50$ $x_2(t) \leq 50$	(130, 60, 30, 50, 40)	45,875	100.28	49.87	14.09	247,256

Table 9: Solutions for a displacement of $100m$ including a slope of $+3°$ at $50m$.

Considering the free final velocity case, we note that the minimum energy consumption is about 28.6% greater than in the case without including slope. The calculation of the solution (first row), by simulating using $RK4$ directly, provides an energy $E_{min} = 33{,}367J$ and a position $x_3(t_f) = 97.44m$. The error is about 2.29% for the energy and 3.03% for the position. Despite the presence of a positive slope, we remark that there is a recovery phase. In fact, the curve of the energy decreases at the end of the cycle (the last two seconds).

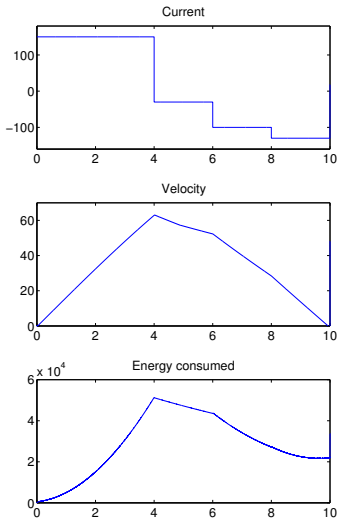

Figure 10: Null final velocity case with a slope of $-3°$.

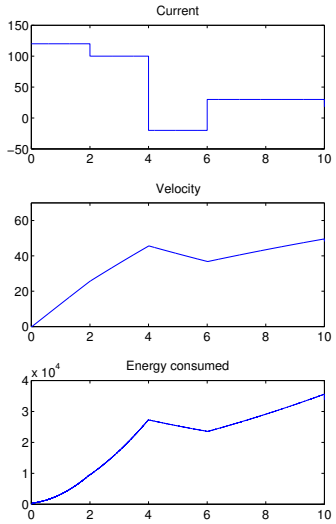

Figure 11: Non-zero final velocity and a limit on the speed with a slope of $-3°$.

Our algorithm can also take into account negative slopes as presented in Table 10.

Instance	*iref*	E_{min} (J)	$x_3(t_f)$ (m)	$x_2(t_f)$ (km/h)	*CPU* (s)	*Iter.*
$x_3(t_f) = 100$ $x_2(t_f)$ free	(90, 90, 100, -10, -60)	27,754	100.09	33.76	59.49	1,819,387
$x_3(t_f) = 100$ $x_2(t_f) = 0$	(150, 150, -20, -100, -130)	23,743	100.30	-0.77	3.27	15,162
$x_3(t_f) = 100$ $x_2(t_f) = 50$ $x_2(t) \leq 50$	(120, 100, -20, 30, 30)	35,589	100.32	49.79	17.12	313,566

Table 10: Solutions for a displacement of $100m$ including a slope of $-3°$ at $50m$.

On Figure 10, we represent the obtained solution for the null final velocity case with a slope of $-3°$. Using the solution $iref = (150, 150, -20, -100, -130)$, the electrical vehicle reaches the position $50m$ presenting the slope $-3°$ in the third sample of time; i.e., between $4s$ and $6s$, which is clear from the curve of the velocity. Indeed, this sample corresponds to the beginning of the phase of deceleration corresponding to the recovery phase of electrical energy.

We consider now the case of the non-zero velocity and a limit on the speed (last row in Table 10). The minimum energy consumption for this case is about 28.9% smaller than in the case with a slope of $+3°$. By validating this solution using $RK4$, this provides an energy of $E_{min} = 35{,}601J$ and a final position $x_3(t_f) = 97.99m$. The error is about 0.03% for the energy and 2.37% for the position. The current takes its maximum at the beginning of the cycle and ends with constant value, 30 amps in the last four seconds. This allows to maintain the velocity at its desired value. In this case, there is no recovery phase, as presented in Figure 11, despite the presence of a negative slope. This is due to the fact that the final velocity must be equal to $50km/h$.

7 Discussions

In Sections 5 and 6, extensions of the algorithm BBA_SC are provided to deal respectively with long travels and with displacements including slopes. A question remains: is it possible to solve problems with long travels which include slopes? In fact, this is quite simple to associate the two algorithms presented in Sections 5 and 6 to answer positively to this question. Thus, as an example, consider a displacement of 1,000m where from 0 to 300m the travel is flat, then we have a slope of $+3°$ until 600m and from 600m to 1,000m the travel becomes again flat. On this example the initial conditions are $(x_1(0), x_2(0), x_3(0)) = (0, 0, 0)$ and the final ones are $x_3(t_f)$=1,000m with $t_f = 70s$; note that we consider two cases: with a free final velocity and with a null final velocity ($x_2(t_f) = 0$). In Table 11, the solutions are reported.

Instance	*iref*	E_{min} (J)	$x_3(t_f)$ (m)	$x_2(t_f)$ (km/h)	*CPU* (s)	*Iter.*
$x_2(t_f)$ free	(90, 40, 20, 30, 30, 50, -20)	243,332	1,000.74	20.46	15,347	561,296,291
$x_2(t_f) = 0$	(90, 30, 30, 20, 40, 50, -40)	249,658	1,000.65	0.29	912	28,677,933

Table 11: Solutions for a displacement of 1,000m including two different slopes.

Considering the solution of the free final velocity case and by simulating it using $RK4$, the error is about 2.09% for the energy ($E_{min} = 238{,}345J$), 4.03% for the position ($x_3(t_f) = 961.89m$) and $\pm 0.23km/h$ for the velocity ($x_2(t_f) = 20.69km/h$). This solution is presented in Figure 12. Figure 13 is the result of the simulation of the solution with the case of null final velocity and by using RK4. This provides an energy of $E_{min} = 243{,}942J$, a position $x_3(t_f) = 967.99m$ and a final velocity $x_2(t_f) = 0.39km/h$. The error is about 2.34% for the energy, 3.37% for the position and $\pm 0.1km/h$ for the velocity. Remark that our method

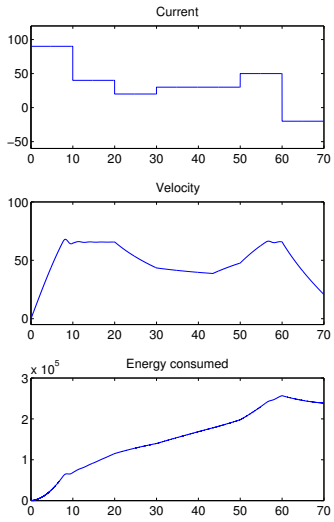

Figure 12: Free final velocity case.

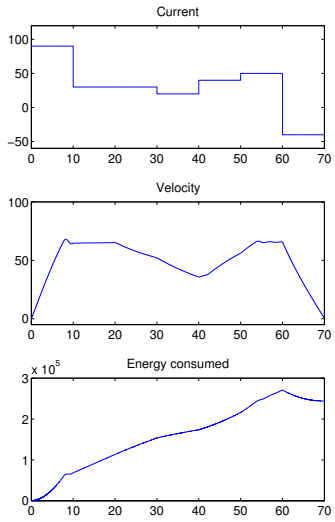

Figure 13: Null final velocity case.

can provide some errors which can reach in the worst case 4%. This could be improved by considering much more thin discretization steps to find more precisely the time where the changes of slope occur.

To conclude, we show in this paper that `BBA_SC` algorithm can be extended to deal with long travels including slopes. Some new more efficient heuristics for computing bounds, are provided and two new algorithms are validated on numerous numerical test problems. The efficiency of such a new implementation in `Fortran90` of these Branch and Bound algorithms are shown. This provides some efficient way to solve operational problems dealing with the minimization of the energy consumed by an electrical vehicle.

References

[1] Bernard, J., Delprat, S., Guerra, T., and Buechi, F. Fuel cell hybrid vehicles: Global optimization based on optimal control theory. *International Review of Electrical Engineering 1* (2006), 352–362.

[2] Hellstrm, E., Ivarsson, M., slund, J., and Nielsen, L. Look-ahead control for heavy trucks to minimize trip time and fuel consumption. *Control Engineering Practice 17*, 2 (2009), 245 – 254.

[3] Henrion, D., Daafouz, J., and Claeys, M. Optimal switching control design for polynomial systems: an LMI approach. In *Proceedings of the 52nd IEEE Conference on Decision and Control, CDC 2013, December 10-13, 2013, Firenze, Italy* (2013), pp. 1349–1354.

[4] Howlett, P. The optimal control of a train. *Annals of Operations Research 98*, 1 (2000), 65–87.

[5] Howlett, P., and Leizarowitz, A. Optimal strategies for vehicle control problems with finite control sets. *Dynamics of Continuous Discrete and Impulsive Systems (DCDIS), Series B. 8* (2001), 41–70.

[6] Lasserre, J.-B. *Moments, positive polynomials and their applications*. Imperial College Press Optimization Series. Imperial College Press, London, 2010.

[7] Merakeb, A., and Messine, F. Toward global minimum solutions for the problem of the energy consumption of an electrical vehicle. In *Proceedings of the global optimization workshop, GOW-TOGO* (2010), vol. 10, pp. 85–88.

[8] Merakeb, A., Messine, F., and Aidène, M. A branch and bound algorithm for minimizing the energy consumption of an electrical vehicle. *4OR 12*, 3 (2014), 261–283.

[9] Musardo, C., Rizzoni, G., Guezennec, Y., and Staccia, B. A-ecms: An adaptive algorithm for hybrid electric vehicle energy management. *European Journal of Control 11*, 4 (2005), 509 – 524.

[10] Omheni, R. Methods and algorithms for the minimization of the energy consumed by an electrical vehicle. Tech. rep., University of Limoges, 2011.

[11] Pudney, P. *Optimal energy management for solar-powered cars*. University of South Australia, 2000.

[12] Pudney, P., and Howlett, P. Critical speed control of a solar car. *Optimization and Engineering 3*, 2 (2002), 97–107.

[13] Sager, S., Bock, H. G., and Diehl, M. The integer approximation error in mixed-integer optimal control. *Mathematical Programming 133*, 1 (2012), 1–23.

[14] Sager, S., Bock, H. G., and Reinelt, G. Direct methods with maximal lower bound for mixed-integer optimal control problems. *Mathematical Programming 118*, 1 (2009), 109–149.

[15] Sager, S., Claeys, M., and Messine, F. Efficient upper and lower bounds for global mixed-integer optimal control. *Journal of Global Optimization 61*, 4 (2015), 721–743.

[16] Sciarretta, A., and Guzzella, L. Control of hybrid electric vehicles. *IEEE Control systems 27*, 2 (2007), 60–70.