Algovize

Luděk Kučera

January 12, 2008

1 Simplex algorithm

In the introduction I have to make a disclaimer: the chapter shows only a very small part of linear programming that is practically important. For example, I would not touch the duality of linear programming, which is the most important result of the theory, because the duality is a mathematical relationship and I know no way of its visualization that would make understanding of the concept easier. For the same reason I will not mention primal and dual methods of linear programming. Outside of the scope of the book are also questions of integer programming and polynomial time algorithms, e.g., the elipsoidal method and the iterior point method (but the latter will hopefully be visualized later). In the present book we restrict ourselves to main ideas of the simplex algorithm.

I will not strictly follow the standard notation and formalism that are used in connection with the duality theory, but they present tricks that are not important for simplex algorithm. A linear programming will be presented as a linear optimization over a general polyhedron and the simplex algorithm as a walk among vertices of a polyhedron that is monotone with respect to a linear functional.

This chapter deals with optimalization in the plane and later in the space, because these cases can easily be visualized and build a geometrical intuition to the linear programming and in particular to the simplex algorithm. It is not difficult to generalize this intuition to cases of higher number of dimensions.

Scene: *Lines in the plane*

Points $[x, y]$ of the plane that satisfy equation $ax+by = c$, where a and b are not simultaneously qual to 0 (and x and y represent x- and y-coordinate of the point) form a line and, conversely, any line in the plane can be describes in this way.

This scene shows the x-axis and the y-axis together with scales. Draw a line by pressing the mouse at a certain point of the plane and releasing it at another point. This will define a line that passes through both points dermined by the mouse actions. The line is associated with its equation. Before the mouse is released, the line is red and the point of the mouse pressing is denoted by a small red circle and an arrow shows the immediate position of the cursor. When the mouse is released, both the circle and the arrow dissappear and the line color changes to dark blue. Lin drawing can be repeated many times. Equations of all lines appear in the display (the line being drawn in a yellow frame, others in white frames).

There are other modes of mouse function that can be chosen in the control bar. In addition to [**Draw line**] you can choose [**Choose and edit line**], [**Delete line**] and [**Move inequalities**].

When [**Choose and edit line**] is chosen, one of drawn lines can be selected by a mouse click. The selected line color changes to magenta and the line is shown with a circle and an arrow that are identical with those when drawing of the line has been finished. In the same time its equation gets yellow. This function is sometimes selected to know which equation corresponds to the line, but we can also change the line by dragging either the circle or the arrow. A click to another place deselects the line (and can select another line).

When [**Delete line**] is chosen, a click to a line deletes the line. The choice makes another button [**Delete all**] to appear that can be used to erase all lines in the screen.

When [**Move inequalities**] is chosen, the block of equations can be dragged to another position, no other mouse function is available. The block of equations can be made larger or smaller by the buttons [**+**] and [**-**] and can be hidden or shown by the checkbox [**Show inequalities**]. It is also possible to hide coordinates.

Let us note that the same line can be described by many different equation, but any equation can be transformed to another one by multiplying the numbers a, b and c in the equation by the same number different from 0. In the applet we use the unique equation such that $|a| + |b| = 1000$.

Scene: *Halfplanes*

A line cuts the plane into two halfplanes. In this scene we will assume closed halfplanes, i.e., the halfplanes include the boundary line. In this way the line is equal to the intersection of the halfplanes.

The two halfplanes determined by a line $ax+by = c$ can be described as sets of points $[x, y]$ that verify respective inequalities $ax + by \leq c$ and $ax + by \geq c$. The latter inequality can also be written as $(-a)x + (-b)y \leq (-c)$ and therefore each halfplane can be written in the form $Ax + By \leq C$ for some A, B, C and use always operator $\leq$ and the constant on the right hand side.

Draw a line in the same way as in the previous scene; a line appears and the division of the plane into two halfplanes is stressed by colors. In the sequel, we will always consider the green halfplane as a halfplane determined by the line. Note that the green halfplane is on the right side if the line is traversed in the direction of the arrow, that is drawn as a halfarrow to underline which halfplane is selected.

If the mouse button is not pressed, the value of the left side of the inequality is given for a point that is determined by the immediate position of cursor. A color of the inequality background tells us whether the inequality is satisfied. If the sharp inequality holds, the background is light green, if the left side is equal to the right side, the background is dark green, and it has a magenta color if the inequality is not satisfied. Check that the background is green if the cursor is in the selected green halfplane and magenta if it is outside.

Unlike in the previous scene, only one line is shown, and a new line erases the previous one.

Scene: *Linear inequalities and a convex polygon*

In this section we will deal with convex polygons. A convex polygon in the plane is a collection of all points $[x, y]$, that satisfy a set of inequelities

$$a_1x + b_1y \leq c_1$$

$$a_2x + b_2y \leq c_2$$

$$\dots$$

$$a_mx + b_my \leq c_m$$

wherein $a_1, b_1, c_1, \dots, a_m, b_m, c_m$ are constants. We see that a convex polygon is an intersection of halfplanes.

This scene allows you drawing a convex polygon by adding linear inequalities that are obtained in the same way as in the previous scene. However, a new line doesn't erase the previous ones, all lines and the corresponding inequalities remain. It is possible to edit and/or delete them in the same way as in the first scene.

You can see many halfplanes in the same time. Points that don't belong to any of dark blue lines have the following colors: points that belong to *all* halfplanes corresponding the the inequalities are *green*. Hence, the green area is the convex polygon determined by the set of inequalities. The remaining points are blue, but some are darker and some lighter. The greater is the number of halfplanes that contain a point, the darker is the blue color of the point. Moreover, if the mouse button is not pressed, the validity of inequalities for the point determined by the cursor is shown in the seame way as in the previous scene. Move the cursor and observe which inequalities are satisfied.

A new (randomly constructed) convex polygon can also be generated by the button [**New polygon**]. The polygon can be edited and modified by adding, editing and/or deleting lines in the way shown in the first scene. The button [**Just polygon**] switches to the mode when all points outside the green polygon have the same blue color, i.e. there are no differences based on the number of halfplanes containig the point. Use this choice if you are interested in the polygon only.

A convex polygon can be either unbounded, or bounded but nonempty, or void. For one inequality ($m = 1$) it consists of one unbounded halfplane; for two inequalities it has usually a form of a wedge, but if the two lines are parallel, it can also be one halfplane, a stripe of the constant width and extending to infinity on both sides (including a line as a stripe of zero width), or it can be void. Try to draw all these possiblities.

The most important form of a polygon with three boundary lines is a bounded triangle, but it can also have several other bounded and unbounded forms.

If inequalities are added on-by-one, then adding a new inequality to a collection that has at least one solution can result in several different possibilities:

1. the polygon gets smaller, but still non-empty, which means that some solutions of the original system of inequalities satisfy the new inequality, some others don't,

2. the polygon remains unchanged - all points of the original polygon satisfy the new inequality, which brings nothing new,

3. no solution of the previous inequalities satisfies the new one, and therefore the polygon becomes void.

Scene: *Linear funktional*

A linear funktional is a function f, which maps any point $u = [x, y]$ of the plane into a number $f(u) = Ax + By$, where A and B are constants. The value is a scalar product of a vector $[A, B]$ and the vector $[x, y]$. The display shows values of a functional that coresponds to a randomly chosen vector (A, B). Values are visualized using a temperature scale: the largest displayed values are white, then yellow that gets darker that is followed by light to dark red, later light blue that also gets darker and eventually changes to the final black (not all colors are necessarily shown). You can also see coordinates and an arrow representing the vector (A, B) that starts in the origin and ends in a point with coordinates A and B.

Move the cursor, the head of the vector arrow is always determined by the cursor (except when the cursor is too close to the origin - avoiding problems of drawing an arrow that is too short, and when the cursor is too far from the origin - in such a case the color field would be too narrow and we would lack colors).

It can be seen that the functional values change fast in the direction of the vector (A, B), while remaining unchanges in the direction perpendicular to the vector. This is because the value of the functional in a point $[x, y]$ is the scalar product of the vectors (A, B) and (x, y). The scalar product of the vector (A, B) with an increment that is perpendicular to (A, B) is zero, while the largest value is when the increment of the vector (x, y) is colinear with (A, B).

The scene also shows some "contour lines", lines connecting points having the same value of the functional. Contour lines are grey and are orthogonal to the vector defining the functional and therefore they are mutually parallel.

It is clear that the functional grows fast if the vector $[A, B]$ is long, while if the vector is very short, the functional values are almost constant in the screen.

As you will se later, only the direction of the vector $[A, B]$ is important, and the point of a polygon that is a solution of the linear optimization problem is unchanged if only the length of the vector changes.

There is a small rectangle in the upper right corner of the display, which shows the value of the functional in the point determined by the cursor. Move by the cursor to see how the functionsl changes.

Scene: *Linear optimalization in plane*

In this scene we arrive to the basic problem that we are going to solve. The display shows a convex polygon together with a linear functional, and our goal is to find the point of the polygon that maximizes the value of the functional.

A polygon in the window is generated randomly and filled by the green color. As a background, the values of the functional are given in the same way as in the previous scene using the temperature scale.

The solution of the linear optimalization problem is the point that is placed in the lightest point of the polygon and is marked by a small red circle. Change the functional by moving the cursor having the mouse button pressed and observe the solution.

It is clear that the solution is one of the vertices of the polygon, i.e. a point of the boundary, where the boundary "bends". Only very infrequently it could happen that the set of solutions is an edge of the polygon, connecting its two adjacent vertices.

Clicking the button [**New polygon**] creates another polygon. Use a checkbox in the control bar to ask for either a bounded or an unbounded polygon. The functional vector can be changed by the mouse. In order to simplify the scene, there is no possibility to modify the polygon in a way shown in previous scenes.

It is clear that if a polygon is bounded and is not void, the problem has always a solution, usually one vertex, sometimes one edge of the polygon (but the solution might fall outside of the screen). However, if the polygon is unbounded, the solution can but need not exist. Create an unbounded polygon and arrange the functional vector for both possibilities.

The problem to solve a linear problem in the plane (i.e., with two variables x and y) is not difficult, and it would be possible to solve it by direct methods, but the method that will be shown in the next scenes generalizes to an arbitrary dimension as the "simplex algorithm", which is still probably the most frequently used method to solve linear programs, even though it was conceived by G. B. Danzing in 1947.

Scene: *Walk along an edge in the plane*

As shown in the previous scene, if the linear optimization problem has a solution, then the solution is, or can be, a vertex of the polygon. The simplex algoritm finds a solution by starting in an arbitrary vertex of the polygon and moving on the boundary of the polygon until the solution is found (or shown not to exist).

In the present scene the polygon is always non-empty and bounded. The computation will be visualized by a yellow token that moves along the polygon boundary. In this scene, movement of the token is triggered by a user. Press one of the buttons [**Go CW**] (clockwise) and [**Go CCW**] (counterclockwise), and the token starts moving along the boundary of the polygon in the selected direction. Try this for some time.

If you find this boring, you can create another polygon using the button [**New polygon**], but most likely you will find the whole scene boring, because the goal is so simple! However, I will still explain how to find an adjacent vertex if we are not looking at the screen, but have just inequalities defining the polygon and the position of the vertex whete the token sits.

Instead of the buttons [**Go CW**] (clockwise) and [**Go CCW**], press one of the buttons [**Explain CW**] and [**Explain CCW**]. Instead of moving the token to a neighbor vertex, Algovision would show facts necessary to explain how an appropriate adjacent vertex is found:

- You can see all lines that pass through polygon edges. The equations of the lines can be obtained from inequalities by replacing $\leq$ by $=$ (equal).
- There are two edges starting in the token vertex. The one that would be traversed by the token after pressing the button [**Go** ...] is now drawn as a red arrow and is a part of a red line and will be called a *pivot* line, the other one is *non-pivot*.
- Intersections of the red line determined by the pivot edge with dark blue lines passing through other edges (other than the non-pivot one) are marked by red and black circles (note that some intersection might fall outside of the screen).

- A halfline determined by the inequality corresponding to the non-pivot edge is now darker than the background (but partially hidden by the polygon). The intersections mentioned in the previous paragraph are red if and only if they belong to this darker halfplane, and they are black otherwise.
- The terminal vertex of the pivot edge, which is the vertex to which the token would move, is a red intersection point that is the closest one to the token.

It is easy to determine an intersection of two lines: it is sufficient to solve a system of two linear equations given by the equations of the two lines and it is even simpler to determine which of them are the red ones: plug their coordinates into the inequality corresponding to the non-pivot edge. Red intersections are those that satisfy the inequality.

This scene could be skipped if you only want to know the idea of the simplex algorithm. However, if you are going to write a code executing the simplex algorithm, stay in this scene until you understand it well.

Scene: *Walk along an edge of an unbounded polygon*

The scene is similar to the previous one, but the polygon is unbounded. Some edges are now unbounded halflines. If the token moves along such an edge, it never arrives to a neighbor node, but it would follow the edge for ever. It is therefore necessary to get it back by clicking the button for the opposite direction.

You can again switch to the [**Explain**...] mode. If the token would go along an infinite edge, no red intersection is created, because the halfline representing the edge (which is an intersection of the red line passing through the pivot edge and the dark halfline corresponding to the non-pivot inequality) doesn't intersect any of the lines passing through other edges.

Scene: *Simplex algorithm in the plane*

The scene essentially repeats two preceeding ones - it shows a convex polygon and a token that moves on its boundary; here, however, Algovision and not user moves the token. Moreover, the scene shows a linear functional and the token always moves up to higher values of the functional.

When the choice [**Functional**] is selected, the functional can be changed by the mouse, [**Place token**] allows a user to select the starting position of the token by clicking a node of the polygon.

After setting the starting situation, click the button [**Compute**]. Algovision determines the optimum of the functional using Simplex Algorithm and switches to animation mode, where the computation can be stepped or run. Each step represents a movement of the token from one node to its neighbor node and steps are repeating until the optimal solution is found. After that the token changes its color to red. The button [**Abort**] switches back to the input mode.

Since we are working in a plane, two edges of the polygon are incident with each node. In most cases the token is in a node that is an end-node of one edge that goes to higher values of the functional, and of one node that descends to lower values (if not at the beginning of the computation, the latter edge is the one used by the token in the previous step). In such a case it is clear how the token moves. At the beginning, it could happen that both edges climb to higher values of the functional; in such a case it is not important, which edge will be chosen. Finally, if the functional decreases (or at least doesn't increase) along both edges, the token is in the optimal node and the computation stops after changing the color of the token to red.

Scene: *Simplex algorithm - unbounded polygon*

The previous scene is repeated, but all convex polygons are unbounded. If the functional increases along an infinite halfline edge, it can reach arbitrarily large values and the solution doesn't exist.

Scene: *Simplex it yourself in the plane*

The scene repeats the previous two scenes, but movements of the token are directed by a user using buttons [**Go CCW**] and [**Go CW**] in the same way as shown above. It is necessary to move the token in such a way that it always moves to higher values of the functional. When no such move is possible, click [**Maximum**].

Depending on the status of the checkbox [**Unbounded Polygon**] on the control bar, you can ask either for a bounded or an unbounded polygon. If the polygon is unbounded and the token is in a node that is incident with an edge represented by an unbounded halfline along which the functional increases, press the button [**Unbounded**] to indicate that the functional is unbounded and its maximum doesn't exist. The token doesn't get red, because it is not in the optimum node. Don't press the button when you see at the display that a solution doesn't exist, but this is not visible from the node where the token is located.

If you give a command to go in the incorrect direction or press one of the buttons [**Maximum**] or [**Unbounded**] too early, Algovision performs no action and protests. Error messages (that begin with "!!!") are

- "Decreasing direction" - if a command was given to go in the direction of decreasing functional
- "Not maximum" - if the button [**Maximum**] was pressed when the token is not in the point of the largest value of the functional on the polygon
- "Unbounded direction" - if a command was given to go in the direction of the functional increasing along an unbounded edge (the button [**Unbounded**] should be pressed in this case)
- "Upper-bounded only" - if the button [**Unbounded**] was pressed when values of the functional on any edge incident with the token node are upper bounded (which happens when all edges incident with the token node are either finite line segments regardless of values of the functional and/or unbounded halflines along which the functional is decreasing when moving to the infinity).

Select an arbitrary initial configuration (polygon, functional, token position) and move the token to a node where you can press either [**Maximum**] or [**Unbounded**] without commiting an error. Repeat computation several times from different initial configurations involving both bounded and unbounded polygons, until no error occurs in several consecutive computations.

Scene: *Polyhedron in the space*

In the present scene we repeat in the three dimensional space everything that has already been said for two dimensional space (plane) in a way that would suggest generalization for spaces of arbitrary finite dimension.

An equation $ax + by + cz = d$, which is an analog of an equation of a line in the plane, determines a plane in the 3D space. Since it is difficult to show a general plane, embedded into the three dimensional space, in a two dimensional window, a scene showing planes in space is not presented.

A plane separates the 3D space into two halfspaces in a way that is similar to separation of the plane by a line. These halfspaces are again considered as closed, i.e., including the separating plane, and therefore the separating plane is their intersection. The halfspaces determined by a plane $ax + by + cz = d$ are given by inequalities $ax + by + cz \leq d$, $ax + by + cz \geq d$, respectively. The latter one will be written as $a'x + b'y + c'z \leq d'$, where $a' = -a$, $b' = -b$, $c' = -c$ and $d' = -d$ in order to use always the operator $\leq$ and a constant term on the right hand. A halfspace is also difficult for visualization.

A polyhedron is the intersection of a finite number of halfspaces. It is a convex body, an example of which is shown in the present scene. Mathematically a polyhedron is a set of points of the three dimensional space, verifying the following collection of inequalities

$$a_1x + b_1y + c_1z = d_1,$$
$$a_2x + b_2y + c_2z = d_2,$$
$$\dots$$

$$a_m x + b_m y + c_m z = d_m,$$

where $a_1, \ldots, a_n, b_1, \ldots, b_n, c_1, \ldots, c_n, d_1, \ldots, d_n$ are constants.

In most cases it is easy to visualize a polyhedron. The scene offers a collection of several polyhedra that can be selected: a random polyhedron, perfect or Platonic solids (Tetrahedron, Hexahedron or Cube, Octahedron, Dodecahedron, Icosahedron) and several pyramids. The choice [**Random**] makes it possible to create a new polyhedron by the button [**New polyhedron**]; it is possible to select either bounded or unbounded polyhedron. The numerical field right to [$N =$] is used to select the number of faces of the polyhedron. However, the actual number of faces is often different. In N is too small and a bounded polyhedron is required, it might happen that N randomly defined inequalities are not enough to create a bounded body; in such a case Algovision adds further random inequalities until a bounded body is obtained. On the other hand, it could happen that certain inequalities are useless - if deleted, the polyhedron doesn't change. Such inequalities are not displayed, because they do not correspond to any face.

Using the mouse, a polyhedron in the screen can be moved. If the choice [**Rotate polyhedron**] is selected, a horizontal movement of the mouse rotates the polyhedron around the vertical axis and a vertical movement along the horizontal axis (this sounds strange, but, as you will see, is quite natural). If the choice [**Move polyhedron**] is selected, the polyhedron can be dragges in the screen without being rotated. Finally the selection of [**Zoom polyhedron**] makes it possible to zoom in and out the polyhedron without rotation by vertical movements of the mouse (horizontal movements have no influence on the size of the polyhedron).

Let us note that a polyhedron can be bounded (as it is when entering this scene and in most examples) or unbounded (use selection [**Random**] and the checkbox [**Unbounded**]). When rotating an unbounded polyhedron, we can happen to be inside. It is dark (black) inside, just edges are white. Internal points belongs to the polyhedron, but I prefer showing the polyhedron as hollow; note that the simplex algorithm operates on the polyhedron boundary. If the polyhedron is bounded, we assume that the observer is sufficiently far from it and it remains always outside.

The anatomy of a three dimensional polyhedron is similar as in two dimensions, but slightly more complex. All inequalities are satisfied as sharp for points of the interior that are not visible from outside. Points satsifying all inequalities, at least one of which as equality, belong to a boundary of the polyhedron. All points of the polyhedron that satisfy certain inequality as equality form a *face* of the polyhedron. We say that an inequality and a face correspond each other.

The window also shows inequalities corresponding to faces of the polyhedron. Inequalities can be hidden. The color of the beckground of any inequality doesn't represent validity of the inequality, but it is the same as the color of the corresponding face, provided the face is visible. In the case of an invisible face, the corresponding inequality is written in grey on a black background.

An *edge* of a polyhedron is a line segment, halfline or line that is an intersection of two neighboring faces. Edges of a bounded polyhedron are always finite line segments, while an unbounded polygon has at least one edge, which is a halfline or a line, in most cases a halfline (lines as edges appear only infrequently). A *vertex* is a point that is common to at least 3 edges (usually exactly 3 edges, but sometimes more, see Platonic bodies). Equivalently, a vertex is a point that is common to at least 3 faces.

Scene: *Inequalities*

Unlike in a plane, in three-dimensional space it is difficult to select a point of the space by the mouse. However, if the cursor points to the polyhedron, a unique point of the polyhedron visible boundary is chosen. We can't select an interior point or a point on the invisible side of the polyhedron (unless the polyhedron is rotated properly). If the cursor points outside of the polyhedron, no point is selected.

If the mouse button is down, colors are the same as in the previous scene, but if it is released, other coloring is used. Inequalities corresponding to invisible faces remain gray on a black background, but inequalities corresponding to visible faces use black letters on a light or dark green background.

The choice of light or dark green shows how inequalities corresponding to visible faces are satisfied for a point determined by the cursor on the visible part of the boundary of the polyhedron.

Move the cursor on display with the mouse button released. If the cursor is outside of polyhedron, the background of all inequalities corresponding to visible faces is *light* green. Now, move the cursor onto the polyhedron. The inequality corresponding to the face where cursor is located is now *dark* green.

There are essentially three possibilities. If the cursor is *inside* of some face, exactly one inequality is dark green - the one coresponding to the face. The cursor could also be located on an edge of the polyhedron, but not in its end-vertex. Since the edge belongs to two faces, two inequalities would be dark green. Finally, the cursor can be located in a vertex of the polyhedron. In most cases, a vertex belongs to exactly 3 faces, and hence 3 inequalities would be dark. However, a vertex is generally an element of at least 3 faces and in some cases 4 or more faces contain a vertex and in such a case 4 or more inequalities would be dark. For example, choose an arbitrary vertex of an octahedron or an icosahedron or the top vertex of a pyramid with at least 4 sides.

It is important to realize that if the cursor is in a vertex, i.e., at least 3 inequalities are dark (satisfied as equality), then choosing an edge starting in the vertex means that exactly two of the dark inequalities are selected and kept dark.

In the usual case, where exactly 3 inequalities are dark in the vertex, selection of two dark inequalities is equivalent to letting one dark inequality get light green (when leaving a vertex along an edge, one leaves one face while remaining on the boundary of remaining two that meet in the vertex). Note also that if 4 or more edges start in a vertex, i.e., the vertex is in 4 or more faces, then not every pair of faces intersect in an edge, and hence not all pairs of dark inequalities can be selected to remain dark when leaving the vertex.

Scene: *Linear functional in the space*

A linear program in the three dimensional space is a mapping that associates any point $[x, y, z]$ of the space with a real number $ax + by + cz$, where a, b and c are fixed real numbers determining the functional.

I don't know to visualize a linear functional in a space in such a way that values of the functional in *all* points of the space are represented. However, this is not necessary for our purposes. As it was in the plane, the simplex algorithm in the space operates on the polyhedron boundary only. Hence a functional will be represented by coloring any (visible) point of the boundary by a color that corresponds to the functional value in the temperature scale that was introduced in the scene dealing with a linear functional in the plane.

A new mouse function appeared on the control bar: it is possible to choose [**Change functional**]. Moving the cursor on the display with this function on rotates the virtual vector (a, b, c) (not shown) determining the functional and you can observe the corresponding changes of functional values on the visible boundary of the polyhedron. I recommend combining functional changes with rotating the polyhedron after switching the mouse function in the control bar.

The problem of linear optimization can now be formulated as search of the lightest color point of the polyhedron.

There is one technical complication when displaying an unbounded polyhedron, because points of its boundary have functional values arbitrarily closed to the plus and/or minus infinity, while the temperature scale used for visualization of functional values is bounded. The problem is solved by cutting off parts of unbounded faces and edges and displaying only a part of the boundary of the polyhedron that involves all vertices, bounded edges and bounded faces, and at least a part of any unbounded edge or face. It is clear that if functional values increase (decrease, resp.) along an unbounded edge to the point where the edge is cut, they would further increase to infinity (decrease to minus infinity, resp.).

I note again that the polyhedron is represented as hollow, because only the boundary of the polyhedron is important for the result and computation of the simplex algorithm.

Scene: *Walk along polyhedron edges*

The present scene shows a polyhedron similarly as the previous scene. However, now we will walk on the boundary of the polyhedron by moving from one vertex to another one along edges.

There is a small yellow token that sits in one vertex of the polyhedron (if it is not visible, rotate the polyhedron to make the token visible). To move the token, swich to the mouse function [**Move token**] and click on any edge incident with the vertex that hosts the token. Move the token along the polyhedron and observe the colors of the inequality backgrounds.

If the token is in a vertex, then the vertex usually satisfies three of the inequalities defining the polyhedron and hence three inequalities are dark green. When the token gets moving along an edge incident with the node, one of the inequalities starts to be satisfied as a strict inequality (and becomes light green - observe it during animation). The token moves along a line that is an intersection of the planes corresponding to two dark green inequalities, while the third originally dark green inequality gets light green and the difference between its right and left sides keeps increasing.

Let us now look to the remaining inequalities. As the token is moving, for some of them the left side decreases gets "more and more valid". Such an inequality would hold even if the token moves to the infinity. On the other hand, if there is an inequality such that its left side increases, then the difference between its right and left sides decreases, and will eventually drop to zero - it will become equality. When the first such moment occurs, it is no more possible to continue the token movement, because the token would get out of the polyhedron. This means that the token is in another vertex of the polyhedron. Note that, again, (at least) three inequalities are dark (i.e., valid as equality): two of them are those that determined the edge along which the token have moved and the third one is the inequality that has become equality when the token movement stopped.

In exceptional situations more that three edges are incident with the vertex that hosts the token (see octahedron and icosahedron in the examples). If the token is in such a vertex, 4 or more inequalities are dark green. When the token starts moving, all but two inequalities get light green. Whe the movement stops in another vertex, it could also happen that the equality would hold in more than three inequalities (more than three of them would be dark green).

Scene: *Simplex algorithm in the space*

The goal of the linear programming is to find a point or points of a given polyhedron that maximize the value of a given linear functional. In our graphical presentation this means finding the whitest point of the polyhedron surface. This scene works with bounded polyhedron, and hence the solution always exists.

Create first a polyhedron and then, usig different modes of mouse functionality, arrange it into an appropriate position and place token into a selected vertex. After that press the button [**Compute**]. Algovision performs a computation, switches to the animation mode and lets you step or run the computation of the simplex algorithm with possible steps back. It is also possible to hide the inequality panel and/or switch off displaying functional values.

The simplex algorithm works in the space in a way similar to the plane:
you choose an arbitrary starting vertex (in fact, in the real situation it might be quite difficult to find a vertex when just inequalities are given; even the question of non-emptiness of a polyhedron determined by a given set of inequalities is generally difficult, but I am not going to discuss these questions here);
keep moving the token as follows: if the functional increases along some edge starting in the vertex occupied by the token, move the token along the edge to its other end-point, otherwise the token is in the optimum vertex.

When the solution is found, the token gets red in the last step of the computation.

Scene: *Simplex algorithm - unbounded polyhedron*

The scene is the same as the previous one, but we work exclusively with unbounded polyhedron. Hence, it could happen that we run into a vertex that is incident with an unbounded edge along which the functional increases. If this happens, we see that there is no finite maximum solution of the problem.

Scene: *Simplex it yourself in the space*

The scene shows again a computatio of the simplex algorithm, but the token is not moved by Algovision, but by the user in the similar way as in the scene “Walk along polyhedron edges”. Algovision checks user commands and protests if they are illegal in the same way as in the scene “Simplex it yourself in the plane”.

2 Laborato

3 kola