Linear Complimentarity Problem

Ferienakademie at Sarntal 2015 – Course 5

Friedrich Menhorn
Fakultät für Informatik
Technische Universität München
Email: menhorn@in.tum.de

***Abstract*— The linear complimentarity problem (LCP) is a problem class from mathematical optimization theory. The general formulation is to seek two vectors $\underline{z}$ and $\underline{w}$ which satisfy $\underline{w} = \underline{q} + M\underline{z}$, with constraints $\underline{w} \geq 0, \underline{z} \geq 0$ and $\underline{w} \circ \underline{z} = 0$, where M is a real matrix. These kind of problems often arise in computer games where one has for example many collisions between different kinds of objects. One can formulate the correct collision detection and collision handling as linear and quadratic programming problems. They, on the other hand, can be reformulated as an LCP, which offers a comfortable way of finding a solution for all objects at the same time. In this paper, the derivation from linear programming to LCP will be presented.**

***Index Terms*— Linear complimentarity problem, linear programming, quadratic programming, mathematical programming, Lemke algorithm, Simplex method**

I. Introduction

Collision detection and collision handling play an important role in computer games. Imagine a scenario where you shoot at a barrel and instead of the barrel getting pushed away it flies towards yourself– or maybe it does not move at all. Game critics would tear that game apart; ratings would tend towards zero with quotes like "you have seen better physics in the movie 'Fast & Furious 6' "; the game would become a shelf hugger.

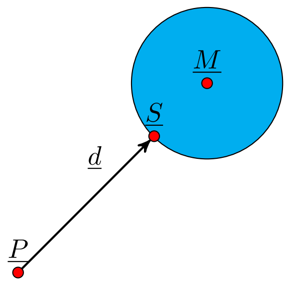

Fig. 1. Distance between point $\underline{P}$ and circle with center $\underline{M}$.

One way to resolve collisions correctly is to simplify the whole geometry– say one only looks at spherical bodies. For collision detection we usually need the distance between the colliding objects. For that, we want to take a look at Figure 1. The distance between a point $\underline{P}$ and the nearest point of a circle with center $\underline{M}$ can be calculated rather easily. We know the position of $\underline{P}$ and $\underline{M}$ and furthermore the radius r of the circle. Hence, the distance can be calculated as

$$||\underline{d}||_2 = ||\underline{M} - \underline{P}||_2 - r, \tag{1}$$

where $||\underline{x}||_2 = \sqrt{x_1^2 + x_2^2}$ is the Euclidean norm.

Of course, however, we do not want to have only circles or spheres in our games. We illustrate an example for that in Figure 2. Is the minimal distance from $\underline{P}$ to the polygon given by $\underline{d_1}$, $\underline{d_2}$ or $\underline{d_3}$? Compared to the first case, here, the distance calculation is already not as straightforward anymore and needs some geometrical understanding. In a hardcoded implementation, one might look at different cases and implement the calculation manually for each case. But is there a more general way do formulate these kinds of problems?

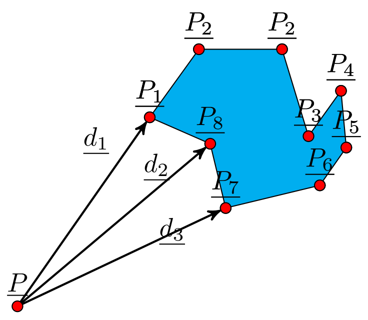

Fig. 2. Distance between point $\underline{P}$ and polygon.

Our requirements are the following: We want to be able to deal with more complex objects and shapes. Furthermore, we want to resolve collisions in multibody systems and not only between pairs, since it may be that by solving one collision we generate an artificial collision with another object which is close by.

In general, a problem of this kind can be described by a *linear complimentarity problem* (LCP), a problem class from mathematical optimization theory. In the next sections, we introduce this problem, its solution and application in collision detection and resolution. In section II we introduce the term *linear programming*, while in section III its correlation to the *dual problem* is presented. Next, we provide a description of the LCP in section IV, while in section V we enhance the newly acquired knowledge from linear programming to *mathematical programming*– in particular to *convex quadratic programming*. In the end we show a few examples of possible applications in section VI. In the description of these problems we follow the derivations in [1].

II. Linear Programming

The term *linear programming* corresponds to the problem of maximizing a given linear function which is subject to linear inequality constraints. It is therefore a special case of mathematical programming which will be described in section V. Here, *programming*, does not stand for programming mathematical algorithms on the computer but rather refers to an optimization problem. First solutions for these kind of problems date back to Fourier in 1827 [2]. Nowadays, it is widely used for example in company management for planning, production and transportation, since the goal of maximizing profit with limited resources can be modelled as a linear programming problem.

The general problem: Given a solution vector $\underline{x} \in \mathbb{R}^n$ and application-specified values $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^{n \times 1}$, $b \in \mathbb{R}^{m \times 1}$, we want to maximize the linear function

$$f(\underline{x}) = \underline{c}^T \underline{x}, \tag{2}$$

which is called *objective function*. The function is furthermore subject to nonnegativity

$$\underline{x} \geq \underline{0} \tag{3}$$

and linear inequality constraints

$$A\underline{x} \leq \underline{b}. \tag{4}$$

A vector which satisfies (4) is called *feasible vector* while a feasible vector which also fulfils (2) is named *optimal feasible vector*.

By looking at the problem formulation, we make an important observation. For the linear programming problem, the maximum is always located in a vertex of the region bounded by the domain [1]. Hence, a solution process is to find the vertex which maximizes f. For this, we first modify the linear constraints $A\underline{x} \leq \underline{b}$. By introducing the so-called *slack variable* $\underline{s}$ we eliminate the inequalities for each constraint $a_1x_1 + ... + a_nx_n \leq b_j, j = 1, ...m$:

$$s_j = b_j - (a_1x_1 + ... + a_nx_n), \tag{5}$$

such that we get an equality constraint

$$(a_1x_1 + ... + a_nx_n) + s_j = b_j \tag{6}$$

and the *normal form*

$$A\underline{x} + \underline{s} = \underline{b}, \underline{x} \geq \underline{0}, \underline{s} \geq \underline{0}. \tag{7}$$

For our search, we first have to find a vertex from which we start. To do this, we introduce a *artificial variables* $\underline{\text{w}}$:

$$\begin{aligned} b_j \geq 0 &: w_j = b_j - a_1x_1 - ... - a_nx_n - s_j \geq 0 \\ b_j \leq 0 &: w_j = -b_j + a_1x_1 + ... + a_nx_n + s_j \geq 0 \end{aligned} \tag{8}$$

and derive the *restricted normal form*

$$\underline{w} = D(\underline{b} - A\underline{x} - \underline{s}), \underline{x} \geq 0, \underline{s} \geq 0, \underline{w} \geq 0. \tag{9}$$

This leads to the *auxiliary objective function*

$$g(\underline{w}) = -\sum_{j=1}^{m} w_j = \sum_{j=1}^{m} d_j (\sum_{i=1}^{n} a_{ij}x_i - b_j + s_j), \tag{10}$$

which we want to maximize for $\underline{w}$. If the maximum is at $\underline{w} \neq \underline{0}$, there exists no solution. Otherwise, for $\underline{w} = \underline{0}$, the pair $(\underline{x}, \underline{s})$ is the starting vertex.

One method to solve this problem is the *simplex method*. In general, this method follows a two-phase process:

Simplex method

1) Solve restricted normal form

$$\underline{w} = D(\underline{b} - A\underline{x} - \underline{s}), \underline{x} \geq 0, \underline{s} \geq 0, \underline{w} \geq 0. \tag{11}$$

by optimizing the auxiliary objective function

$$g(\underline{w}) = -\sum_{j=1}^{m} w_j = \sum_{j=1}^{m} d_j (\sum_{i=1}^{n} a_{ij}x_i - b_j + s_j) \tag{12}$$

to find initial feasible basis vector $\underline{x}_0$.

2) Start search with $\underline{x}_0$ to solve the normal form

$$A\underline{x} + \underline{s} = \underline{b}, x \geq 0, s \geq 0. \tag{13}$$

by optimizing the objective function

$$f(\underline{x}) = \underline{c}^T \underline{x} \tag{14}$$

III. Dual Problem

The linear programming problem has a complimentary problem, namely the *dual problem*. To be consistent, we further on refer to the linear programming problem as *primal problem*. We introduced the primal problem as

$$\text{Maximize } f(\underline{x}) = \underline{(c)}^T x \text{ subject to } \underline{x} \geq \underline{0} \text{ and } A\underline{x} \leq \underline{b}. \tag{15}$$

Now, we can associate the dual problem

$$\text{Minimize } g(\underline{y}) = \underline{(b)}^T y \text{ subject to } \underline{y} \geq \underline{0} \text{ and } A\underline{y} \leq \underline{c}. \tag{16}$$

(16) is also solvable using the previously introduced simplex method. The difference is just to minimize instead of maximize the objective function (2).

The dual problem has a few properties which we want to shortly mention without further proof in Figure 3. A more extended derivation can be found in [1]. These properties state the relation between the primal and dual problem.

IV. Linear Complementarity Problem

The *linear complimentarity problem* (LCP) is now the connection between the primal and dual problem. To derive its definition we first introduce the *complimentarity slackness*. Remember the slackness vector for the primal problem

$$\underline{s} = \underline{b} - A\underline{x} \geq 0 \tag{17}$$

and for the dual problem

$$\underline{u} = A^T \underline{y} - \underline{c}. \tag{18}$$

The complimentarity problem is given as follows:

- $\underline{x} = (x_1, ..., x_n)$ feasible for primal problem
- $\underline{y} = (y_1, ..., y_m)$ feasible for dual problem

- Weak Duality Principle:

  If $\underline{x}$ is a feasible vector for the primal problem and $\underline{y}$ is a feasible vector for the dual problem, then $f(\underline{x}) \leq g(\underline{y})$
- Unboundedness Property:

  Definition: Linear programming model has unbounded solution if the objective function can be made arbitrarily large
  - If the primal problem has an unbounded solution, then the dual problem has no solution (*infeasible*)
  - If the dual problem has an unbounded solution, then the primal problem is infeasible
  - The primal problem has a finite optimal solution iff the dual problem has a finite optimal solution
- Strong Duality Principle:

  If $\underline{x}$ is a feasible vector for the primal problem and $\underline{y}$ is a feasible vector for the dual problem, then these vectors optimize f and g iff $f(\underline{x}) = g(\underline{y})$

Fig. 3. Properties of the primal and dual problem.

- $\underline{x}, \underline{y}$ optimize f, g iff:

$$\begin{aligned} x_i &= 0 \text{ or } u_i = 0 \\ y_j &= 0 \text{ or } s_j = 0 \end{aligned} \tag{19}$$

or in vector notation

$$\underline{x} \circ \underline{u} = \underline{0}, \underline{y} \circ \underline{s} = \underline{0}. \tag{20}$$

Here, the $\circ$-operator denotes $\underline{x} \circ \underline{y} = 0 \Leftrightarrow x_1 * y_1 = 0, x_2 * y_2 = 0, ..., x_n * y_n = 0$.

Using this, we can now directly derive the LCP, which can be seen as a combination of the primal and dual problem. Given $\underline{q} \in \mathbb{R}^{k \times 1}$ and $M \in \mathbb{R}^{k \times k}$, construct $\underline{w}, \underline{z} \in \mathbb{R}^{k \times 1}$, such that

$$\underline{w} = \underline{q} + M\underline{z}, \underline{w} \circ \underline{z} = \underline{0}, \underline{w} \geq \underline{0}, \underline{z} \geq \underline{0}. \tag{21}$$

Due to the condition $\underline{w} \circ \underline{z}$, w_i and z_i are said to be *complementary*. For the linear programming problem, this leads to the following system

$$\begin{aligned} \underline{q} &= \begin{bmatrix} -\underline{c} \\ \underline{b} \end{bmatrix}, M = \begin{bmatrix} 0 & A^T \\ -A & 0 \end{bmatrix}, \\ \underline{w} &= \begin{bmatrix} \underline{u} \\ \underline{s} \end{bmatrix}, \text{ and } \underline{z} = \begin{bmatrix} \underline{x} \\ \underline{y} \end{bmatrix}, \end{aligned} \tag{22}$$

which can also be solved with the simplex method.

Another method, to solve these kind of systems is called *Lemke algorithm*. Here, we consider $\underline{w} = \underline{q} + M\underline{z}$ as a dictionary for $\underline{w}$ defined in terms of $\underline{z}$. This means each word w_i is defined by already defined words z_i. If $\underline{q} \geq \underline{0}$, the dictionary is considered *feasible*. Then, the solution is trivial with $\underline{w} = \underline{q}$ and $\underline{z} = \underline{0}$. Otherwise, the two-phase Lemke algorithm is applied with the goal of finding a feasible dictionary. The algorithm is described in Figure 4. A variable is *nonbasic* if it is part of the dictionary; that means it is defined by other variables, hence it stands on the left side of the equation. The algorithm results in an iterative scheme until a terminal dictionary is reached.

Lemke algorithm

- Starting point: The dictionary $\underline{w} = \underline{q} + M\underline{z}$ is not feasible
- Two-phase algorithm:
  1) Add auxiliary variable $z_0 \geq 0$ by modifying the dictionary

$$\underline{w} = \underline{q} + M\underline{z} + z_0\underline{1} \tag{23}$$

  2) Exchange z_0 and w_i such that:

     A. z_0 is nonbasic

     B. z_i or w_i is nonbasic for each i
- If A and B are satisfied, dictionary is called *terminal*
- If only B is satisfied, dictionary is called *balanced*

Fig. 4. The Lemke algorithm

Two problems may arise during the scheme. Due to degeneracies, which means constant terms can become zero, the algorithm may lead to cycling– a variable which was added earlier just leaves the dictionary again. To resolve this, one can apply a small pertubation ϵ^j with $\epsilon \in (0,1)$ to each equation $j = 1, ..., k$. This leads to a solution depending on ϵ which can be solved in the final step. However, we want to make the reader aware of the fact that in terms of implementation this leads to a symbolic solution process which adds to the complexity. Another pitfall may be that a variable complementary to the one which has just left the dictionary cannot enter because it was already removed. One can show, however, that the LCP has no solution in this case [1]. Hence, the algorithm may not have a solution but always terminates.

V. Mathematical Programming

Linear programming is a special case of mathematical programming. The general problem is defined as:

Minimize arbitrary objective function $f(\underline{x}), f : \mathbb{R}^n \to \mathbb{R}$, subject to constraining functions $g(\underline{x}) \leq 0, g : \mathbb{R}^n \to \mathbb{R}^m$.

Note: If we want to maximize f, we can just minimize $-f$. The linear programming definition can also be defined as mathematical programming problem:

$$f(\underline{x}) = -\underline{c}^T x, g(\underline{x}) = \begin{bmatrix} A\underline{x} - \underline{b} \\ -\underline{x} \end{bmatrix} \tag{24}$$

Another subfamily of mathematical programming is *convex quadratic programming*. This is especially interesting for us, since for example distance functions usually derive from a quadratic equations. In general the problem states as:

$$f(\underline{x}) = \underline{x}^T S \underline{x} - \underline{c}^T \underline{x} + K, \tag{25}$$

with a constant, symmetric matrix S, constant vector $\underline{c}$ and constant scalar K. The constraining function is given by

$$g(\underline{x}) = (A\underline{x} - b, -\underline{x}) \leq 0 \tag{26}$$

and from S being positive-semidefinite we know that f is convex. Here, we want to mention the *Karush-Kuhn-Tucker conditions* [3] which allow us to reformulate the quadratic programming problem as LCP:

$$\underline{w} = \underline{q} + M\underline{z}, \tag{27}$$

where

$$\begin{aligned} &\underline{x}_s = \underline{b} + A\underline{x}, \underline{q} = \begin{bmatrix} -c \\ b \end{bmatrix}, M = \begin{bmatrix} 2S & A^T \\ -A & 0 \end{bmatrix}, \\ &\underline{w} = \begin{bmatrix} \underline{u}_s \\ \underline{x}_s \end{bmatrix}, \underline{z} = \begin{bmatrix} x \\ \underline{u}_d \end{bmatrix} \end{aligned} \tag{28}$$

VI. Applications

Due to this fact of being able to transform for example linear as well as quadratic programming problems to LCP gives this solution method a powerful field of applications.

Returning to our initial example of collisions we can take a look at distance calculations. In terms of a quadratic programming problem the distance between a point $\underline{P}$ and a convex polygon is given as:

$$\begin{aligned} &f(\underline{x}) = |\underline{x} - \underline{P}|^2 = \underline{x}^T I \underline{x} - 2\underline{P}^T \underline{x} + |P|^2 \\ &S = I, \underline{c} = 2\underline{P}, K = |\underline{P}^2|, \underline{x} \geq 0 \end{aligned} \tag{29}$$

The constraints hereby describe the area inside the polygon. Similarly, this holds for convex polyhedrons and distances between more complex shapes. We can solve these by the above described transformation to a LCP and the application of, for example, the Lemke algorithm.

The same applies for the computation of contact forces. Say we want to minimize $|A\underline{f} + \underline{b}|^2$ with constraints $\underline{f} \geq 0$ which means forces are repulsive. Further constraints are $A\underline{f} + \underline{b} \geq \underline{0}$ (bodies cannot interpenetrate) and $A\underline{f} + \underline{b} \leq \underline{c}$ (kinetic energy cannot be gained via impulses). Hereby, $\underline{f}$ is the impulse magnitude, $A \in \mathbb{R}^{n \times n}, \underline{b}, \underline{c} \in \mathbb{R}^{n \times 1}$. The quadratic form can be written as

$$\underline{f}^T (A^T A) \underline{f} + (2\underline{b}^T A) \underline{f} + |b|^2 \tag{30}$$

and can again be reformulated as an LCP.

Another application might be a multibody collision. Imagine that there occur multiple collisions in the current time step and multiple objects of various shapes are overlapping. A pair wise collision handling may result in new collision with other shapes. With the help of the mathematical programming problem, we can set up a system where we want to minimize the distances between the different shapes with the constraint that the objects are not overlapping. Afterwards, this system can then be solved with LCP again.

VII. Conclusion

With its usually high number of objects and their interactions, computer games is a big field of application for LCP. LCP helps to make a game look realistic while not draining too many resources. For that reason, also many physics engines are using LCP internally, implementing different kinds of solution algorithms– one example is the Bullet Physics Library [4]. Also for Matlab, for example, an LCP solver can easily be found when searching the web [5].

Since this paper can only be seen as an appetizer for LCP and mathematical programming problems and their various fiels of applications, we like to use this section as well to refer to a few other sources for the interested reader. Apart from the mentioned book by Eberly and Shoemake [1], [6] and [7] also offer a more detailed view on LCP and its applications.

References

[1] D. H. Eberly and K. Shoemake, *Game Physics*, 2004.

[2] G. Sierksma, "Linear and integer programming : Theory and practice."

[3] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, Calif.: University of California Press, 1951, pp. 481–492.

[4] B. P. Library, November 2015. [Online]. Available: http://bulletphysics.org/wordpress/?page_id=9

[5] Yuval and Mathworks, "Lcp / mcp solver (newton-based)," November 2015. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/20952-lcp---mcp-solver--newton-based-

[6] J. Friedman, "Linear Complementarity and Mathematical (Non-linear) Programming," pp. 1–22, 1998. [Online]. Available: https://www.math.ubc.ca/~jf/courses/old_kkt.pdf

[7] R. W. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem*. SIAM, 1992. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=bGM80_pSzNIC&pgis=1