Time-optimal velocity planning by a bound-tightening technique

Time-optimal velocity planning by a bound-tightening technique

Federico Cabassi, Luca Consolini, Marco Locatelli *

November 11, 2021

Abstract

Range reduction techniques often considerably enhance the performance of algorithmic approaches for the solution of nonconvex problems. In this paper we propose a range reduction technique for a class of optimization problems with some special structured constraints. The procedure explores and updates the values associated to the nodes of a suitably defined graph. Convergence of the procedure and some efficiency issues, in particular related to the order into which the nodes of the graph are explored. The proposed technique is applied to solve problems arising from a relevant practical application, namely velocity planning along a given trajectory. The computational experiments show the efficiency of the procedure and its ability of returning solutions within times much lower than those of nonlinear solvers and compatible with real-time applications.

KEYWORDS: Range Reduction, Velocity Planning, Minimum-Time Problems, Local Search.

1 Introduction

In this paper we study classes of optimization problems where some or all the constraints have a special structure. Such structure will enable us to develop a quite efficient range reduction procedure, i.e., a procedure which allows reducing the range of the variables. Range reductions are of primary importance in the field of non convex optimization. Indeed, many convex underestimators for the objective and/or constraint functions are defined over a box and the smaller the box, the sharper is the underestimator. They are classified into *feasibility-based* (the range of the variables is reduced without eliminating feasible solutions) and *optimality-based* (the range of the variables is reduced without eliminating optimal solutions). In literature different general range reduction strategies have been proposed (see, e.g., [3, 7, 15, 17, 18, 22, 25, 29, 35]). Range reduction for special structured problems can be found, e.g., in [4, 9, 10, 21, 26, 31]. In [30] a theory of range reduction tools, based on Lagrange multipliers, is presented and many earlier results are re-derived from this theory. More theoretical results can be found in [7, 8]. It is also worthwhile to recall that `BARON`, one of the most successful software for global optimization, strongly relies on reduction techniques.

The range reduction proposed in this paper is a feasibility-based one and is accomplished by solving a problem over a suitably defined graph, depending on the variables and the special structured constraints. Besides enabling range reductions, in some cases the proposed technique will also allow to solve very efficiently problems arising from relevant applications. In this paper we will discuss one such applications, namely planning the velocity of a vehicle along a given trajectory, but other applications are possible like, e.g., the numerical solution of dynamic programming equations (see, e.g., Appendix A in [2]), or the computation of velocity profiles for robotic manipulators (see, e.g., [24]). The paper is structured as follows. In Section 2 we describe the class of

*Dipartimento di Ingegneria e Architettura, Università di Parma, Parco Area delle Scienze, 181/A, 43124 Parma (Italy), e-mails: federico.cabassi@studenti.unipr.it, luca.consolini@unipr.it, marco.locatelli@unipr.it

optimization problems we will deal with throughout the paper. In Section 3 we will prove some theoretical results and develop the algorithm for the range reduction. In Section 4 we will describe the previously mentioned application in the field of velocity planning over a given trajectory. We will see that, after discretization, the planning problem can be reformulated as an optimization problem within the class (1). Depending on the constraints that we will impose during the planning, we will see that the optimization problem can either be solved exactly or we are able to return a lower bound and a feasible solution for the problem. In the latter case, in Section 6 we will further exploit the special structure of the optimization problem in order to refine the feasible solution. In Section 7 we will perform computational experiments which show that the refinement procedure described in Section 6 is able to return results which are either optimal or very close to optimality. Moreover, we will show that the computing times are much lower than those attained by nonlinear solvers and scale quite well as we decrease the discretization step and, thus, increase the size of the optimization problem. The computing times are compatible with real-time applications.

2 Description of the problem class

Before describing the problem class we introduce some notation. We denote by $N = \{1, \dots, n\}$ and $M = \{1, \dots, m\}$ the index sets of the variables and of the special structured constraints, respectively. Set $B_{\mathbf{u}}$ denotes the n-dimensional box $[\mathbf{0}, \mathbf{u}]$. Set X is a subset of $B_{\mathbf{u}}$. For some $J \subseteq N$ and $\mathbf{x} \in B_{\mathbf{u}}$, we denote by $\mathbf{x}_J$ the sub-vector of $\mathbf{x}$ whose components are those within the index subset J. The class of optimization problems we consider is the following

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ & \mathbf{x} \in X \cap C. \end{aligned} \tag{1}$$

Set C is defined as follows

$$C = \{\mathbf{x} \in \mathbb{R}^n \ : \ x_{i_j} \leq g_j(\mathbf{x}_{K_j}), \ j \in M\}, \tag{2}$$

where for each $j \in M$:

- $K_j \subseteq N \setminus \{i_j\}$, i.e., g_j depends on a subset of variables which does not include variable x_{i_j};
- g_j is a non negative function over $B_{\mathbf{u}}$, i.e.:

$$g_j(\mathbf{x}) \geq 0 \quad \forall \mathbf{x} \in B_{\mathbf{u}}; \tag{3}$$

- g_j is monotonic non decreasing over $B_{\mathbf{u}}$, i.e.,

$$\mathbf{x}, \mathbf{y} \in B_{\mathbf{u}}, \quad \mathbf{x} \geq \mathbf{y} \quad \Rightarrow \quad g_j(\mathbf{x}) \geq g_j(\mathbf{y}). \tag{4}$$

In some cases we also ask for the following additional assumption, also known as superiority condition:

$$\forall \mathbf{x} \in B_{\mathbf{u}} \ \ \forall j \in M, \ \ \forall k \in K_j : \quad g_j(\mathbf{x}_{K_j}) \geq x_k. \tag{5}$$

The objective function f and the constraint functions g_j, $j \in M$, are required to be continuous but do not need be differentiable.

In Section 3 we will prove that set C has the following important property

$$\exists \bar{\mathbf{x}} \in C \cap B_{\mathbf{u}} \ : \ \mathbf{x} \leq \bar{\mathbf{x}} \quad \forall \mathbf{x} \in C \cap B_{\mathbf{u}}, \tag{6}$$

i.e., set C admits a component-wise maximum. The existence proof of $\bar{\mathbf{x}}$ will be constructive, i.e., we will provide a procedure that returns $\bar{\mathbf{x}}$ after a finite time (if (5) holds) or converges to $\bar{\mathbf{x}}$ (if (5) does not hold). The detection of $\bar{\mathbf{x}}$ has some relevant implications, namely:

- we are able to perform a range reduction of the variables, since the initial box $B_{\mathbf{u}}$ can be reduced to the box $B_{\bar{\mathbf{x}}}$;
- if f is a monotonic non increasing function, i.e.,

$$\mathbf{x} \geq \mathbf{y} \quad \Rightarrow \quad f(\mathbf{x}) \leq f(\mathbf{y}), \tag{7}$$

  then $f(\bar{\mathbf{x}})$ is a readily available lower bound for the optimal value of (1);
- if f is monotonic non increasing and $X = B_{\mathbf{u}}$, then $\bar{\mathbf{x}}$ is, in fact, the optimal solution of (1).

Remark 2.1 *In case the feasible region is a subset of a region*

$$C' = \{\mathbf{x} \in \mathbb{R}^n \ : \ x_{i_j} \geq g_j(\mathbf{x}_{K_j}), \ j \in M'\},$$

where with respect to the definition (2) of C the inequalities are reversed, we could similarly prove the existence of a component-wise minimum for C' and, thus, update the lower limits of the variables over the feasible region. The discussion of this case is omitted since it is analogous to the discussion about the region C.

3 Existence and detection of a component-wise maximum within the set C

Existence of $\bar{\mathbf{x}}$ as defined in (6) can be proved by lattice theory. Consider the lattice $(B_{\mathbf{u}}, \geq)$ and the operator $\mathcal{M} : \mathbb{R}^n \to \mathbb{R}^n$ defined as follows

$$[\mathcal{M}(\mathbf{x})]_k = \min\{x_k, \min\{g_j(\mathbf{x}_{K_j}) \ \forall j \ : \ i_j = k\}\} \quad k = 1, \ldots, n.$$

Then, Knaster-Tarski Fixpoint Theorem (see, e.g., Theorem 2.35 in [14]) states that the greatest fixed point of $\mathcal{M}$ over $B_{\mathbf{u}}$ is equivalent to $\bar{\mathbf{x}}$ as defined in (6). In this section we provide a constructive proof of the existence of $\bar{\mathbf{x}}$, i.e., we introduce Algorithm 1 and then prove that, under suitable assumptions, it allows to detect $\bar{\mathbf{x}}$. The algorithm updates two vectors:

- $\boldsymbol{\gamma}$, an n-dimensional vector whose components will be proved to be upper bounds for the values of the variables over $C \cap B_{\mathbf{u}}$ in Lemma 3.1;
- $\boldsymbol{\mu}$, a m-dimensional vector whose components are the values of the constraint functions at the current $\boldsymbol{\gamma}$.

The vector $\boldsymbol{\gamma}$ is initialized with the vector $\mathbf{u}$ of upper bounds for the variables, while each component μ_j, $j \in M$, is initialized with $g_j(\mathbf{u}_{K_j})$. The set U of updated nodes is initialized to M, while the iteration counter k is set equal to 1. Within the outer `While` cycle (lines 9-29) the set E of explored nodes is initialized to the empty set. Then, an inner `While` (lines 12-22) cycle is run where at each iteration: (i) an element $j \in U \setminus E$ is selected such that μ_j is lowest within $U \setminus E$ (line 13); (ii) j is moved from U to E (line 14); (iii) if $\gamma_{i_j} > \mu_j = g_j(\boldsymbol{\gamma}_{K_j})$ (i.e., constraint j is violated by the current $\boldsymbol{\gamma}$), then γ_{i_j} is updated and set equal to μ_j and, consequently, also all values μ_h such that $i_j \in K_h$ are updated, and h is moved into U (lines 15-21). After the inner `While` cycle, $\boldsymbol{\xi}^k$ is set equal to the current $\boldsymbol{\gamma}$ and ρ_k measures the overall violation of the constraints at $\boldsymbol{\xi}^k$ (lines 23-24). The algorithm stops as soon as $U = \emptyset$, i.e., no update has been performed during the last iteration of the outer `While` cycle, in which case $\boldsymbol{\xi}^k \in C$, or $\rho_k \leq \varepsilon$, i.e., the constraint violation falls below a given tolerance ε (the stopping condition is evaluated at lines 25-28).

We first prove that each vector $\boldsymbol{\gamma}$ is an upper bound for the feasible points in $C \cap B_{\mathbf{u}}$.

Lemma 3.1 *For any $\mathbf{x} \in C \cap B_{\mathbf{u}}$, $\mathbf{x} \leq \boldsymbol{\gamma}$ always holds.*

Data: $\mathbf{u} \in \mathbb{R}^n_+$, $\varepsilon > 0$;
foreach $s \in N$ **do**
  Set $\gamma_s = u_s$;
end
foreach $r \in M$ **do**
  Set $\mu_r = g_r(\mathbf{u}_{K_r})$;
end
Set $U = M$, $k = 1$;
Set `STOP`=false;
while `STOP` *is false* **do**
  Set `STOP`=true;
  Set $E = \emptyset$;
  while $U \setminus E \neq \emptyset$ **do**
    Let $j \in \arg\min_{h \in U \setminus E}\{\mu_h\}$;
    Set $U = U \setminus \{j\}$ and $E = E \cup \{j\}$;
    if $\gamma_{i_j} > \mu_j$ **then**
      Set $\gamma_{i_j} = \mu_j$;
      foreach $h \in M \ : \ i_j \in K_h$ **do**
        Set $U = U \cup \{h\}$;
        Set $\mu_h = g_h(\boldsymbol{\gamma}_{\boldsymbol{K_h}})$;
      end
    end
  end
  Set $\boldsymbol{\xi}^k = \boldsymbol{\gamma}$;
  Set $\rho_k = \max_{j \in M} \max\{0, \gamma_{i_j} - g_j(\boldsymbol{\gamma}_{K_j})\}$;
  if $U \neq \emptyset$ `AND` $\rho_k > \varepsilon$ **then**
    Set `STOP`=false;
    Set $k = k + 1$;
  end
end

Algorithm 1: Algorithm for the detection of the component-wise maximum within set C.

Proof. The proof is done by induction. The statement is obviously true when $\boldsymbol{\gamma}$ is initialized to $\mathbf{u}$. Now, let us assume that it is true before an update of $\boldsymbol{\gamma}$ and we prove it is still true after the update. Vector $\boldsymbol{\gamma}$ is updated only if for some $j \in M$ we have that $\gamma_{i_j} > g_j(\boldsymbol{\gamma}_{K_j})$, in which case γ_{i_j} is set equal to $g_j(\boldsymbol{\gamma}_{K_j})$. Then, the result follows from

$$\gamma_{i_j} = g_j(\boldsymbol{\gamma}_{K_j}) \geq g_j(\mathbf{x}_{K_j}) \geq x_{i_j} \quad \forall \mathbf{x} \in C \cap B_{\mathbf{u}},$$

where the first inequality is a consequence of the inductive assumption (which implies $\boldsymbol{\gamma}_{K_j} \geq \mathbf{x}_{K_j}$) and of (4), while the second inequality follows from the definition of C. □

Observation 3.1 *If (3)-(5) are fulfilled, then Algorithm 1 stops after a single iteration and* $\boldsymbol{\xi}^1 = \bar{\mathbf{x}}$.

Proof. As soon as $j \in M$ enters the set E of explored nodes, it is also removed from U, the set of updated nodes. We only need to prove that under the given assumptions, j never enters U again. Let us denote by $\bar{\boldsymbol{\gamma}}$ the value of $\boldsymbol{\gamma}$ when j enters E. We first make the following remark. When $j \in M$ is explored, i.e., it enters set E, γ_{i_j} is possibly updated and set equal to $\mu_j = g_j(\bar{\boldsymbol{\gamma}}_{K_j})$. Consequently, also the value μ_h is updated for each h such that $i_j \in K_h$. In view of (5)

$$\mu_h = g_h(\boldsymbol{\gamma}_{K_h}) \geq \gamma_{i_j} = \mu_j,$$

i.e., each updated value μ_h is not smaller than μ_j. By induction it also follows that

$$\mu_h \geq \mu_j \quad \forall h \in M \setminus E,\ j \in E. \tag{8}$$

Now, let us assume by contradiction that at some point $j \in E$ enters U again. That happens as soon as we explore some $r \in M \setminus E$ and update the value $\bar{\gamma}_{i_r}$ such that $i_r \in K_j$. Before the update $\bar{\gamma}_{i_r} > \mu_r$ holds. Moreover, in view of $i_r \in K_j$ and (5)

$$\mu_j = g_j(\bar{\boldsymbol{\gamma}}_{K_j}) \geq \bar{\gamma}_{i_r} > \mu_r \geq \mu_j,$$

where the last inequality follows from (8). This is a contradiction. Thus, at the end of the first iteration

$$\xi^1_{i_j} \leq g_j(\boldsymbol{\xi}^1) \quad \forall j \in M,$$

i.e., $\boldsymbol{\xi}^1 \in C \cap B_{\mathbf{u}}$. Moreover, in view of Lemma 3.1, $\boldsymbol{\xi}^1 \geq \mathbf{x}\ \forall \mathbf{x} \in C$, from which we can conclude that $\boldsymbol{\xi}^1 = \bar{\mathbf{x}}$. □

When (5) does not hold, the algorithm stops after a finite number of iterations for any $\varepsilon > 0$. The following observation gives an upper bound for the number of iterations of the outer `While` cycle when (3)-(4) hold.

Observation 3.2 *If (3)-(4) are fulfilled, then Algorithm 1 stops after a number of iterations not larger than*

$$\frac{\sum_{i=1}^{n} u_i}{\varepsilon}.$$

Proof. We introduce the following measure of infeasibility at the end of the k-th iteration of the outer `While` cycle

$$\delta_k = \sum_{l \in N} \max_{j\ :\ i_j \equiv l} \max\{0, \xi^k_{i_j} - g_j(\boldsymbol{\xi}^k_{K_j})\} \geq \rho_k \geq \varepsilon.$$

Then, the values of the variables at the end of the $k+1$-th iteration are reduced as follows with respect to the values at the end of the k-th iteration

$$\sum_{l \in N} \xi^{k+1}_l \leq \sum_{l \in N} \xi^k_l - \delta_k.$$

With respect to the initial iteration the following holds

$$\sum_{l\in N} \xi_l^{k+1} \leq \sum_{l\in N} u_l - \sum_{s=1}^{k} \delta_s \leq \sum_{l\in N} u_l - k\varepsilon.$$

Since in view of (3) $\sum_{l\in N} \xi_l^{k+1} \geq 0$ holds, then

$$\sum_{l\in N} u_l - k\varepsilon \geq 0,$$

from which the result follows. □

The following corollary states convergence of Algorithm 1 when $\varepsilon = 0$.

Corollary 3.1 *If $\varepsilon = 0$, then*

$$\boldsymbol{\xi}^k \to \bar{\mathbf{x}}.$$

Proof. In view of Lemma 3.1, $\boldsymbol{\xi}^k \geq \mathbf{x}$ $\forall \mathbf{x} \in C$, $\forall k$. Since the sequence of vectors $\{\boldsymbol{\xi}^k\}$ is non increasing,

$$\boldsymbol{\xi}^k \to \bar{\boldsymbol{\xi}} \geq \mathbf{x} \quad \forall \mathbf{x} \in C.$$

Moreover, using the same reasoning as in the proof of Observation 3.2, it can be proved that

$$\rho_k \to 0.$$

Since ρ_k measures the infeasibility of $\boldsymbol{\xi}^k$ we can conclude, in view of the continuity of the functions g_j, $j \in M$, that $\bar{\boldsymbol{\xi}} \in C$ and, thus, that $\bar{\boldsymbol{\xi}} = \bar{\mathbf{x}}$. □

Next, we prove an observation about the computational cost of Algorithm 1.

Observation 3.3 *The computational cost for each iteration of the outer* `While` *cycle in Algorithm 1 is $O\left(|M|^2\right)$.*

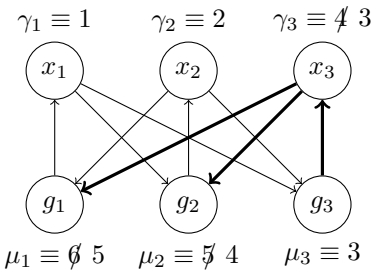
Proof. Each iteration of the outer `While` cycle is made up by an inner `While` cycle. At each iteration of the inner cycle the computational cost is determined by the search for the minimum μ_h, $h \in M \setminus E$, and by the possible update of $|K_j| \leq |M|$ values. Since both require at most $|M|$ operations, and since the number of iterations of the inner cycle is not larger than $|M|$, the result follows. □

In fact, the cost per iteration can often be reduced, as suggested by the following remark.

Remark 3.1 *The order into which the nodes are explored does not affect the convergence of Algorithm 1 but may affect the overall number of iterations. In particular, it has been experimentally observed that the rule employed in Algorithm 1, where the order into which the elements of M are explored is dynamically determined by searching for the minimum of the current values μ_h, $h \in M \setminus E$, allows for a low number of iterations. On the other hand, the search for the minimum is costly and alternative orders could be considered. One possibility is to fix the order into which the elements of M are explored* before *entering the inner* `While` *cycle, for instance by ordering them in nondecreasing order with respect to their μ values. In that case, the computational cost per iteration of the outer* `While` *cycle becomes*

$$O\left(\max\left\{|M|\log(|M|), \sum_{j\in M} K_j\right\}\right).$$

Figure 1: A case where condition (5) is fulfilled



A further improvement is the following. After the first iterations we do not expect large variations of the μ values. Therefore, after a fixed number of iterations the order is not changed any more. In that case the cost per iteration becomes

$$O\left(\sum_{j\in M} K_j\right).$$

Finally, we remark that in some applications it is possible to determine an order in advance. This will be the case for the application discussed in Section 4.

The algorithm can be viewed as an algorithm over an oriented graph $G = (V, A)$ with one node, indexed by i, for each variable x_i, and one node, indexed by $n+j$, for each constraint $x_{i_j} \leq g_j(\mathbf{x}_{K_j})$, i.e.,

$$V = \{1, \ldots, n, n+1, \ldots, n+m\},$$

while for $s \in \{1, \ldots, n\}$, $r \in \{n+1, \ldots, n+m\}$:

$$\begin{cases} (s, r) \in A & \text{if } s \in K_{r-n} \\ (r, s) \in A & \text{if } s \equiv i_{r-n}. \end{cases}$$

A value γ_i is associated to each node $i = 1, \ldots, n$, while a value μ_j is associated to each node $n + j$, $j = 1, \ldots, m$. The following example illustrates how the algorithm works over two simple instances, one where condition (5) is fulfilled, the other where it is not.

Example 3.1 *Let:*

$$C = \left\{(x_1, x_2, x_3) \in \mathbb{R}^3_+ \ : \ x_1 \leq g_1(x_2, x_3),\ x_2 \leq g_2(x_1, x_3),\ x_3 \leq g_3(x_1, x_2)\right\}. \tag{9}$$

Let $u_1 = 1$, $u_2 = 2$, and $u_3 = 4$. We first consider the case where

$$g_1(x_2, x_3) = x_2 + x_3, \quad g_2(x_1, x_3) = x_1 + x_3, \quad g_3(x_1, x_2) = x_1 + x_2,$$

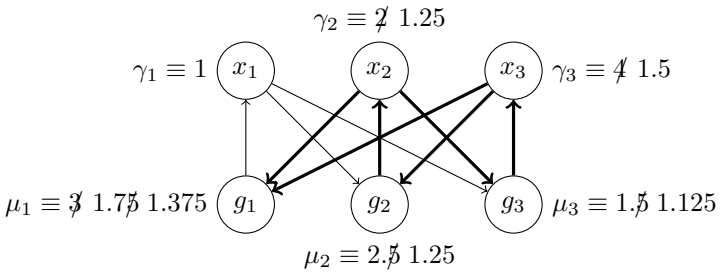
for which condition (5) is fulfilled. The sequence of updates (corresponding to thick arcs) performed by Algorithm 1 is illustrated in Figure 1. We remark that the arcs which determine the updates do not form a cycle, so that the algorithm stops after a single iteration with the detection of the point $\bar{\mathbf{x}} = (1\ 2\ 3)$. Next, we consider the case where

$$g_1(x_2, x_3) = \frac{x_2 + x_3}{2}, \quad g_2(x_1, x_3) = \frac{x_1 + x_3}{2}, \quad g_3(x_1, x_2) = \frac{x_1 + x_2}{2},$$

for which condition (5) is not fulfilled. The sequence of updates (corresponding to thick arcs) performed by Algorithm 1 is illustrated in Figure 2. In this case the arcs for a cycle so that the algorithm needs to go through further iterations. In fact, it can easily be seen that the algorithm converges to $\bar{\mathbf{x}} = (1\ 1\ 1)$.

Note that when (5) holds it can be seen that the algorithm is equivalent to a generalized shortest path algorithm (see [1]).

Figure 2: A case where condition (5) is not fulfilled



4 Velocity planning along a given trajectory

A common way to address the task of motion planning for an autonomous vehicle is to subdivide it into two subtasks, namely a (geometric) path planning followed by a minimum-time velocity planning on the planned path (see, e.g., [19]). In this paper we discuss how to tackle the second subtask, i.e., we assume that the path that joins the initial and the final configuration is given and we search for the time-optimal speed law that satisfies some kinematic and dynamic constraints. In the literature about this problem some works address the problem in time domain (see, e.g., [11, 16, 23, 28, 32, 34]). In some other works, such as [5, 20, 27, 33], the speed law is represented as a function v of the arc-length position s and not as a function of time. The latter approach, adopted in this paper, is interesting since it allows for a reformulation of the velocity planning problem as a convex one. In what follows we describe in detail the problem tackled in this paper.

The path followed by a vehicle is represented by the image set of a smooth function $\gamma : [0, s_f] \to \mathbb{R}^2$ such that $(\forall \lambda \in [0, s_f])\ \|\gamma'(\lambda)\| = 1$. The initial and final configuration are $\gamma(0)$ and $\gamma(s_f)$, respectively. We aim at minimizing the overall transfer time while satisfying some kinematic and dynamic requirements. The vehicle position as a function of time is represented by $\lambda : [0, t_f] \to [0, s_f]$, a differentiable monotone increasing function. The vehicle velocity at position s is represented by a function $v : [0, s_f] \to [0, +\infty)$ such that, $\forall t \in [0, t_f]$, $\dot{\lambda}(t) = v(\lambda(t))$. The position of the vehicle as a function of time is given by $x : [0, t_f] \to \mathbb{R}^2$, $x(t) = \gamma(\lambda(t))$, while the velocity and acceleration are given by

$$\dot{x}(t) = \gamma'(\lambda(t))v(\lambda(t)),\ \ddot{x}(t) = a_T(t)\gamma'(\lambda(t)) + a_N(t)\gamma'^{\perp}(\lambda(t)),$$

where $a_T(t) = v'(\lambda(t))v(\lambda(t))$, $a_N(t) = k(\lambda(t))v(\lambda(t))^2$ are, respectively, the tangential and normal components of acceleration. Here $k : [0, s_f] \to \mathbb{R}$ is the scalar curvature, defined as $k(s) = \langle \gamma''(s), \gamma'(s)^{\perp} \rangle$. After making the change of variable $w = v^2$, we consider the following optimization problem

$$\min_{w \in C^1([0,s_f],\mathbb{R})} \int_0^{s_f} w^{-1/2}(s)ds \tag{10a}$$

$$w(0) = 0,\ w(s_f) = 0, \tag{10b}$$

$$0 < w(s) \leq \bar{v}^2, \qquad s \in (0, s_f), \tag{10c}$$

$$|w'(s)| \leq A_T, \qquad s \in [0, s_f], \tag{10d}$$

$$|k(s)|w(s) \leq A_N, \qquad s \in [0, s_f], \tag{10e}$$

$$|w'(s) - w'(t)| \leq A_J|s - t|, \qquad s, t \in [0, s_f]. \tag{10f}$$

The objective function (10a) is the overall transfer time, to be minimized. Constraints (10b) set the initial and final velocity of the vehicle to 0. Constraint (10c) imposes an upper bound for the maximum velocity. Constraints (10d) and (10e) bound the tangential and normal accelerations,

respectively. The last constraint (10f) enforces a smoother velocity profile by imposing a Lipschitz condition on the first derivative.

The problem above can be discretized as follows

$$\min_{w} \sum_{i=1}^{n-1} \frac{2h}{\sqrt{w_i} + \sqrt{w_{i+1}}} \tag{11a}$$

$$w_1 = u_1 = 0,\ w_n = u_n = 0, \tag{11b}$$

$$0 \le w_i \le u_i = \min\left\{\bar{v}^2, \frac{A_N}{|k(hi)|}\right\} \qquad i = 2, \dots, n-1, \tag{11c}$$

$$w_{i+1} - w_i \le hA_T, \qquad i = 1, \dots, n-1, \tag{11d}$$

$$w_i - w_{i+1} \le hA_T, \qquad i = 1, \dots, n-1, \tag{11e}$$

$$w_{i+1} + w_{i-1} - 2w_i \le A_J h^2, \qquad i = 2, \dots, n-1 \tag{11f}$$

$$2w_i - w_{i+1} - w_{i-1} \le A_J h^2, \qquad i = 2, \dots, n-1, \tag{11g}$$

where $h = \frac{s_f}{n-1}$ is the discretization step. In [6] it has been proved that the optimal solution of (11) followed by quadric interpolation can be employed to approximate with an arbitrary precision $\varepsilon > 0$ the optimal solution of problem (10) in a polynomial time with respect to $\frac{1}{\varepsilon}$.

In what follows we discuss an efficient way to solve the discretized problem (11). We first discuss in Section 5 the solutions of problems including a subset of the constraints. Next, in Section 6 we address the solution of the complete problem.

Constraints (11f)-(11g) will be called in what follows Positive Acceleration Rate (PAR) and Negative Acceleration Rate (NAR) constraints, respectively. Moreover, for the sake of simplicity, in what follows we will often refer to the values w_i as velocity values although they are squared velocity values.

5 The cases with acceleration and NAR constraints

As pointed out at the end of Remark 3.1, in some cases we can fix in advance the order into which the nodes are explored, thus reducing the complexity of the algorithm. In this section we show that this is true for problem (11) if we only consider, besides constraints (11b) and (11c): (i) the acceleration constraints (11d)-(11e); (ii) the NAR constraints (11g); (iii) the acceleration and NAR constraints.

5.1 Acceleration constraints

The case with the acceleration constraints (11d)-(11e) is the simplest one. These can be written as follows

$$w_i \le g_{i-1}(w_{i+1}) = w_{i+1} + hA_T \qquad i = 2, \dots, n-1$$

$$w_{i+1} \le g_{2n-3-i}(w_i) = w_i + hA_T \quad i = 1, \dots, n-2,$$

so that the right-hand side functions g_j, $j \in \{1, \dots, 2n-4\}$, satisfy (5). In this case we do not need to select at each iteration the node in $U \setminus E$ with the lowest μ_h value. We can simply explore the nodes in the order $g_1, g_2, \dots, g_{2n-4}$. This gives rise to Algorithm 2, whose complexity is $O(n)$ and whose correctness has been proved in [12, 13].

Data: $\mathbf{u} \in \mathbb{R}^n_+$;
$w_1 = u_1,\ w_n = u_n$;
foreach $i \in \{1, \ldots, (n-1)\}$ **do**
  $w_{i+1} = \min\{u_{i+1}, w_i + hA_T\}$;
end
foreach $i \in \{(n-1), \ldots, 1\}$ **do**
  $w_i = \min\{w_i, w_{i+1} + hA_T\}$;
end

Algorithm 2: Algorithm `SolveAcc` for the solution of the problem with the acceleration constraints

5.2 NAR constraints

Now we consider the case with the NAR constraints (11g) alone, besides (11b) and (11c). Let $\delta = \frac{h^2 A_J}{2}$. For each $\ell, k \in \{1, \ldots, n\}$, $\ell < k$, we denote by

$$P_{\ell,k}(x) = -\delta(x-\ell)(x-k) + u_\ell \frac{k-x}{k-\ell} + u_k \frac{x-\ell}{k-\ell},$$

the (concave) parabola with curvature $-\delta$ and interpolating the upper bound profile at $x = \ell$ and $x = k$. Note that this parabola defines the values w_j, $j \in \{\ell, \ldots, k\}$, in case all NAR constraints

$$w_j \le \frac{w_{j-1} + w_{j+1}}{2} + \delta, \quad j = \ell + 1, \ldots, k-1,$$

are active, and, moreover, $w_\ell = u_\ell$, $w_k = u_k$. The solution of the problem with the NAR constraints is given, in closed form, in the following observation.

Observation 5.1 *The solution of the problem with the NAR constraints is*

$$w_1^* = w_n^* = 0, \quad w_i^* = \min\left\{u_i, \min_{\ell,k\ :\ \ell<i<k} P_{\ell,k}(i)\right\}, \quad i = 2, \ldots, n-1. \tag{12}$$

Proof. Let $\bar{\mathbf{w}}$ be the optimal solution of the problem. We would like to prove that $\bar{\mathbf{w}} = \mathbf{w}^*$. We do that by first proving that $\bar{w}_i \ge w_i^*$ for each $i \in \{1, \ldots, n\}$, and then that $\bar{w}_i \le w_i^*$ for each $i \in \{1, \ldots, n\}$.

Proof of $\bar{w}_i \ge w_i^*$. This follows from the observation that for each $j \in \{1, \ldots, n\}$, at the optimal solution $\bar{\mathbf{w}}$ either the upper bound constraint $w_j \le u_j$, or the NAR constraint $w_j \le \frac{w_{j-1}+w_{j+1}}{2} + \delta$ is active. Indeed, if this were not true for some j, we could decrease the objective function value by increasing $\bar{w}_j$, thus contradicting optimality of $\bar{\mathbf{w}}$. If $\bar{w}_i = u_i$, then we can immediately conclude, by definition of w_i^*, that $\bar{w}_i \ge w_i^*$. Otherwise, there exist ℓ_i, k_i, $\ell_i < i < k_i$, such that the NAR constraints are active for all $j \in \{\ell_i + 1, \ldots, k_i - 1\}$, while $\bar{w}_{\ell_i} = u_{\ell_i}$ and $\bar{w}_{k_i} = u_{k_i}$, so that $\bar{w}_i = P_{\ell_i,k_i}(i) \ge w_i^*$ (in the most extreme case $\ell_i = 1$ and $k_i = n$).

Proof of $\bar{w}_i \le w_i^*$. In fact, we prove this by showing that at *any* feasible solution $\mathbf{w}$, and, thus, also at the optimal solution $\bar{\mathbf{w}}$, $w_i \le w_i^*$ holds. By the NAR constraint

$$w_i \le \frac{w_{i-1} + w_{i+1}}{2} + \delta \le \frac{u_{i-1} + u_{i+1}}{2} + \delta = P_{i-1,i+1}(i). \tag{13}$$

Next, we observe that in view of (13)

$$w_{i+1} \le \frac{w_i + w_{i+2}}{2} + \delta \ \Rightarrow\ w_i \le \frac{2}{3}w_{i-1} + \frac{1}{3}w_{i+2} + 2\delta \le \frac{2}{3}u_{i-1} + \frac{1}{3}u_{i+2} + 2\delta = P_{i-1,i+2}(i). \tag{14}$$

Moreover,

$$w_{i+2} \leq \frac{w_{i+1}+w_{i+3}}{2}+\delta, \qquad w_{i+1} \leq \frac{w_i+w_{i+2}}{2}+\delta \quad \Rightarrow \quad w_{i+2} \leq \frac{1}{3}w_i+\frac{2}{3}w_{i+3}+2\delta,$$

which combined with (14) leads to

$$w_i \leq \frac{3}{4}w_{i-1}+\frac{1}{4}w_{i+3}+3\delta \leq \frac{3}{4}u_{i-1}+\frac{1}{4}u_{i+3}+3\delta = P_{i-1,i+3}(i).$$

From the above inequality it also follows that

$$w_{i-1} \leq \frac{w_i+w_{i-2}}{2}+\delta \quad \Rightarrow \quad w_i \leq \frac{3}{5}w_{i-2}+\frac{2}{5}w_{i+3}+6\delta \leq \frac{3}{5}u_{i-2}+\frac{2}{5}u_{i+3}+6\delta \leq P_{i-2,i+3}(i).$$

By iterating all this, we can prove that

$$w_i \leq P_{\ell,k}(i) \quad \forall\ \ell < i < k,$$

so that $w_i \leq w_i^*$ immediately follows from the further obvious observation that $w_i \leq u_i$. □

The solution can be obtained by an efficient algorithm by observing that once we have identified for some i the pair ℓ, k, with $\ell < i < k$ returning the value w_i^* in (12), we are able to fix the values w_j^* for $j \in \{\ell, \dots, k\}$ as follows

$$w_j^* = P_{\ell,k}(j),$$

since the value $w_i^* = P_{\ell,k}(i)$ can be attained only if the NAR constraints between $\ell+1$ and $k-1$ are active, and the upper bound constraints are active at ℓ and k. Thus, we can separately search for the remaining velocity values over the two subintervals $\{1, \dots, \ell\}$ and $\{k, \dots, n\}$. In case $w_i^* = u_i$, we set $k = \ell = i$ and we can still split the search into two subintervals, namely $\{1, \dots, \ell\}$ and $\{k, \dots, n\}$. This gives rise to a recursive procedure described in Algorithm 3.

Data: $\mathbf{u} \in \mathbb{R}^n_+$, $s, t \in \{1, \dots, n\}$ $s \leq t$;
Set $w_s^* = u_s$, $w_t^* = u_t$;
if $s < t-1$ **then**
  Set $i = \left\lfloor \frac{s+t}{2} \right\rfloor$;
  Compute w_i^* through (12);
  if $w_i^* = u_i$ **then**
    Set $\ell^* = k^* = i$;
  end
  else
    Let $(\ell^*, k^*) \in \arg\min_{(\ell,k)\ :\ s \leq \ell < i < k \leq t} P_{\ell,k}(i)$;
    foreach $j \in \{\ell^*, \dots, k^*\}$ **do**
      Set $w_j^* = P_{\ell^*,k^*}(j)$;
    end
  end
  `SolveNegJerk`$(\mathbf{u}, s, \ell^*)$;
  `SolveNegJerk`$(\mathbf{u}, k^*, t)$;
end

Algorithm 3: Algorithm `SolveNegJerk` for the solution of the problem with NAR constraints

In order to solve the problem with the NAR constraints we need to call procedure `SolveNegJerk` with input parameters $\mathbf{u}, 1, n$. The overall number of operations needed to solve the problem are stated in the following observation.

Observation 5.2 *The problem with the NAR constraints can be solved with $O(n^2)$ operations by running the procedure* `SolveNegJerk`$(\mathbf{u}, 1, n)$.

Proof. The first call of `SolveNegJerk`, which is also the unique call to this procedure at level $h = 1$, requires the computation of a minimum over all possible pairs (ℓ, k), with

$$\ell < \left\lfloor \frac{n+1}{2} \right\rfloor, \quad k > \left\lfloor \frac{n+1}{2} \right\rfloor.$$

The number of such pairs is $\left(\frac{n}{2}\right)^2$. In the worst case, after this call we fix only the value w_i^* for $i = \left\lfloor \frac{n+1}{2} \right\rfloor$. In such case each of the two new calls of the procedure `SolveNegJerk` at level $h = 2$ requires the computation of the minimum over a number of pairs $\left(\frac{n}{4}\right)^2$ and, again in the worst case, only two new values w_i^* are fixed. By iterating the above observations, in the worst case at level h:

- the number of calls to `SolveNegJerk` is at most 2^{h-1}, so that the overall number of levels h is at most $\log(n)$;
- the overall number of operations for each call at level h is of order

$$\left(\frac{n^2}{2^{2h}}\right).$$

Consequently, the total number of computations to be performed is bounded from above by

$$\sum_{h=1}^{\log(n)} 2^{h-1} \left(\frac{n^2}{2^{2h}}\right) = O(n^2),$$

as we wanted to prove. □

5.3 Acceleration and NAR constraints

Algorithm 4 is an iterative one for the solution (within predefined tolerances $\varepsilon_1 > 0$, $\varepsilon_2 > 0$ for the acceleration and NAR constraints, respectively) of the problem with acceleration and NAR constraints. By a proof completely analogous to that of Observation 3.2, the algorithm can be proved to terminate after a finite number of iterations. It is important to note that this iterative procedure corresponds to a decomposition of the graph associated to the full problem with acceleration and NAR constraints, where, at each iteration, first the subgraph associated to the acceleration constraints is considered and next the subgraph associated to the NAR constraints is considered. This way we exploit the fact that the problems over the two subgraphs can be solved very efficiently.

We also remark the following. In principle, we could solve the problem with only the NAR constraints by fixing $k_1 = 1$ and $k_2 = n$. However, we observed that it is more efficient to split the interval $[1, n]$ into a number of subintervals, whose extremes are the current minima of the velocity profiles, detected by the procedure `FindMinima`. This way it may happen that the NAR constraints (which are certainly not violated in the interior of the subintervals) are violated at the border of two intervals. If this happens, the intervals are merged into a single one and the minimum at the border is removed through the procedure `RemoveMinima` (in fact, we also remove a minimum if its value falls below its upper bound). The solution is recomputed until no violation occurs.

6 A refinement procedure for the case with acceleration, PAR and NAR constraints

The PAR constraints

$$w_{i+1} + w_{i-1} \leq 2w_i + A_J h^2 \quad i = 2, \ldots, n-1, \tag{15}$$

Data: $\mathbf{u} \in \mathbb{R}^n_+$, $\varepsilon_1 > 0$, $\varepsilon_2 > 0$;
`STOP`=false;
while `STOP` *is false* **do**
  $\mathbf{v}$=`SolveAcc`($\mathbf{u}$);
  $\mathbf{u} = \mathbf{v}$;
  $[TotMin, \mathbf{M}]$=`FindMinima`($\mathbf{u}$);
  $ViolJerk = \max\left\{0, \max_{i=2,\dots,n-1} -(w_{i-1} + w_{i+1} - 2w_i) - h^2 A_J\right\}$;
  while $ViolJerk > \varepsilon_1$ **do**
    foreach $s \in \{1, \dots, (TotMin - 1)\}$ **do**
      $k_1 = \mathbf{M}[s]$;
      $k_2 = \mathbf{M}[s+1]$;
      `SolveNegJerk`($\mathbf{u}, k_1, k_2$);
    end
    $ViolJerk = \max\left\{0, \max_{i=2,\dots,n-1} -(w_{i-1} + w_{i+1} - 2w_i) - h^2 A_J\right\}$;
    if $ViolJerk > \varepsilon_1$ **then**
      $[TotMin, \mathbf{M}]$=`RemoveMinima`($\mathbf{u}, \mathbf{v}, \mathbf{M}, TotMin$);
    end
  end
  $ViolAcc = \max\left\{0, \max_{i=1,\dots,n-1} |w_i - w_{i+1}| - hA_T\right\}$;
  if $ViolAcc \leq \varepsilon_2$ `and` $ViolJerk \leq \varepsilon_1$ **then**
    `STOP`=true;
  end
end

Algorithm 4: Algorithm for the solution of the problem with the acceleration and NAR constraints

Data: $\mathbf{u} \in \mathbb{R}^n_+$;
$TotMin = 2$;
$\mathbf{M}[1] = 1$;
foreach $s \in \{2, \dots, n-1\}$ **do**
  if $u_s \leq u_{s-1}$ `and` $u_s \leq u_{s+1}$ `and` $u_s < \max\{u_{s-1}, u_{s+1}\}$ **then**
    $\mathbf{M}[TotMin] = s$;
    $TotMin = TotMin + 1$;
  end
end
$\mathbf{M}[TotMin] = n$;
return $TotMin, \mathbf{M}$;

Algorithm 5: `FindMinima`

Data: $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n_+$, $\mathbf{M}$, $TotMin$;
foreach $s \in \{2, \dots, (TotMin - 1)\}$ **do**
  if $w_{\mathbf{M}[s]} < u_{\mathbf{M}[s]}$ `or` $-(w_{\mathbf{M}[s]+1} + w_{\mathbf{M}[s]-1} - 2w_{\mathbf{M}[s]}) - h^2 A_J > \varepsilon_1$ **then**
    Remove $\mathbf{M}[s]$ and set $TotMin = TotMin - 1$;
  end
end
return $TotMin, \mathbf{M}$;

Algorithm 6: `RemoveMinima`

do not fulfill the requirements, outlined in the Introduction, for the definition of the sets (2). In particular, in this case as soon as we update the value of w_i, and, consequently the right-hand side of constraint (15), we have multiple choices for the update of the variables w_{i-1} and w_{i+1}. However, the structure of these constraints still allows to develop an efficient approach to solve the problem. The key observation is the following. If we have a sequence of consecutive active PAR constraints at times $t, \dots, t+h$, then for each $r \in \{1, \dots, h+1\}$ we have that

$$w_{t+r} = w_t + \left(w_t - w_{t-1} + \frac{h^2 A_J}{2} \right) r + \frac{h^2 A_J}{2} r^2, \tag{16}$$

i.e., the velocity values lie along a parabola with curvature equal to $\frac{h^2 A_J}{2}$. In order to obtain a velocity profile that satisfies the PAR constraints, we first need to identify, in the solution produced by Algorithm 4, the positions where the PAR constraints are violated, and then remove the violations with the addition of parables. We first make the following observation.

Observation 6.1 *If the input maximum velocity* $\mathbf{u}$ *for Algorithm 4 satisfies all PAR constraints, then the output velocity also satisfies all the PAR constraints.*

Proof. We would like to show that, under the given assumption, at each position t the PAR constraint

$$w_{t-1} + w_{t+1} \leq 2w_t + h^2 A_J, \tag{17}$$

is satisfied by the optimal solution of the problem with acceleration and NAR constraints. In such problem the constraints imposing an upper limit for the values of variable w_t are

$$\begin{aligned} & w_t \leq w_{t+1} + hA_T \\ & w_t \leq w_{t-1} + hA_T \\ & 2w_t \leq w_{t+1} + w_{t-1} + h^2 A_J \\ & w_t \leq u_t. \end{aligned}$$

At least one of these constraints must be active, otherwise w_t could be increased, thus contradicting optimality. Let us assume that $w_t < u_t$. If the active constraint is $2w_t \leq w_{t+1} + w_{t-1} + h^2 A_J$, then constraint (17) is obviously satisfied and is not active. If the active constraint is $w_t \leq w_{t+1} + hA_T$, then we notice that the optimal solution also fulfills the constraint $w_t \geq w_{t-1} - hA_T$, so that

$$w_t \geq \frac{w_{t+1} + w_{t-1}}{2},$$

i.e., (17) is satisfied and not active. The development for the case when the active constraint is $w_t \leq w_{t-1} + hA_T$ is analogous. Thus, we are only left with the case $w_t = u_t$. But in such case

$$w_{t-1} + w_{t+1} \leq u_{t-1} + u_{t+1} \leq 2u_t + h^2 A_J = 2w_t + h^2 A_J,$$

where the second inequality follows from the assumption that $\mathbf{u}$ satisfies the PAR constraints. This concludes the proof. □

Such observation shows that problem (11) can also be formulated as follows

$$\begin{aligned} \min \quad & F(\mathbf{y}) \\ & y_{i-1} + y_{i+1} - 2y_i \leq h^2 A_J \quad i = 2, \dots, n-1 \\ & \mathbf{0} \leq \mathbf{y} \leq \mathbf{u}, \end{aligned}$$

where $\mathbf{u}$ is the initial upper bound vector for the velocities, and the function value $F(\mathbf{y})$ is the one returned by Algorithm 4 when the input velocity is $\mathbf{y}$, i.e., it is the optimal value of the problem

with acceleration and NAR constraints when the upper bound vector for the velocities is $\mathbf{y}$. The proof of Observation 6.1 also shows that at the positions where the PAR constraints are violated (if any), the maximum velocity is attained. After the identification of those positions, we impose over the current velocity profile a series of parables like the one described in (16) with the aim of removing the violation of the PAR constraints. The parables are associated to so called *critical pairs*, whose definition is given below.

Definition 6.1 *Given a velocity profile* $\mathbf{w}$ *the set of* critical pairs *is*

$$S = \left\{ (p, w_p) : \frac{w_{p-1} + w_{p+1}}{2} > w_p + \frac{h^2 A_J}{2} \right\},$$

i.e., the set of points where the PAR constraints are violated. We call the value p *the* critical point *associated with the critical pair* (p, w_p)*. We call* critical sequence *a set of consecutive critical points* $\{p, p+1, \ldots, p+q\}$ *such that either* $p-1$ *or* $p+q+1$ *(or both) are not critical points and the corresponding sequence* $w_p, \ldots, w_{p+q}$ *is monotonic (increasing or decreasing).*

In view of Observation 6.1 the set of critical points is a subset of the points at which the maximum velocity profile violates the PAR constraints. Thus, the number of critical points is usually small. E.g., in case the maximum velocity profile is a piecewise constant or piecewise linear function, set S is a subset of the discontinuity points of the function or of its derivative, respectively. In this case each critical sequence is a singleton. We first discuss this practically relevant case while in Remark 6.1 we will discuss how to deal with the case where the critical sequences are not singleton.

Given a critical pair, we can associate to it two different kinds of parables which go through this pair and whose curvature δ is the one imposed by the PAR constraints, i.e., $\delta = \frac{h^2 A_J}{2}$. These parables are displayed in Figure 3. The first parabola is defined as follows: Given a critical point p, a velocity profile $\mathbf{w}$ and a curvature value δ, we can define the convex parabola whose vertex is the critical pair and whose curvature is δ:

$$\alpha_{\delta,\mathbf{w},p}(x) = w_p + (p - x)^2 \delta. \tag{18}$$

This is the parabola associated to the critical pair if the following condition holds:

$$\alpha_{\delta,\mathbf{w},p}(p-1) < w_{p-1} \quad \text{and} \quad \alpha_{\delta,\mathbf{w},p}(p+1) < w_{p+1}.$$

It produces the largest minimum velocity value at the two points $p-1$ and $p+1$ near the critical one, while respecting the PAR constraints. This case is represented in Figure 3a. Otherwise, if

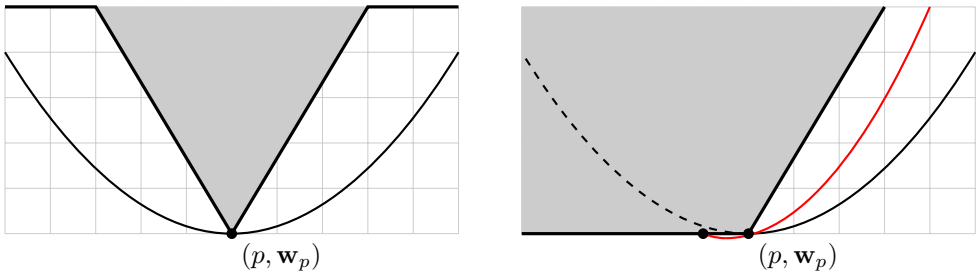
$$\alpha_{\delta,\mathbf{w},p}(p-1) \geq w_{p-1} \tag{19}$$

the parabola associated to the critical pair is shown in Figure 3b and is obtained by imposing that the parabola goes through the points $(p-1, w_{p-1})$ and (p, w_p) and has curvature δ. Similarly, if $\alpha_{\delta,\mathbf{w},p}(p+1) \geq w_{p+1}$ holds, then we consider the parabola going through $(p+1, w_{p+1})$ and (p, w_p) and with curvature δ. Note that the condition

$$\alpha_{\delta,\mathbf{w},p}(p-1) \geq w_{p-1} \quad \text{and} \quad \alpha_{\delta,\mathbf{w},p}(p+1) \geq w_{p+1},$$

can never hold, since otherwise in p there would be no PAR violation. In Algorithm 7 procedure `Parabola`$(p, i, \mathbf{w})$ returns the value at point i of the parabola associated to the critical point p, as defined above.

Once we have detected all critical pairs and we associated a parabola to each of them, we obtain a velocity profile satisfying all PAR constraints by taking the minimum between the current velocity profile and all the parables. In fact, a better profile (i.e., a profile with higher velocities) can be obtained by taking into account interferences between parables. More precisely, it may



(a) (b)

Figure 3: All possible parables associated to a given critical pair $(p, \mathbf{w}_p)$. The grey area represents the epigraph of the current velocity profile and the black parables are the parables whose vertices are the critical pairs. In Figure 3b the red parabola interpolates the velocity profile at $p-1$ and p and has curvature δ.

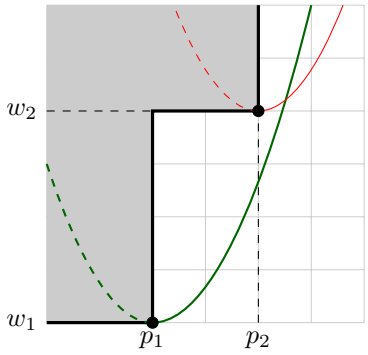


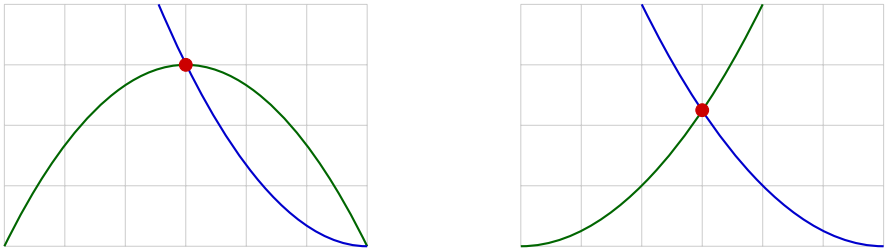
Figure 4: A representation of a simple interference between parables. We can see that the parabola associated with the critical pair (p_1, w_{p_1}) removes the PAR violation over the interval associated with the critical pair (p_2, w_{p_2}). We remark that, for this particular case, the intervals where the PAR constraints are violated are single points.

happen that a parabola associated to a critical pair eliminates the PAR violation at another critical pair. Thus, the parabola associated to the latter can be eliminated. It is possible to see in Figure 4 that if these situations are not identified and parables are not eliminated, the velocity profile is lower in some regions. The detection of these situations can be done by considering pairs of parables. In particular, given a pair of critical points p_1 and p_2, if

$$\alpha_{\delta,\mathbf{w},p_1}(p_2) < \alpha_{\delta,\mathbf{w},p_2}(p_2),$$

the PAR violation at p_2 is eliminated by parabola $\alpha_{\delta,\mathbf{w},p_1}$. In this case we say that $\alpha_{\delta,\mathbf{w},p_1}$ *dominates* $\alpha_{\delta,\mathbf{w},p_2}$ and we eliminate the latter. All couples of parables need to be processed and this should be done in a given order. First, all couples involving the parabola with the lowest vertex should be processed, next the couples involving the parabola with the second lowest vertex among the remaining ones, and so on. In Algorithm 7 the procedure `CheckInterference`(S) checks, starting from the complete set of critical pairs S, if there are some interferences between parables, returning a reduced set S_r of critical pairs and associated parables.

Once the minimum between the velocity profile and the parables has been taken, the resulting velocity profile respects all the PAR constraints but may present some regions in which the acceleration constraints or the NAR constraints are not satisfied. In particular this happens at points where two parables intersect or where a parabola intersects the previous velocity profile



(a) (b)

Figure 5: Configurations of parables that can produce violations on NAR constraints. Figure 5a depicts the case in which a parabola intersects the previous velocity profile, while Figure 5b depicts the case of intersection between two parables. The red points are those where the NAR constraints are violated.

Figure 6: The comparison between the global optimal solution, the red one, and the solution obtained with the critical pairs from Definition (6.1), the blue one. The global optimal solution has an objective function of 127.15 while the local optimal has an objective value of 131.90. This profile presents two critical pairs: $(30, 0.2)$ and $(40, 0.2)$.

(see Figures 5a-5b). In order to obtain a profile that satisfies all the constraints it is necessary to remove these violations by running Algorithm 4 again with the current velocity as the initial maximum velocity profile. Since this profile does not violate the PAR constraints, in view of Observation 6.1 the output velocity will be a feasible solution. Algorithm 7 describes the whole procedure to obtain an admissible velocity profile, given a set of critical pairs.

Data: $\mathbf{w} \in \mathbb{R}^n_+, S, \varepsilon_1 > 0, \varepsilon_2 > 0$;
foreach $i \in \{1, \dots, n\}$ **do**
  $S_r \leftarrow \texttt{CheckInterference}(S)$;
  $w_i = \min_{\forall (p,s) \in S_r} \{w_i, \texttt{Parabola}(p, i, \mathbf{w})\}$;
end
$\mathbf{w} = \texttt{SolveAccelerationNegativeJerk}(\mathbf{w}, \varepsilon_1, \varepsilon_2)$;

Algorithm 7: `ObtainSolution`

Though feasible, there is no guarantee that the solution obtained in this way is also optimal. This can be seen in Figure 6 for a problem with $u_i = 0.2$ for $i \in \{30, \dots, 40\}$, and $u_i = 0.5$ otherwise. The problem has two critical points at $p = 30$ and $p = 40$, respectively. The solution returned by Algorithm 7, represented by the blue curve, has a flat region between the two critical pairs in which the values coincide with the maximum velocity profile, i.e., they are equal to 0.2. But this is not necessarily the best possible choice. Lowering the velocities in this region may allow to increase them outside the region, with an overall gain in terms of objective function, as it can be seen from the optimal solution, represented by the red curve.

In order to find better solutions, it is possible to define a local search strategy, where at each iteration critical pairs are perturbed and new parables are associated to these perturbations. The vertices of the new parables are lower than those of the parables associated to the original critical pairs, but this way faster accelerations and decelerations at the critical pairs are possible since the slope of the new parabola is higher than the slope of the original parabola associated to the critical pair. We still impose that the new parables go through the original critical pairs. Algorithm 9 describes the overall local search procedure (see also the illustration of some iterations in Figure 7). The algorithm is an iterative one. The starting point is the current set of critical pairs. At

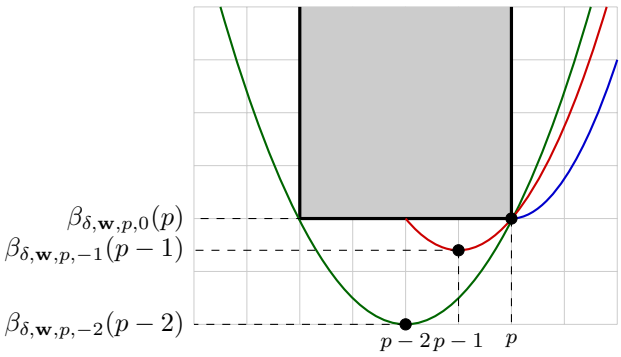


Figure 7: Some iterations of Algorithm 9.

each iteration a neighborhood of the current set of pairs is explored. The original critical pair (p, w_p) is perturbed into $(p-1, w_p - \delta)$ (left perturbation) or $(p+1, w_p - \delta)$ (right perturbation). In both cases the perturbed pair is the vertex of a parabola with curvature δ and vertex at the perturbed pair. Generalizing, we can make a m-perturbation of the original critical pair (p, w_p), with a new pair $(p+m, w_p - m^2\delta)$. Note that m can be either positive (right perturbation) or negative (left perturbation). The new pair is the vertex of a parabola with curvature δ going through the original critical pair. Thus, the parabola associated to the perturbed pair $(p+m, w_p - m^2\delta)$ is

$$\beta_{\delta,\mathbf{w},p,m}(x) = w_p - (p+m)^2\delta + (p+m-x)^2\delta. \tag{20}$$

Data: $S, \mathbf{w} \in \mathbb{R}^n_+, \varepsilon_1 > 0, \varepsilon_2 > 0$;
$\mathbf{d} \in \mathbb{R}^{|S|}$;
$i = 1$;
foreach $(p, \mathbf{w}_p) \in S$ **do**
  $S_l = (S \setminus \{(p, \mathbf{w}_p)\}) \cup (p-1, \beta_{\delta,\mathbf{w},p,-1}(p-1))$;
  $S_r = (S \setminus \{(p, \mathbf{w}_p)\}) \cup (p+1, \beta_{\delta,\mathbf{w},p,1}(p+1))$;
  $\mathbf{w}_l = \texttt{ObtainSolution}(\mathbf{w}, S_l, \varepsilon_1, \varepsilon_2)$;
  $\mathbf{w}_r = \texttt{ObtainSolution}(\mathbf{w}, S_r, \varepsilon_1, \varepsilon_2)$;
  if $f(\mathbf{w}_l) < f(\mathbf{w}_r)$ **then**
    $\mathbf{d}_i = -1$;
  else
    $\mathbf{d}_i = 1$;
  $i = i + 1$;
end
return $\mathbf{d}$;

Algorithm 8: `GetDirections`

Since each point, except for the first and last ones, has a left and right neighbor, each critical pair can be perturbed in both directions. The first step of the local search procedure identifies the perturbation direction for each critical pair. This is achieved by the `GetDirection`$(S, \mathbf{w})$ procedure, whose code is presented in Algorithm 8. For each member (p, w_p) of the set of critical pairs S, the procedure evaluates the solution obtained by perturbing the critical point by -1 and by +1. The best of these solutions identify the direction associated to the critical pair. Thus, the procedure returns the perturbation direction vector $\mathbf{d} \in \{-1, +1\}^{|S|}$.

Given the current set of perturbations m_p for each $(p, w_p) \in S$, i.e., given the current set of perturbed pairs $(p + m_p, w_p - m_p^2\delta)$, procedure `GetCombinations`$(\mathbf{d})$ generates all possible new

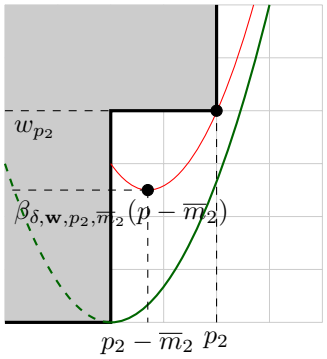


Figure 8: A possible interference that may happen when perturbed parables are compared. In this picture it is possible to see that the green curve dominates the red one since condition (21) is satisfied.

displacements $\mathbf{c}$, where $c_p = d_p$ or 0, so that the overall set of possible displacements C has cardinality $2^{|S|}$. The current set of pairs is perturbed with al possible vectors $\mathbf{c} \in C$ and the procedure `ObtainSolution` is run in order to evaluate the new perturbation by associating to each pair the corresponding parabola β. If no new perturbation improves over the current pair, then the procedure stops, otherwise the current pair is moved to the best among all the evaluated perturbations and the procedure is iterated. Note that, when running the procedure `ObtainSolution`,we need to recalculate interferences between parables: parabola $\beta_{\delta,\mathbf{w},p_1,m_1}(x)$ dominates parabola $\beta_{\delta,\mathbf{w},p_2,m_2}(x)$ if

$$\beta_{\delta,\mathbf{w},p1,m_1}(p_2+m_2) \leq \beta_{\delta,\mathbf{w},p_2,m_2}(p_2+m_2) \text{ and } \beta_{\delta,\mathbf{w},p1,m_1}(p_2) \leq \beta_{\delta,\mathbf{w},p_2,m_2}(p_2), \tag{21}$$

i.e., $\beta_{\delta,\mathbf{w},p_1,m_1}(x)$ is not larger than $\beta_{\delta,\mathbf{w},p_2,m_2}(x)$ both at the vertex of the latter and at the original critical point of the latter. The new interference condition is illustrated in Figure 8.

The best velocity profile returned by the algorithm is denoted by $\mathbf{w}_{best}$.

A further refinement is possible. Until now we have always assumed that each new parabola has its vertex at an integer x-coordinate. In fact, some improvements are possible by relaxing this assumption. A new local search is performed as follows. Given the new pair $(p+\overline{m}, w_p - \overline{m}^2\delta)$, perturbations of this pair are tested, where the x-coordinate of the vertex of the parabola is sampled within the interval $[p+\overline{m}-1, p+\overline{m}+1]$. In other words, rather than performing integer perturbations in $\{-1,+1\}$, small real perturbations are performed, thus performing a more refined local search.

Remark 6.1 *In case the critical sequences are not singleton, we do not associate a parabola to each critical point, rather we select a representative for each critical sequence and associate a parabola only to it. We may need to perform a search among all possible representatives of the critical sequences. This can be done by brute force, i.e., by considering all possible combinations of the representatives. Alternatively, one might perform a neighborhood search where: i) an initial set of representatives is evaluated and selected as initial incumbent solution; ii) the sets of representatives in a neighborhood of the current incumbent are evaluated; iii) the set of the representatives in the neighborhood with the lowest function value is selected as the next incumbent, in case its function value is better than the current one, otherwise the search is stopped and the current incumbent is returned.*

Remark 6.2 *In this paper we considered the case where the initial and final velocity are equal to 0, i.e., $u_1 = u_n = 0$. In this case the problem to be solved is easily seen to be feasible since the*

```
Data: u, w ∈ R^n_+, S, ε1 > 0, ε2 > 0;
s_best = +∞, s = 0;
w_best = w;
m̄, m ∈ R^|S|;
m̄ = 0;
d = GetDirections(S, w);
C ← GetCombinations(d);
while not STOP do
    m = 0;
    STOP = true;
    foreach c ∈ C do
        w_t = ObtainSolution(w, S_p, m̄ + c, ε1, ε2);
        if f(w_t) ≤ f(w_best) then
            w_best = w_t;
            s_best = s;
            m = c;
            STOP = false;
        end
    end
    m̄ = m̄ + m;
end
return w_best;
```

Algorithm 9: `LocalSearch`

solution with all the velocities equal to 0 is feasible (though, obviously, not optimal). In practice, it might be interesting also to consider cases where initial and final velocities are positive values. In this case feasibility is not always guaranteed (just think about the case with initial velocity 0 and final velocity a large value which can not be reached due to the limitations on the acceleration). However, in such case we can replace the two constraints $w_1 = u_1$ and $w_n = u_n$ with $w_1 \leq u_1$ and $w_n \leq u_n$ so that feasibility is always guaranteed, and add a penalization term to the objective function

$$\gamma \left[\max\{0, u_1 - w_1\}\right]^2 + \gamma \left[\max\{0, u_n - w_n\}\right]^2,$$

where $\gamma > 0$ is a suitably large constant. We notice that after adding this penalization term the objective function is still convex and decreasing, so that we could still employ the proposed techniques with the penalized objective function replacing the original one.

7 Computational experiments

In order to test the proposed approach we made some experiments over a set of 100 instances with $n = 100$ sample points. The instances are generated by assuming that the trajectory over which the vehicle should travel is subdivided into up to $l = 5$ intervals over which the curvature is constant. Thus, the n-dimensional vector $\mathbf{u}$ of upper bounds for the velocities is generated as follows. First we fix $u_1 = u_n = 0$, i.e., the initial and final velocity must be equal to 0. Next, we partition the set $\{2, \ldots, n-1\}$ into l subintervals I_j, $j = 1, \ldots, l$, which correspond to intervals with constant curvature. Then, for each subinterval we randomly generate a value $\bar{w}_j \in (0, w_{max}]$, where w_{max} is the physical upper limit for the velocity of the vehicle (which has been fixed to 0.5). Finally, for each $j \in \{1, \ldots, l\}$ we set

$$u_k = \bar{w}_j \; \forall k \in I_j.$$

Thus, the upper bound vector $\mathbf{u}$ defines a discrete step function. All these vectors, as well as the corresponding ones for a further experiment described below, can be downloaded at `http:`

//www.ce.unipr.it/~locatell/VelPlanTests. Parameter A_T has been fixed to 0.02, A_J to 0.008 and the step h to 0.5.

Our experiments aim at showing that the proposed approach is both reliable, i.e., it returns a very good approximation of the optimal solution, and fast, i.e., the computing times are very small. Note that low computing times are quite important especially in applications where the velocity profile needs to be recomputed very often, in particular within the framework of real-time applications. Problem (11) is a convex one. Indeed, its feasible region is polyhedral while the objective function is convex. In fact, it can also be reformulated as a SOCP (Second Order Cone Programming) problem (see, e.g., [33]). In our experiments we employed the SOCP solver CPLEX, although some nonlinear local solvers such as MINOS and IPOPT have similar performance. All the experiments have been performed on an Intel(R) Core(TM) i7-3537U CPU @ 2.00GHz with Linux OS and 6 GB of RAM.

All the 100 instances with $n = 100$ have been solved by the proposed algorithm LocalSearch and by the SOCP solver CPLEX. The velocity profiles returned by the LocalSearch algorithm only have negligible violations of the PAR and NAR constraints, measured by

$$e_{ar}(\mathbf{w}) = \max\left\{0, \max_{i\in\{2,\dots,n-1\}}\{|w_{i-1} - 2w_i + w_{i+1}| - h^2 A_J\}\right\},$$

and of the acceleration constraints, measured by

$$e_{acc}(\mathbf{w}) = \max\left\{0, \max_{i\in\{1,\dots,n-1\}}\{|w_{i+1} - w_i| - hA_T\}\right\}.$$

In both case the maximum violations are around 10^{-15}. The quality of the solution returned by the LocalSearch algorithm has been measured in terms of *percentage relative error*

$$e_{rel}(\mathbf{w}) = \frac{f(\mathbf{w}) - f(\mathbf{w}^*)}{f(\mathbf{w}^*)} \cdot 100,$$

where $\mathbf{w}^*$ is the solution returned by the SOCP solver.

The SOCP solver encountered numerical problems in 6 out of 100 instances. In the remaining instances the minimum, average and maximum computing times of the SOCP solver were 0.038 s, 0.056 s, 0.33 s, respectively, with a variance equal to 0.00132. The results for the LocalSearch algorithm, which never encountered numerical problems, are reported in Table 1. The columns of the table report the average, the minimum, the maximum, and the variance respectively, both for what concerns the computing times (in ms) and the percentage relative error. The table reports the results for two different precision requirements in the final local search phase. The first two rows refer to a low precision final search, while the last two rows refer to a high precision local search. The low precision local search guarantees to return a solution of very good quality (the maximum percentage relative error is about 0.14%) in a very short computing time (the average computing time is 0.07 ms), while the high precision local search returns almost exact results (the maximum percentage error is 0.02%) at the cost of slightly larger computing times (on average, around 0.145 ms). In both cases very accurate solutions are returned with computing times much lower than those of the SOCP solver. As a further experiment we considered a U-turn path composed of a line segment, a circle arc and a final line segment (see Figure 9). This path has a length of $s_f = 500$m, and has the following $\mathcal{C}^2$ curvature function:

$$k(s) = \begin{cases} 0, & \text{for } s \in [0, 225) \cup (275, 500] \\ 0.07844, & \text{for } s \in (235, 265) \\ 0.07844\frac{(s-225)^3(245-s)^3}{10^6}, & \text{for } s \in [225, 235] \\ 0.07844\frac{(275-s)^3(s-255)^3}{10^6}, & \text{for } s \in [265, 275]. \end{cases} \tag{22}$$

Table 1: Computing times (in milliseconds) and percentage relative error for the proposed `LocalSearch` algorithm over the 100 instances with $n = 100$.

-	Avg.	Min	Max	Var
time [ms]	0.07	7.83e-3	0.362	0.58e-03
e_{rel}	2.87e-03	0	0.14	0.024
time [ms]	0.145	0.007	0.82	2.98e-03
e_{rel}	5.16e-04	0	2.67e-02	9.36e-04

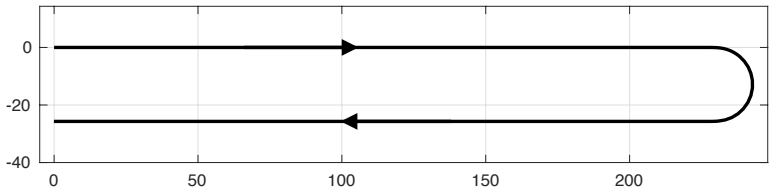


Figure 9: U-turn path obtained from curvature k defined in (22). Arrows denote the travel direction.

The tangential acceleration limit is set to $A_T = 2.78\ \mathrm{m\,s^{-2}}$ and the maximum normal acceleration to $A_N = 4.9\ \mathrm{m\,s^{-2}}$ (typical values for a generic economy car), the regularization term is set to $A_J = 0.4\ \mathrm{s^{-2}}$ The maximum velocity of the vehicle is $\bar{v} = 13.89\mathrm{m\,s^{-1}}$. Recall that $h = \frac{s_f}{n-1}$, where n is the number of sample points. Thus, the upper bounds u_i, $i = \{1, \ldots, n\}$ for the squared velocities appearing in (11c) are computed as follows:

$$u_i = \min \left\{ 13.89^2, \frac{A_N}{k(hi)} \right\} \quad i = 1, \ldots, n.$$

In order to see how the performance varies with the number of sampled points n, we performed tests with different n values (up to $n = 10000$) and compared them with the SOCP solver. Recalling the discussion in Remark 6.1, we notice that for large n values (say, $n \geq 1000$) critical sequences are not singleton. In particular, for $n = 10000$ there are two critical sequences whose length is 31. We only report the results for the case where a neighborhood search within the set of representatives is performed but we mention that even when the brute force approach is employed (i.e., all possible combinations of representatives for the critical sequences are considered) the computing times are clearly inferior with respect to the SOCP solver (10 s for the case $n = 10000$). The graph of the computing times is displayed in Figure 10, where the circles represent the computing times for the SOCP solver, while the squares represent the computing times for our approach. For the sake of illustration we also report in Figure 11 the solution with $n = 10000$ sampling points, whose traveling time equal to 24.85 s (the black line is the maximum velocity profile, while the grey one is the optimal velocity profile).

8 Conclusions and future work

In this paper we have developed a feasibility-based range reduction technique which can be applied in presence of special structured constraints. It is shown that the technique can be seen as an application of an algorithm over a graph whose nodes and arcs are related to the variables and constraints. The performance of the algorithm can be strongly enhanced by a proper choice of the order into which the nodes are explored, or, equivalently, by a proper decomposition of the graph. This is illustrated through the application of the proposed technique to a relevant practical application, namely velocity planning for a vehicle along a given trajectory. In this problem one searches for the velocity profile which minimizes the vehicle's traveling time taking into account physical bounds for the velocity, for the normal and tangential acceleration, and

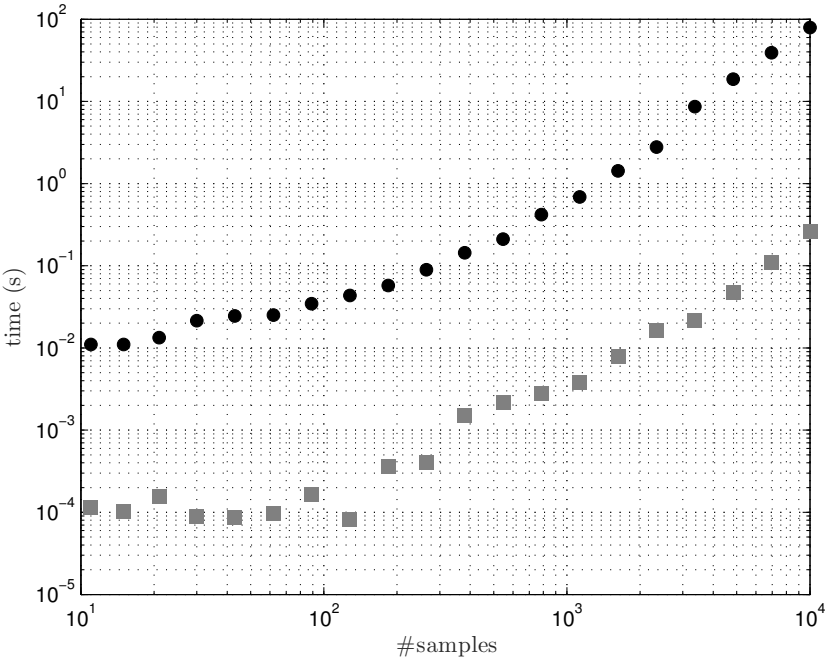


Figure 10: Computing times (in logarithmic scale) for different n values of the SOCP solver (circles) and of our approach (squares).

'comfort' constraints which allow for a smooth profile. It is first shown that subproblems which take into account only a subset of the constraints can be solved very efficiently. Taking this into account, a procedure is developed where the full problem is solved through the iterative solution of different subproblems which can be efficiently solved. The computational experiments show that this procedure compares quite favorably with the existing solvers and, above all, its computing times are compatible with real-time applications, where the velocity profile needs to be recomputed very often. Future developments include the study of the theoretical convergence of the proposed approach applied to problem (11) and the possible addition of other constraints.

References

[1] G. Ausiello, P. Franciosa, and D. Frigioni. Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach. In *Theoretical Computer Science. ICTCS 2001 - Lecture Notes in Computer Science*, volume 2202, pages 312–328, 2001.

[2] M. Bardi and I. Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Systems & Control: Foundations & Applications. Birkhäuser, 1997.

[3] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächther. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24:597–634, 2009.

[4] H. P. Benson. Simplicial branch-and-reduce algorithm for convex programs with a multiplicative constraint. *Journal of Optimization Theory and Applications*, 145:213–233, 2010.

[5] J. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3–17, 1985.

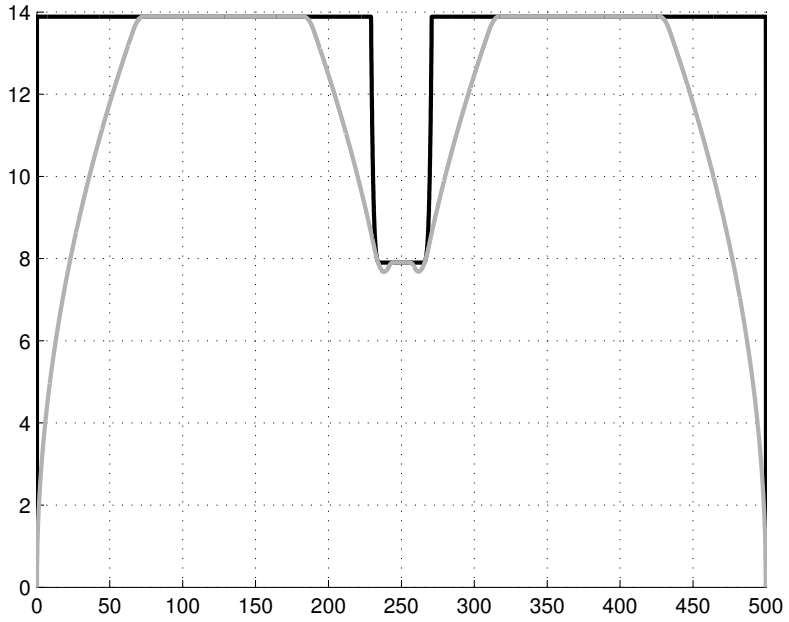


Figure 11: Optimal velocity profile (grey line) for the case with $n = 10000$ sampling points.

[6] F. Cabassi, L. Consolini, M. Laurini, and M. Locatelli. Convergence analysis of spatial-sampling based algorithms for time-optimal smooth velocity planning. 2017. submitted.

[7] A. Caprara and M. Locatelli. Global optimization problems and domain reduction strategies. *Mathematical Programming*, 125:123–137, 2010.

[8] A. Caprara, M. Locatelli, and M. Monaci. Theoretical and computational results about optimality-based domain reductions. *Computational Optimization and Applications*, 64(2):513–533, 2016.

[9] A. Caprara and M. Monaci. Bidimensional packing by bilinear programming. *Mathematical Programming*, 118:75–108, 2009.

[10] L. G. Casado, J. A. Martinez, I. Garcia, and Y. D. Sergeyev. New interval analysis support functions using gradient information in a global minimization algorithm. *Journal of Global Optimization*, 25:345–362, 2003.

[11] C. Chen, Y. He, C. Bu, J. Han, and X. Zhang. Quartic Bézier curve based trajectory generation for autonomous vehicles with curvature and velocity constraints. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6108–6113, May 2014.

[12] L. Consolini, M. Locatelli, A. Minari, and A. Piazzi. A linear-time algorithm for minimum-time velocity planning of autonomous vehicles. In *Proceedings of the 24th Mediterranean Conference on Control and Automation (MED), IEEE*, 2016.

[13] L. Consolini, M. Locatelli, A. Minari, and A. Piazzi. An optimal complexity algorithm for minimum-time velocity planning. *Systems & Control Letters*, 103:50–57, 2017.

[14] B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[15] D. C. Faria and M. J. Bagajewicz. Novel bound contraction procedure for global optimization of bilinear MINLP problems with applications to water management problems. *Computers and Chemical Engineering*, 35:446–455, 2011.

[16] M. Frego, E. Bertolazzi, F. Biral, D. Fontanelli, and L. Palopoli. Semi-analytical minimum time solutions for a vehicle following clothoid-based trajectory subject to velocity constraints. In *2016 European Control Conference (ECC)*, pages 2221–2227, June 2016.

[17] A. S. E. Hamed and G. P. McCormick. Calculation of bounds on variables satisfying nonlinear inequality constraints. *Journal of Global Optimization*, 3:25–47, 1993.

[18] P. Hansen, B. Jaumard, and S. H. Lu. An analytical approach to global optimization. *Mathematical Programming*, 52:227–254, 1991.

[19] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

[20] X. Li, Z. Sun, A. Kurt, and Q. Zhu. A sampling-based local trajectory planner for autonomous driving along a reference path. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 376–381, June 2014.

[21] M. Locatelli and U. Raber. Packing equal circles in a square: A deterministic global optimization approach. *Discrete Applied Mathematics*, 122:139–166, 2002.

[22] C. D. Maranas and C. A. Floudas. Global optimization in generalized geometric programming. *Computers and Chemical Engineering*, 21:351–370, 1997.

[23] V. Muñoz, A. Ollero, M. Prado, and A. Simón. Mobile robot trajectory planning with dynamic and kinematic constraints. In *Proc. of the 1994 IEEE Int. Conf. on Robotics and Automation*, volume 4, pages 2802–2807, San Diego, CA, May 1994.

[24] A. Nagy and I. Vajk. LP-based velocity profile generation for robotic manipulators. *International Journal of Control*. to appear, available online.

[25] H. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8:107–139, 1996.

[26] P. Shen, Y. Ma, and Y. Chen. Global optimization for the generalized polynomial sum of ratios problem. *Journal of Global Optimization*, 50:439–455, 2011.

[27] J. J. E. Slotine and H. S. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics and Automation*, 5(1):118–124, Feb 1989.

[28] R. Solea and U. Nunes. Trajectory planning with velocity planner for fully-automated passenger vehicles. In *IEEE Intelligent Transportation Systems Conference, ITSC '06*, pages 474 –480, September 2006.

[29] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming Theory, Algorithms, Software, and Applications*, volume 65 of *Nonconvex Optimization and Its Applications*. Springer Verlag, 2003.

[30] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004.

[31] B. Tóth and L. Casado. Multi-dimensional pruning from the Baumann point in an interval global optimization algorithm. *Journal of Global Optimization*, 38:215–236, 2007.

[32] E. Velenis and P. Tsiotras. Minimum-time travel for a vehicle with acceleration limits: Theoretical analysis and receding-horizon implementation. *Journal of Optimization Theory and Applications*, 138(2):275–296, 2008.

[33] D. Verscheure, B. Demeulenaere, J. Swevers, J. D. Schutter, and M. Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10), Oct 2009.

[34] J. Villagra, V. Milanés, J. Pérez, and J. Godoy. Smooth path and speed planning for an automated public transport vehicle. *Robotics and Autonomous Systems*, 60:252–265, 2012.

[35] J. M. Zamora and I. E. Grossmann. A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. *Journal of Global Optimization*, 14:217–249, 1999.