CONSYS: a dynamic constraint system

SFU-CMPT TR 1998-22

Lou Hafer

December, 1998

Abstract

CONSYS is a constraint system utility package tailored for large, sparse constraint systems with frequent addition and deletion of constraints and variables. It is specialised for use in a linear or integer programming environment. CONSYS makes use of an underlying package of vector utility functions which handle expanded and packed vectors.

1 General Structure

CONSYS is intended to support dynamic linear programming (LP) and LP-based branch-and-cut algorithms for integer linear programming (ILP) and mixed-integer linear programming (MILP). The overall design goals were to support efficient addition and deletion of constraints and variables and efficient access to the constraint system by row and by column.

CONSYS recognises two classes of constraints. *Architectural constraints* are the constraints which comprise the original problem. *Cut constraints* (cutting planes) are additional constraints generated in the course of solving an ILP or MILP problem. Overall, constraints are kept as a compact set, and each class is kept as a compact set. Specifically, if there are m_a architectural constraints and m_c cut constraints, the architectural constraints will have indices $1, \ldots, m_a$ and the cut constraints will have indices $m_a + 1, \ldots, m = m_a + m_c$. CONSYS recognises equality ($a_i x = b_i$), inequality ($a_i x \leq b_i$ and $a_i x \geq b_i$), range ($\check{b}_i \leq a_i x \leq \hat{b}_i$), and nonbinding constraint types.

CONSYS recognises two classes of variables. *Architectural variables* are the variables which are used to express the constraints. *Logical variables* (commonly separated into slack, surplus, and artificial variables) are those variables added as part of the process of solving a LP using the simplex method. Specifically, if there are m constraints and n_a architectural variables, the logical variables will have indices $1, \ldots, m$ and the architectural variables will have indices $m + 1, \ldots, n = m + n_a$. This is not the usual numbering convention (which gives architectural variables indices $1, \ldots, n_a$ and logical variables indices $n_a + 1, \ldots, n_a + m$), but it provides a fixed mapping of constraint indices to logical variable indices and it allows constant time addition and deletion of constraints and variables. CONSYS recognises continuous, general integer, and binary variable types.

Logical variables cannot be manipulated independently of constraints. Typically, a user will perform the initial load of a constraint system with logical variables disabled, then call a utility routine to create logicals for the constraints and enable them for future manipulations. When they are enabled, CONSYS automatically creates an appropriate logical variable when a constraint is added, and deletes it when the constraint is deleted. An artificial variable is used for an equality, a slack variable for a $\leq$ inequality, a surplus variable for a $\geq$ inequality, and a bounded slack variable (*i.e.*, a slack variable with a finite upper bound in addition to the usual lower bound of 0) for range constraints.

The constraint coefficient matrix is held in a row- and column-linked sparse matrix, providing equal access to the constraint system by row or by column. Access subroutines are provided, and the details of this structure are kept invisible to the outside world.

CONSYS supports a set of *attached vectors*, including upper and lower bounds for variables, objective function coefficients, right-hand-side (b) values, and constraint and variable types. These vectors are fully visible, and are assumed to have the usual vector structure. For flexibility, generic row and column vectors, with elements of arbitrary size, may be attached to the constraint system. As the constraint system grows and shrinks, these vectors are automatically reordered and, if necessary, enlarged. Row vectors are indexed from $1, \ldots, n$ and column vectors are indexed from $1, \ldots, m$.

CONSYS relies on the existence of a complementary package of vector utilities which provide support for a packed vector type as well as additional support for the standard C vector type (also called an expanded vector, to distinguish it from the packed vector type).

This document provides an overview of the organisation and algorithms of CONSYS. For specific details, consult the source files consys.h, consys_utils.c, consys_mathutils.c, and consys_io.c.

2 Addition and Deletion of Constraints and Variables

To implement constant-time addition and deletion of constraints and variables, CONSYS moves individual constraints or variables instead of entire blocks.

In a system with n_a architectural variables, addition of a new architectural variable simply requires the addition of the new column $n_a + 1$.

In a system with n_a architectural variables, deletion of architectural variable n_a simply requires the deletion of the column. Deletion of an architectural variable with index $k < n_a$ is handled by moving the architectural variable with index n_a to occupy index position k. At most one variable changes index, regardless of the position of the deleted variable.

Logical variables cannot be added or deleted independently, as mentioned above.

In a system with no cut constraints, addition of a constraint when logical variables are disabled implies the same set of actions used to add and delete architectural variables. Specifically, a constraint is added at the end of the index set of the existing constraints. When a constraint is deleted, the hole in the index set is filled by moving the last constraint to occupy the hole.

Now consider a system with m_a architectural constraints $1, \dots, m_a$ and m_c cut constraints $m_a + 1, \dots, m = m_a + m_c$. To add an architectural constraint, the cut constraint with index $m_a + 1$ is moved to index $m + 1$ and the new architectural constraint occupies index $m_a + 1$. To delete an architectural constraint with index $k < m_a$, the architectural constraint with index m_a is moved to index k and the cut constraint with index m is moved to index m_a. If the deleted architectural constraint occupied index position m_a, only the cut constraint is moved from m to m_a. This sequence requires at most two constraints to move independent of the index of the deleted constraint, and retains the properties that architectural and cut constraints occupy compact index sets and that the constraints as a whole occupy a compact index set.

Addition or deletion of a cut constraint requires motion only within the set of cut constraints; the architectural constraints are not affected. The architectural constraints occupy the first positions of the index set since they are assumed to be comparatively stable.

When logical variables are enabled, the addition or deletion of a constraint (architectural or cut) implies a corresponding addition or deletion of a logical variable. This is handled automatically by CONSYS, in a manner analogous to the addition and deletion of architectural constraints.

To follow the most complex cases, first consider the addition of an architectural constraint at index position $m_a + 1$. The architectural variable at index $m + 1$ is moved to $n + 1$, then the logical variable associated with the cut constraint occupying $m_a + 1$ is shifted to position $m + 1$ (matching the shift of the cut constraint), and finally the logical variable for the new architectural constraint is created at index $m_a + 1$ (matching the creation of the constraint).

To delete a constraint at index $k < m_a$, the architectural constraint at index m_a is moved to k and the associated logical at index m_a is moved to k. The cut constraint at position m is moved to m_a, and its logical is moved from index m to m_a. Finally, the architectural variable at index n is moved to index m.

The worst case requires the movement of two constraints and three variables, independent of the size of the constraint system, and maintains the property that index sets for variables and constraints are compact.

To see why logical variables must precede architectural variables in the index order, consider adding an architectural variable in a system where architectural variables have indices $1, \dots, n_a$ and logical variables have indices $n_a + 1, \dots, n = n_a + m$. We would need to move the logical at index $n_a + 1$ to index $n_a + m + 1$ in order to make a space for the new architectural variable at $n_a + 1$. But, if we do not move the constraint at index 1 to index $m + 1$, we lose the invariant relation of logical indices to constraint indices. If we do move the constraint, we're left with a hole at index 1. Neither situation is acceptable.

3 The Constraint System Header

The constraint system header structure is visible to the user, and contains various statistics and general information, pointer fields for a number of vectors often attached to the constraint system, and a few fields that should be regarded as opaque data structures. A name can be assigned to the constraint system, and there is a flag field to specify options (presence/absence of logical variables, and requests for warnings).

Counts of total, architectural, logical, integer, and binary variables are kept in the header, and counts of total, architectural, and cut constraints.

The maximum capacity of the constraint system is kept in colsze and rowsze. The maximum lengths actually present in the constraint system are kept in maxrowlen and maxcollen, with corresponding indices maxrowndx and maxcolndx.

Provision is made to install the objective function as an equality constraint, with a free variable x_z, as $cx - x_z = 0$, in anticipation that this might be useful as a cut. The index of the constraint is kept in objndx, and the index of the variable x_z is kept in xzndx. These indices are automatically updated if either the constraint or variable are moved due to changes in the constraint system.

4 Attached Vectors

The implementation of CONSYS reorders constraints and variables as described in §2; this implies a corresponding reordering of the elements of row and column vectors associated with the constraint system (objective and righ-hand-side coefficients, variable bounds, *etc.*). In order that this be as transparent as possible to the user, CONSYS provides the ability to *attach* vectors to the constraint system. Attached vectors are automatically reordered and expanded whenever necessary to follow changes to the constraints or variables. Attached vectors can be either row or column vectors. The size of an element in a vector can be specified at the time it is attached.

CONSYS has specific flags for the presence of an attached objective function, a right-hand-side vector, a range constraint right-hand-side vector, upper and lower bound vectors for variables, upper and lower bound vectors for the left-hand-side of constraints, a variable type vector, and a constraint type vector. There is a dedicated pointer field in the constraint header for each of these vectors.

To attach a vector, the user gives the address of a pointer to the vector. Attachment is required even if the pointer being loaded is one of the dedicated fields in the constraint system header. The address of the pointer is required so that CONSYS can update the pointer if the vector is reallocated due to expansion of the constraint system. Multiple pointers to the same vector are allowed. If the pointer is null, CONSYS will allocate a vector with the appropriate number of elements and return it in the pointer. All vectors are initialised to 0 with the exception of a vector flagged as an upper bound vector for variables, which is initialised to $+\infty$.

While CONSYS will reorder attached vectors, further user action is required to update entries in the common case where the elements of the vector store constraint or variable indices (*e.g.*, a simplex basis).

5 CONSYS Utility Routines

5.1 Constraint System Creation and Deletion

consys_struct *consys_create (char *nme, flags parts, flags opts, int concnt, int varcnt)

Creates an empty constraint system with the specified capacity. Attached vectors of the appropriate size will be created as specified in parts.

void consys_free (consys_struct *consys)

Deallocates a constraint system.

bool consys_realloc (consys_struct *consys, char rowcol, int incr)

Increases the maximum row or column capacity of a constraint system.

5.2 Attaching and Detaching Vectors

bool consys_attach (consys_struct *consys, flags what, int elsze, void **pvec)

Attaches a vector to consys, of the type given in what, with element size elsze, using the pointer specified by pvec. If *pvec is null, a vector of the appropriate size is allocated and initialised.

bool consys_detach (consys_struct *consys, void **pvec, bool all)

Detaches the reference specified by pvec. If all is true, all references to *pvec are detached. Note that pvec is not cleared or examined in any way.

bool consys_update (consys_struct *consys, void *old, void *new)

Changes all references to the attached vector old to point to new. It is the caller's responsibility to make sure that some pointer to old remains.

5.3 Adding and Deleting Constraints and Variables

bool consys_addcol_pk (consys_struct *consys, vartyp_enum vartyp, pkvec_struct *pkcol, double obj, double vlb, double vub)

Adds an architectural variable (column) to consys. The type of variable is specified by vartyp, the name and coefficients of the column are given in the packed vector pkcol, and values may be supplied for the objective function coefficient and upper and lower bounds. To create an empty column, use a packed vector with an allocated coefficient capacity of 0 (*i.e.*, pkcol->sze = 0). The index assigned to the column is returned in pkcol.

bool consys_addrow_pk (consys_struct *consys, char class, contyp_enum contyp, pkvec_struct *pkrow, double rhs, double rhslow, conbnd_struct *cub, conbnd_struct *clb)

Adds a constraint (row) to consys. The class (architectural or cut) is specified by class and the type by contyp. The name and coefficients are given in the packed vector pkrow. Values for the rhs, range rhs, and upper and lower bounds on the left-hand-side may be supplied. To create an empty row, use a packed vector with an allocated coefficient capacity of 0 (*i.e.*, pkrow->sze = 0). The index assigned to the constraint is returned in pkrow.

bool consys_getcol_pk (consys_struct *consys, int colndx, pkvec_struct **pkvec)

Retrieves the column specified by colndx in the vector specified by *pkvec. If *pkvec is null, a vector of the appropriate capacity is allocated. If pkcol->sze = 0, only the column header information is retrieved.

bool consys_getcol_ex (consys_struct *consys, int colndx, double **vec)

As for consys_getcol_pk, but the column is retrieved as an expanded vector. There is no option to retrieve only the header information.

bool consys_getrow_pk (consys_struct *consys, int rowndx, pkvec_struct **pkvec)

As for consys_getcol_pk, but retrieves the specified row. If pkrow->sze = 0, only the column header information is retrieved.

bool consys_getrow_ex (consys_struct *consys, int rowndx, double **vec)

As for consys_getrow_pk, but the row is retrieved as an expanded vector. There is no option to retrieve only the header information.

bool consys_delcol (consys_struct *consys, int colndx)

Deletes the specified variable (column).

bool consys_delrow (consys_struct *consys, int rowndx)

Deletes the specified constraint (row).

bool consys_logicals (consys_struct *consys)

Generates logical variables for consys.

5.4 Mathematical Utilities

double consys_dotcol (consys_struct *consys, int colndx, double *vec)

Calculates the dot product of the expanded vector specified by vec with the column specified by colndx.

double consys_dotrow (consys_struct *consys, int rowndx, double *vec)

Calculates the dot product of the expanded vector specified by vec with the row specified by rowndx.

double consys_1normrow (consys_struct *consys, int rowndx)

Calculates the 1-norm $\|vec_j\|_1 = \sum_j |vec_j|$ of the row specified by rowndx.

double consys_2normrow (consys_struct *consys, int rowndx)

Calculates the 2-norm $\|vec\|_2 = \left(\sum_j vec_j^2\right)^{\frac{1}{2}}$ of the row specified by rowndx.

double consys_ssqrow (consys_struct *consys, int rowndx)

Calculates the sum of squares $\sum vec_j^2$ of the row specified by rowndx. On occasion this is more useful than $\|vec\|_2$ and it saves the computation and loss of accuracy arising from calculating $\|vec\|_2^2$.

double consys_infnormrow (consys_struct *consys, int rowndx)

Calculates the ∞-norm $\|vec_j\|_\infty = \max_j |vec_j|$ of the row specified by rowndx.

double consys_1normcol (consys_struct *consys, int colndx)

As consys_1normrow, for the column specified by colndx.

double consys_2normcol (consys_struct *consys, int colndx)

As consys_2normrow, for the column specified by colndx.

double consys_ssqcol (consys_struct *consys, int colndx)

As consys_ssqrow, for the column specified by colndx.

double consys_infnormcol (consys_struct *consys, int colndx)

As consys_infnormrow, for the column specified by colndx.

double consys_mulrow (consys_struct *consys, int rowndx, double scalar)

Multiplies the row specified by rowndx by scalar.

5.5 I/O Utilities

char *consys_prtvartyp (vartyp_enum vartyp)

Returns a pointer to a string representation of the variable type.

char *consys_prtcontyp (contyp_enum contyp)

Returns a pointer to a string representation of the constraint type.

char *consys_assocnme (consys_struct *consys, flags which)

Returns a pointer to a string representation of the associated vector, based on the type of the vector as specified in which. If consys is non-null, the name of the constraint system is used as a prefix. The string representation is constructed in an internal static buffer.

char *consys_conbndnme (char bndlett, int cndx, conbnd_struct *bnd)

Returns a pointer to a string representation of the name of a constraint bound. The string representation is constructed in an internal static buffer.

char *consys_conbndval (conbnd_struct *bnd)

Returns a pointer to a string representation of the value of a constraint bound. The string representation is constructed in an internal static buffer.

char *consys_nme (consys_struct *consys, char cv, int ndx, bool pfx, char *clientbuf)

Returns a pointer to a string representation of the name of the constraint or variable specified by cv and ndx. If pfx is false, the pointer is nonvolatile. If pfx is true, the name of the constraint system is used as a prefix, and the string representation is constructed in clientbuf (if it is non-null) or in an internal static buffer.

Note that routines which use an internal static buffer to construct a string produce a volatile result in the sense that each call overwrites the result of the previous call.

6 Compilation Options

CONSYS recognises two conditional compilation options, PARANOIA and NDEBUG.

NDEBUG enables row and column index parameter range error checking, warnings for zero length rows and columns, and warnings if a repeat attempt is made to attach a vector. This provides a useful level of casual error checking.

PARANOIA invokes numerous checks during any manipulations of the data structure. None of the individual checks are particularly costly, but taken together they impose a substantial overhead. This option can be defined when doing serious debugging or code development, to detect parameter errors and inconsistency in the data structures of CONSYS. For best effect, NDEBUG should also be defined.

7 Packed Vectors

The packed vector type used by CONSYS consists of a header structure and an array of nonzero coefficients stored as (index, value) pairs. For a column vector, the header contains the column index, a field which can be used to hold a pointer to the name of the column, and bookkeeping information. Each element of the coefficient array contains a row index and a value. Row vectors are analogous, with the appropriate interchange of row and column indices in the data structure.

The utility routines pkvec_new and pkvec_free take care of allocating and freeing the space occupied by the vector header and the coefficient array. A new vector is assigned a pointer to a static name string; this pointer can be simply overwritten. The user is responsible for allocating and releasing any space associated with a user-supplied name string.

For details, consult vector.h and vector_utils.c.

7.1 Packed Vector Utility Routines

pkvec_struct *pkvec_new (int sze)

Allocates a packed vector with capacity for a maximum of sze entries in the coefficient array.

bool pkvec_resize (pkvec_struct *pkvec, int sze)

Increases the maximum capacity of the coefficient array to sze.

void pkvec_free (pkvec_struct *pkvec)

Destroys the pkvec passed as a parameter.

7.2 Expanded Vector Mathematical Routines

double exvec_1norm (double *vec, int len)

Calculates the 1-norm $\|vec_j\|_1 = \sum_j |vec_j|$ of vec.

double exvec_2norm (double *vec, int len)

Calculates the 2-norm $\|vec_j\|_2 = \left(\sum_j vec_j^2\right)^{\frac{1}{2}}$ of vec.

double exvec_ssq (double *vec, int len)

Calculates the sum of squares $\sum vec_j^2$ of vec.

double exvec_infnorm (double *vec, int len)

Calculates the ∞-norm $\|vec_j\|_\infty = \max_j |vec_j|$ of vec.