The Algebraic Under Constraint Solver as a Design Tool

Hiroyuki Sawada
Mechanical Engineering Laboratory
Namiki 1–2, Tsukuba, Ibaraki 305-8564, JAPAN
Tel: +81 298 58 7130
Fax: +81 298 58 7091
E-mail: sawada@mel.go.jp

Abstract

This paper describes an algebraic under constraint solver and its feasibility as a design tool. In general, a design problem is regarded as an under constrained problem that constraints are not enough to uniquely determine the solutions. In order to support solving such a design problem, the algebraic under constraint solver provides three facilities: 1) finding out constraints inconsistent with each other, 2) obtaining new constraints, 3) calculating numerical solutions as examples. Based on results of those constraint solving, a designer goes on the design work by relaxing constraints or determining parameter values. The algebraic under constraint solver omits the possibility of looking over an existing solution or wasting time to search a solution which does not actually exist, and decreases designer's work load and improves the working efficiency.

1 Introduction

It is well known that a design problem can be regarded as a constraint satisfaction problem. Usually, it is formulated as an under constrained problem.

In general, constraints are classified into two groups: one is a group of constraints which can decrease the number of unknown variables, the other is a group of constraints which cannot. For example, an equation belongs to the former group and an inequality belongs to the latter. Usually, when the number of unknown variables is equal to that of constraints of the former group, solutions are uniquely determined, then, the obtained solutions are evaluated by using the constraints of the latter group. An under constrained problem is defined as a problem in which solutions cannot be uniquely obtained because there are not enough constraints which can decrease the number of unknown variables.

Conventionally, a constraint which cannot decrease the number of unknown variables is not used until all the variables are assigned concrete values. In the case of an under constrained problem, those concrete values are often determined by trial and error. That is, by assigning some concrete value to some variable based on human designer's intuition or experience, unknown variables are decreased one by one, then all the variables are finally assigned concrete values. Such a method, however, has an intrinsic defect. There are possibilities of looking over existing solutions and wasting time to search a solution which does not actually exist in the case that any solution cannot be allowed by the constraints which cannot decrease the number of unknown variables.

For the purpose of supporting a designer to solve a design problem, this paper proposes new methods which overcome the difficulty in under constraint problem solving by making use of constraints which cannot decrease the number of unknown variables, and realizes them as an algebraic under constraint solver. Section 2 describes facilities, architecture and algorithms of the algebraic under constraint solver. Then, in Section 3, examples of its application to design problems are shown. In this paper, it is assumed that all the constraints are given in an algebraic form.

2 Algebraic under constraint solver

2.1 Supported design stage

Design stages are roughly classified into three stages: conceptual design, embodiment design and detail design. Conceptual design determines the conceptual structure of a product. In embodiment design, for each conceptual design plan, basic variables necessary to describe product's functions are defined and concrete values are assigned to them. In detail design, all the variable values necessary for manufacturing and assembly are determined.

The algebraic under constraint solver supports embodiment design of the three design stages. It is used to evaluate constraint sets defined at the conceptual design stage.

2.2 Provided facilities

What is important for supporting embodiment design is to give a designer necessary information to determine variable values. To do that, the algebraic under constraint solver provides the following facilities and design tasks are conducted based on the results.

1. Finding out constraints inconsistent with each other
   This means to show a set of constraints which cannot be simultaneously satisfied. The designer has to relax or modify some of these constraints.

2. Obtaining new constraints
   In some cases, several inequalities can be replaced by equations. For example, if we have constraints "$x \leq 0$" and "$x \geq 0$", these constraints are replaced by "$x = 0$". This function is useful for simplifying the problem.

3. Computing several example solutions satisfying the given constraints
   This function is useful for confirming there exists a real solution. If there exists a real solution, the numerical solution is computed. Otherwise, empty is returned. The designer can determine values of design variables by referring to the example solutions. Or in order to obtain preferable solutions, the designer re-compute solutions by defining additional constraints.

2.3 Architecture of the algebraic under constraint solver

The algebraic under constraint solver consists of two parts: the symbolic part and the numeric part. Figure 1 shows the architecture of the algebraic under constraint solver. The constraint set is firstly

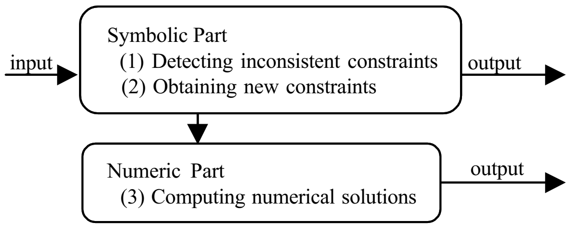

Figure 1: Architecture of the algebraic under constraint solver

received by the symbolic part. In the symbolic part, a reduction tree is constructed and the following operations are conducted:

1. Detecting constraints inconsistent with each other;

2. Obtaining new constraints.

If inconsistency is found or a new constraint is obtained, it is immediately reported to the designer. Otherwise, the constraint set is sent to the numeric part. In the numeric part, several numerical solutions are computed by making the under constrained problem result in the constraint satisfaction minimizing problem. If there exists no solution, an empty set is returned. In Section 2.4 and 2.5, algorithms used in the numeric part and symbolic part are described.

2.4 Symbolic part

The symbolic part conducts the following computations by constructing a reduction tree.

1. Finding out inequalities which cannot be satisfied simultaneously
2. Obtaining a set of inequalities which can be replaced by a set of equations

Furthermore, the symbolic part detects design variables whose values should be changed in order to remove inconsistency. Because, in embodiment design, it is quite important for a designer to know what variable values are not appropriate.

Section 2.4.1 explains the computation using a reduction tree. After that, in Section 2.4.2, the method of detecting design variables whose values should be changed is described.

2.4.1 Reduction tree

Let Eq. (1) be the constraint set defined at the conceptual design stage.

$$\begin{array}{c} f_1(x_1,\ldots,x_n)=0,\ldots,f_p(x_1,\ldots,x_n)=0, \\ g_1(x_1,\ldots,x_n)\neq 0,\ldots,g_q(x_1,\ldots,x_n)\neq 0, \\ h_1(x_1,\ldots,x_n)\geq 0,\ldots,h_r(x_1,\ldots,x_n)\geq 0. \end{array} \tag{1}$$

By introducing slack variables s_j and t_k, Eq. (1) is transformed to Eq. (2) and (3).

$$\begin{array}{c} f_1(x_1,\ldots,x_n)=0,\ldots,f_p(x_1,\ldots,x_n)=0, \\ g_1(x_1,\ldots,x_n)\cdot s_1=1,\ldots,g_q(x_1,\ldots,x_n)\cdot s_q=1, \\ h_1(x_1,\ldots,x_n)=t_1,\ldots,h_r(x_1,\ldots,x_n)=t_r. \end{array} \tag{2}$$

$$t_1\geq 0,\ldots,t_r\geq 0. \tag{3}$$

A set of Eq. (2) and (3) is equivalent to Eq. (1). The symbolic part computes equations consisting of $(t_1,\ldots,t_r)$ from Eq. (2), and finds out inconsistencies and new constraints according to the following criteria.

Criteria of inconsistency

1. *If all the coefficients of a polynomial consisting of* $(t_1,\ldots,t_r)$ *are positive and it is equal to a negative constant, there exists inconsistency.*
   For example, if "$t_1+t_2t_3=-3$" is computed, there exists inconsistency among "$h_1(x_1,\ldots,x_n)\geq 0$", "$h_2(x_1,\ldots,x_n)\geq 0$" and "$h_3(x_1,\ldots,x_n)\geq 0$".
2. *If all the coefficients of a polynomial consisting of* $(t_1,\ldots,t_r)$ *are positive and it is equal to zero, a new constraint is obtained.*
   For example, if "$t_1+t_2t_3=0$" is computed, a new constraint "$h_1(x_1,\ldots,x_n)=0\wedge\{h_2(x_1,\ldots,x_n)=0\vee h_3(x_1,\ldots,x_n)=0\}$" is obtained.

□

Figure 2 shows the geometrical meaning of this method of checking inconsistency. By introducing slack variables $\boldsymbol{s}$ and $\boldsymbol{t}$, the surface defined by Eq. (1) in $\boldsymbol{x}$-space is transformed to the surface defined by Eq. (2) and (3) in $(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$-space. The condition that there exists inconsistency in Eq. (2) and (3) means that the surface defined by Eq. (2) does not pass through the region defined by Eq. (3). Thus, by projecting the surface defined by Eq. (2) in $(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$-space to $\boldsymbol{t}$-space, the inconsistency in Eq. (2) and Eq. (3) can be checked.

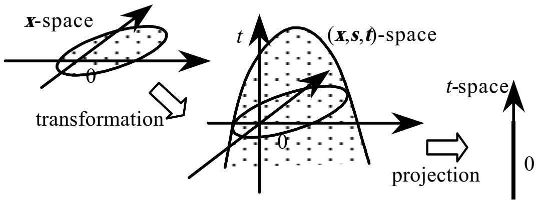

Figure 2: Geometric meaning of the inconsistency checking

Polynomials consisting of $(t_1, \ldots, t_r)$ are computed by constructing a reduction tree. At the initial state, a reduction tree is a tree whose nodes are Eq. (2). After that, each equation is regarded as a reduction rule which reduces [2] other equations, and new nodes are generated by applying the reduction rules according to the following procedures. In reduction, the slack variables $\boldsymbol{t}$ are given the lexicographically lower rank than any other variable. Figure 3 shows an example of a reduction tree.

Procedures of constructing a reduction tree

1. The slack variables $\boldsymbol{t}$ are given the lexicographically lower rank than any other variable. Let C and C' be $\{f_i(\boldsymbol{x}) = 0, g_j(\boldsymbol{x}) \cdot s_j = 1, h_k(\boldsymbol{x}) = t_k \ (1 \leq i \leq p, 1 \leq j \leq q, 1 \leq k \leq r)\}$ and $\emptyset$ respectively. Then, a tree whose leaf nodes are equations in C is constructed.

2. An equation held at each node of the reduction tree is reduced by each equation in C only once. If more than one equation in C can reduce the equation at the node, all of them are applied. Then, child nodes which hold the new equations generated by reduction are added to the node. If no equation in C can reduce the equation, it is appended to C'.

3. If an equation consisting of $\boldsymbol{t}$ is generated, it is checked by the criteria described above. If inconsistency or a new constraint is found, the computation is finished.

4. If there is a new node added to the reduction tree, go back to Step 2.

5. If C' is empty, the computation is finished. Otherwise, let C be $C \cup C'$ and go back to Step 2.

□

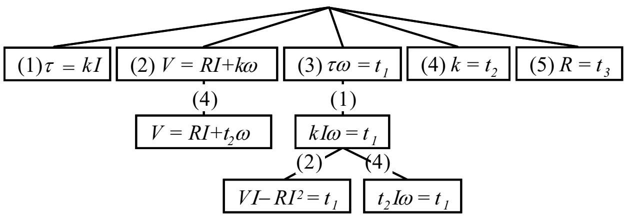

Figure 3: Reduction tree

In order to efficiently find out inconsistency, the symbolic part uses the following heuristics.

Heuristics improving computational efficiency

1. Decreasing the number of generated nodes

   If an equation held at a node can be reduced by several equations in C whose head terms are prime to each other, only one equation of them is applied and only one child node is generated.

In order to improve computational efficiency, it should be avoided to generate same equations. This heuristics is based on the fact that the normal form [2] of an equation obtained by applying equations whose head terms are prime to each other is identical irrespective of the order of using these equations.

2. Ordering between variables and normalizing the equation set

   Variables appearing in an inequality are given the lowest rank next to the slack variables $\boldsymbol{t}$*, and the equation set is normalized by computing its Gröbner base.*

   A constraint interact with another constraint which has common variables. Figure 4 shows a constraint graph [1, 3] which visually represents interrelationships between constraints. A constraint graph is defined as a bipartite graph whose nodes are classified into two groups: variable nodes and constraint nodes. A variable node is connected with constraint nodes in which the variable appears. Reductions are conducted between two constraint nodes connected with the common variable nodes. Thus, it is considered that if distances between inequality constraint nodes are short then equations consisting of $\boldsymbol{t}$ are efficiently obtained. This heuristics has an effect of transforming a constraint graph into the new one which has shorter distances between inequality constraint nodes.

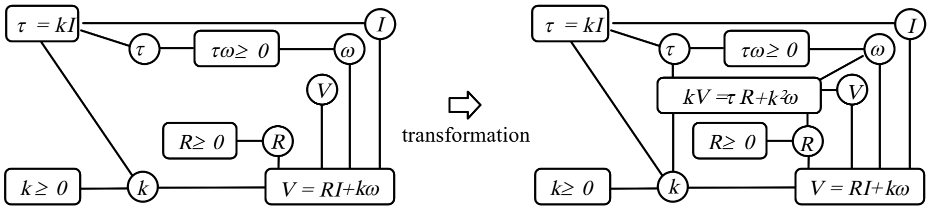

Figure 4: Constraint graph and its transformation

□

2.4.2 Design variables whose values should be changed

From the viewpoint of supporting design, detecting inconsistent inequalities is not sufficient help to a designer. Especically in embodiment design, it is quite important for a designer to know what variable values are not appropriate. This section describes the method of finding out design variables whose values should be changed.

Suppose that $c(t_1, \ldots, t_r) = 0$ is an equation showing inconsistency among inequalities. Assignment of a concrete value to a design variable is expressed as a univariate linear equation. That is, univariate linear equation "$x_k = a$" means "a is assigned to x_k". Thus, by searching for univariate linear equations used for computing $c(t_1, \ldots, t_r) = 0$, design variables whose values should be changed can be found out. If "$x_k = a$" is used to obtain $c(t_1, \ldots, t_r) = 0$, "$x_k = a$" has something to do with the inconsistency. Otherwise, "$x_k = a$" has nothing to do with it.

Let C be the whole constraint set including univariate linear equations. Obviously, $c(t_1, \ldots, t_r) = 0$ belongs to the ideal defined by C. If $c(t_1, \ldots, t_r) = 0$ does not belong to the ideal defined by $C \setminus \{x_k = a\}$, it is concluded that $x_k = a$ is used to compute $c(t_1, \ldots, t_r) = 0$. Otherwise, $x_k = a$ is not necessary to obtain $c(t_1, \ldots, t_r) = 0$. Therefore, design variables whose values should be changed are found out according to the following procedures.

Method of detecting design variables whose values should be changed

1. Let C be the whole constraint set Eq. (2), and U be a set of univariate linear equations included in C. Suppose that $c(t_1, \ldots, t_r) = 0$ is the equation which shows inconsistency among inequalities in Eq. (1). Let E be an empty set.

2. If U is empty, return E.
3. Let $x_k = a$ be the first element in U, and U be $U \setminus \{x_k = a\}$.
4. The Gröbner base G of $C \setminus \{x_k = a\}$ is computed.
5. Let c' be a normal form of $c(t_1, \ldots, t_r)$ by G. If c' is not 0, let E be $E \cup \{x_k = a\}$.
6. Go back to Step 2.

$\square$

2.5 Numeric part

The numeric part receives Eq. (2) and (3), and computes numerical solutions to it. Let R be a region defined by Eq. (2) and (3) and $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ be an objective function which has the minimum point in $(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$-space.

$$\begin{aligned} R &= \{(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t}) | f_1(\boldsymbol{x}) = 0, \ldots, f_p(\boldsymbol{x}) = 0, g_1(\boldsymbol{x}) \cdot s_1 = 1, \ldots, g_q(\boldsymbol{x}) \cdot s_q = 1, \\ &\qquad h_1(\boldsymbol{x}) = t_1, \ldots, h_r(\boldsymbol{x}) = t_r, t_1 \geq 0, \ldots, t_r \geq 0\}. \end{aligned}$$

Since all the boundaries of region R are closed, if R is not empty then the objective function $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ has its minimum value in region R. Therefore, by computing the minimum value of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ in R, a solution to Eq. (2) and (3) is obtained as the minimum point. In this way, the algebraic under constrained problem Eq. (2) and (3) results in the constraint satisfaction minimizing problem. If $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ does not have its minimum value in R, R is empty.

The minimum point of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ in R is obtained by computing its extremal points. The extremal points are calculated by finding zero-points of differential of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$. Thus, it is necessary to divide R into sub-regions in which differential of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ is continuous.

$$\begin{aligned} R &= \bigcup R_{ij}, \\ R_{ij} &= R \cap \{(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t}) | t_{a_j(i,1)} = \ldots = t_{a_j(i,j)} = 0, t_{a_j(i,j+1)} > 0, \ldots, t_{a_j(i,r)} > 0\}, \end{aligned} \tag{4}$$

where $\{a_j(i,1), \ldots, a_j(i,j)\}$ is a combination of j integers in the range of $[1, r]$ and there exist ${}_rC_j$ combinations. The suffix i is used to distinguish these combinations and changes from 1 to ${}_rC_j$.

If sub-region R_{ij} consists of a finite number of points, they are directly computed. Otherwise, stationary points of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ and singular points of R_{ij} computed.

Stationary points and singular points

- Stationary points of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$

  The stationary points of $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ are computed by Lagrange Multiplier Method. Firstly, Lagrange function $L(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\eta})$ is defined and its stationary points are calculated.

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\nu}, \boldsymbol{\eta}) &= u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t}) + \sum_{k=1}^{p} \lambda_k f(\boldsymbol{x}) + \sum_{k=1}^{q} \mu_k (g(\boldsymbol{x}) \cdot s_k - 1) \\ &\quad + \sum_{k=1}^{r} \nu_k (h(\boldsymbol{x}) - t_k) + \sum_{k=1}^{j} \eta_k t_{a(i,k)}, \end{aligned} \tag{5}$$

  where $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, $\boldsymbol{\nu}$ and $\boldsymbol{\eta}$ are Lagrange Multipliers. Next, the stationary points which satisfy $t_{a(i,j+1)} > 0, \ldots, t_{a(i,r)} > 0$ are selected.

- Singular points of R_{ij}

In sub-region R_{ij}, a point at which the $(p+q+r+j) \times (n+q+r)$ matrix of partial derivatives $J(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ has rank less than $(p+q+r+j)$ is a singular point.

$$
\begin{aligned}
J(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t}) &= \left(\frac{\partial W_k}{\partial z_l}\right)_{k=1,\ldots,p+q+r+j;\ l=1,\ldots,n+q+r}, \\
W_k &= \begin{cases} f_k(\boldsymbol{x})\ (k=1,\ldots,p), \\ g_{k-p}(\boldsymbol{x}) \cdot s_{k-p} - 1\ (k=p+1,\ldots,p+q), \\ h_{k-p-q}(\boldsymbol{x}) - t_{k-p-q}\ (k=p+q+1,\ldots,p+q+r), \\ t_{a_j(i,k-p-q-r)}\ (k=p+q+r+1,\ldots,p+q+r+j), \end{cases} \\
z_l &= \begin{cases} x_k\ (k=1,\ldots,n), \\ s_{k-n}\ (k=n+1,\ldots,n+q), \\ t_{k-n-q}\ (k=n+q+1,\ldots,n+q+r), \end{cases}
\end{aligned} \tag{6}
$$

Since a singular point is often missed when computing extremal points, it has to be searched in another method.

The condition that $J(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ has rank less than $(p+q+r+j)$ is equivalent to the condition that the row vectors of $J(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ are linearly dependent. Then, at a singular point, Eq. (7) is valid.

$$
\begin{gathered}
\exists c_1,\ldots,c_{p+q+r+j},\ \neg(c_1 = \ldots = c_{p+q+r+j} = 0) \\
c_1 \left\{ \begin{array}{c} \frac{\partial W_1}{\partial z_1} \\ \vdots \\ \frac{\partial W_1}{\partial z_{n+q+r}} \end{array} \right\} + \cdots + c_{p+q+r+j} \left\{ \begin{array}{c} \frac{\partial W_{p+q+r+j}}{\partial z_1} \\ \vdots \\ \frac{\partial W_{p+q+r+j}}{\partial z_{n+q+r}} \end{array} \right\} = \left\{ \begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \right\}.
\end{gathered} \tag{7}
$$

Therefore, by solving an equation set obtained by appending $c_k = 1$ $(k=1,\ldots,p+q+r+j)$ to Eq. (4) and (7) one by one, singular points corresponding to $c_k \neq 0$ are computed.

□

If a finite number of extremal points or singular points exist, they are directly computed. Otherwise, in order to check whether there is a point which satisfies $t_{a(i,j+1)} > 0, \ldots, t_{a(i,r)} > 0$, the obtained equation set of extremal points or singular points is regarded as a new constraint set and the above computation is recursively executed.

In the case that there exist an infinite number of extremal points, the objective function $u(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ has to be replaced by another function $u'(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$. The new objective function $u'(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ must not be constant under the condition that $u(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ is constant. Because, since the extremal points of $u(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ define a locus which makes $u(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ constant, they are also extremal points of a function which is constant when $u(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ is constant. That is, if $u'(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ is constant under the condition that $u(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ is constant, new extremal points of $u'(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ obtained by the recursive computation are completely identical with the old.

The numeric part uses the new objective function $u'(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ obtained by translation of $u(\boldsymbol{x},\boldsymbol{s},\boldsymbol{t})$ (Figure 5).

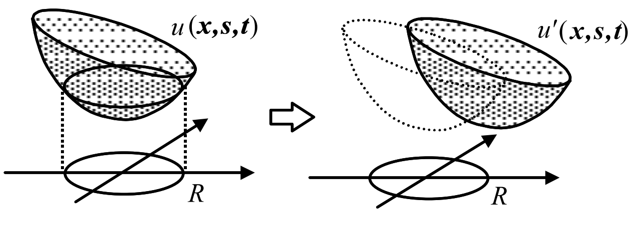

Figure 5: Replacement of the objective function

Figure 6 shows the procedures of computing numerical solutions. First of all, it is checked whether

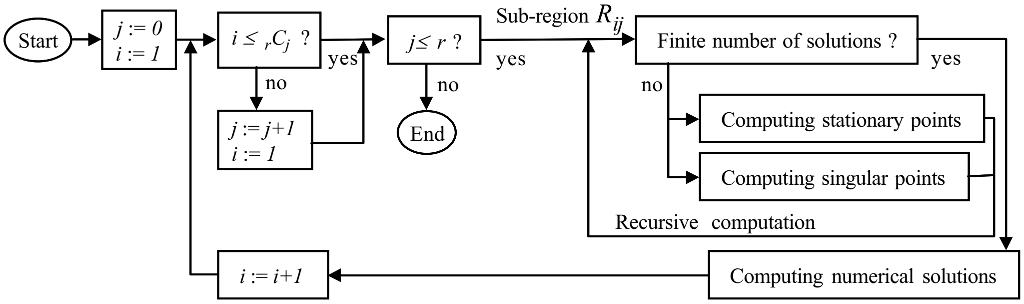

Figure 6: Procedures of computing numerical solutions

sub-region R_{ij} consists of a finite number of points. If it does, the numerical solutions are computed. Otherwise, singular point loci of R_{ij} and stationary point loci of the objective function $u(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ are computed. In the stationary point computation, a list of objective functions which have been already used is made in order to avoid applying the same function in the recursive computation. Then, it is checked whether each obtained locus have a finite number of points. The computation is recursively executed until there exist a finite number of solutions.

3 Example

This section shows two examples of application of the algebraic under constraint solver to engineering design: one is a linear-armed robot design problem, the other is a gearbox layout problem. They are shown in Section 3.1 and 3.2 respectively.

3.1 Linear-armed robot design problem

Figure 7 shows the construction of a linear-armed robot consisting of a battery, a DC motor and a ball thread. In Figure 7, f_o and v_o are the output force and velocity of the robot hand respectively.

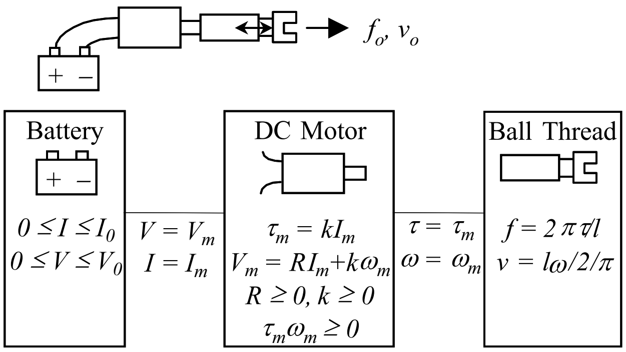

Figure 7: Construction of a linear-armed robot

Sets of constraints holding these components are given as below.

- Battery

  $0 \leq I \leq I_0$, $0 \leq V \leq V_0$.

  I: output current

  I_0: upper limit of output current

V: output voltage
V_0: upper limit of output voltage

- DC motor

  $\tau_m = kI_m,\ V_m = RI_m + k\omega_m,\ R \geq 0,\ k \geq 0,\ \tau_m\omega_m \geq 0.$

  τ_m: output torque
  ω_m: output angular velocity
  I_m: input current
  V_m: input voltage
  k: torque constant
  R: resistance

- Ball thread

  $f = 2\pi\tau/l,\ v = l\omega/2/\pi.$

  f: output force
  v: output velocity
  ω: input angular velocity
  l: lead

Battery's output voltage and current are the same as DC motor's input voltage and current, and DC motor's output torque and angular velocity are the same as ball thread's input torque and angular velocity. Furthermore, ball thread's output force and velocity are identical with the output force and velocity of the robot hand. Therefore, whole the constraints holding the linear-armed robot are given as Eq. (8).

$$\begin{array}{l} 0 \leq I \leq I_0,\ 0 \leq V \leq V_0, \\ I_m = I,\ V_m = V, \\ \tau_m = kI_m,\ V_m = RI_m + k\omega_m,\ R \geq 0,\ k \geq 0,\ \tau_m\omega_m \geq 0, \\ \tau = \tau_m,\ \omega = \omega_m, \\ f = 2\pi\tau/l,\ v = l\omega/2/\pi, \\ f_o = f,\ v_o = v. \end{array} \tag{8}$$

Let Eq. (9) be the given requirement to the linear-armed robot.

$$f_o > 90[\mathrm{N}]. \tag{9}$$

In this problem, the designer has to determine variable values satisfying Eq. (8) and (9). In actual design, the designer does not design a battery, a DC motor or a ball thread. Instead, commercial components are used. Firstly, they are roughly selected based on the size or the weight. After that, their performance are checked in detail and the acceptable components are used. In this example, for simplicity, it is assumed that the following components are used.

Battery: $I_0 = 1[\mathrm{A}],\ V_0 = 9[\mathrm{V}]$.

DC motor: $k = 0.02[\mathrm{Nm/A}],\ R = 2[\Omega]$.

Ball thread: $l/2/\pi = 1.6 \times 10^{-4}[\mathrm{m}]$.

Suppose that the designer likes to determine the output velocity of the robot hand and gives a constraint "$v_o > 0.07[\mathrm{m/s}]$" to the algebraic under constraint solver. Then, all the inequalities are transformed into a form of equation by introducing slack variables, the reduction tree is constructed, and inconsistency is found among the following constraints.

$$f_o > 90,\ v_o > 0.07,\ V \leq V_0,\ V_0 = 9\ , R = 2, k = 0.02, l/2/\pi = 1.6 \times 10^{-4}.$$

The designer may relax the constraint about the output velocity v_o and replace the constraint "$v_o > 0.07[\mathrm{m/s}]$" by a new one "$v_o > 0.06[\mathrm{m/s}]$". Then, any inconsistency is found at the symbolic part of the algebraic under constraint solver, and the following numerical solution is computed by the numeric part.

$$\begin{array}{l} f_o = f = 90.05[\mathrm{N}],\ v_o = v = 6.005 \times 10^{-2}[\mathrm{m/s}], \\ \tau = \tau_m = 1.441^{-2}[\mathrm{Nm}],\ \omega = \omega_m = 3.753 \times 10^{2}[\mathrm{rad/s}], \\ V_m = V = 8.947[\mathrm{V}],\ I_m = I = 0.7204[\mathrm{A}]. \end{array}$$

3.2 Gearbox layout problem

Figure 8 shows a three-axes gearbox. This section shows an example of the gear axes layout problem to determine the positions of gear axes for given gear diameters and size of the box.

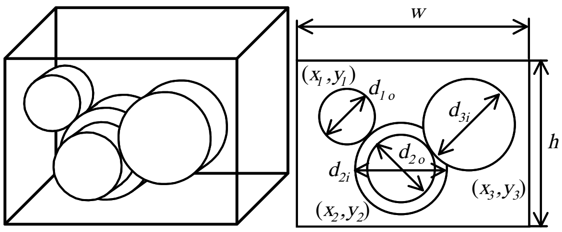

Figure 8: Three-axes gearbox

Each variable in Figure 8 shows:

- (x_j, y_j) $(j = 1, 2, 3)$: position of the jth axis
- d_{ji}, d_{jo} $(j = 1, 2, 3)$: diameter of the input and output gear of the jth axis
- w, h: width and height of the gearbox

The constraints holding the problem are given as below.

$$\begin{array}{l}(x_2 - x_1)^2 + (y_2 - y_1)^2 = ((d_{1o} + d_{2i})/2)^2, \\ (x_3 - x_2)^2 + (y_3 - y_2)^2 = ((d_{2o} + d_{3i})/2)^2, \\ (x_1 - x_3)^2 + (y_1 - y_3)^2 \geq ((d_{1o} + d_{3i})/2)^2, \\ d_{1o}/2 \leq x_1 \leq w - d_{1o}/2,\ d_{1o}/2 \leq y_1 \leq h - d_{1o}/2, \\ d_{2i}/2 \leq x_2 \leq w - d_{2i}/2,\ d_{2o}/2 \leq x_2 \leq w - d_{2o}/2, \\ d_{2i}/2 \leq y_2 \leq h - d_{2i}/2,\ d_{2o}/2 \leq y_2 \leq h - d_{2o}/2, \\ d_{3i}/2 \leq x_3 \leq w - d_{3i}/2,\ d_{3i}/2 \leq y_3 \leq h - d_{3i}/2. \end{array} \tag{10}$$

In Eq. (10), the first equation shows meshing between the first axis and the second axis gear. The second equation shows meshing between the second axis and the third axis gear. The third shows no meshing between the first axis and the third axis gear. The rest of Eq. (10) is the condition that all the gears are in the gearbox.

Let Eq. (11) be the given gear diameters.

$$d_{1o} = 42[\mathrm{mm}], d_{2i} = 86[\mathrm{mm}], d_{2o} = 70[\mathrm{mm}], d_{3i} = 110[\mathrm{mm}]. \tag{11}$$

Suppose the following gearboxes are given.

1. $w = 180[\mathrm{mm}], h = 180[\mathrm{mm}]$.

   The constraint set is firstly received by the symbolic part and inconsistency is not found. After removing redundant constraints, numerical solutions are computed by the numeric part. The following example solution is obtained.

   $$(x_1, y_1, x_2, y_2, x_3, y_3) = (97.4, 54.2, 43, 87.9, 125, 125)$$

2. $w = 160[\mathrm{mm}], h = 160[\mathrm{mm}]$.

   The constraint set is firstly received by the symbolic part and inconsistency is not found. After removing redundant constraints, numerical solutions are computed by the numeric part. However, no numerical solution is obtained and it concluded that there exists no solution which satisfies the given constraints.

4 Conclusion

This paper proposes the algebraic under constraint solver consisting of the symbolic and numeric parts and shows examples of its application to embodiment design. Each part of the algebraic under constraint solver has the following advantage and disadvantage. As shown in the examples, the symbolic part can determine the constraints inconsistent with each other. However, it may possibly look over inconsistencies. On the other hand, the numeric part does not fail to find out inconsistencies though it cannot determine which constraint is inconsistent with the others. By using these parts complementarily, it is possible to support embodiment design and improve working efficiency. The author plans to apply the algebraic under constraint solver to several actual design problems, clarify existing problems and improve its performance including computational efficiency.

References

[1] Dennis Bahler, Jim Bowen, Peter O'Grady, and Robert E. Young. Constraint networks for concurrent engineering. *Issues in design/manufacture integration*, DE–Vol.29:59–67, 1990.

[2] Bruno Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional Systems Theory*, chapter 6, pages 184–232. D. Reidel Publishing Company, 1985.

[3] Yasuo Nagai and Kenji Ikoma. Structural analysis of constraint graph by means of sparse orthogonal factorization and its application to efficient constraint processing. ICOT Technical Memorandum TM–0928, Institute for New Generation Computer Technology, 1990.