COCONUT Deliverable D13

Upgraded State of the Art Techniques Implemented as Modules

July 29, 2003

ILOG
TU Darmstadt
University of Coimbra
University of Vienna
University of Nantes
Catholic University of Louvain
Swiss Federal Institute of Technology

CHAPTER 6

UWien Basic Splitter

H. SCHICHL
University of Vienna, Department of Mathematics
A-1090 Wien, Austria
email: Hermann.Schichl@esi.ac.at

1. Copyright

2. Module name

Module name is `Basic Splitter`

3. Type of module

`Basic Splitter` is a splitter.

4. Description of the module

`Basic Splitter` is a module which computes useful splitting coordinates and splitting points for the model contained in the work node.

5. Limits

There are no hardcoded limits.

6. Service *split*

`Basic Splitter` computes a series of splits, each consisting of a split coordinate and one or more split points in this coordinate.

If there are Lagrange multipliers available, the slope s of the Lagrangian in the original variables x is computed using these multipliers. Then the variable with highest $\mathrm{width}(s_i)\widehat{\mathrm{width}}(x_i)$ is chosen as splitting coordinate. If there are no multipliers, the widest coordinate is chosen.

If there are exclusion boxes available, $\widehat{\mathrm{width}}(x_i)$ is computed as $\mathrm{width}(x_i) - \mathrm{width}(b_i)$ where $\mathbf{b}$ is the exclusion box.

If the exclusion box is comparatively big (its volume is bigger than `excl_ratio` times the volume of the variable box), then the box is trisected along $\underline{\mathbf{b}_i}$ and $\overline{\mathbf{b}_i}$. Otherwise the box is bisected along one of these coordinates, depending on when the resulting boxes are more similar in size.

If there is no exclusion box, the box is bisected in the middle (geometric mean) or at 0, if $\mathbf{b}_i$ contains 0. If $\mathbf{b}_i$ is unbounded, then the splitting coordinate is either chosen using an evaluation region (trisect at the bound of this evaluation region) or at the signed square of the finite bound.

6.1. Kind of input. Any work node.

6.2. Output provided. The code returns with a number `split_delta`s each representing a possible and useful split.

There is no useful statistical information collected, the efficiency is always 1.

6.3. Control parameters.

- `excl_ratio`: This parameter is of type `double` and specifies the ratio of exclusion box size vs. box size at which the exclusion box coordinates are used for trisection instead of bisection. The default is `1`.
- `number of splits`: With this parameter of type `int` the number of alternative splits is specified. The default is `1`.
- `multiplier`: Using this `vector<double>` the user can specify an estimate for the Lagrange multipliers. These are then use as weights for the constraints. Default: no multiplier.

6.4. Termination reason. There is no termination reason except `'SUCCESS'` (0), `'no valid split coordinate'` (-2), and `'NO MODEL DEFINED'` (-1).

7. Statistical information collected

No useful statistical information is collected.

8. Service *hard variables*

`Basic Splitter` uses the split finding algorithms to determine the variables which are most likely to be split, and therefore can be considered hard.

8.1. Kind of input. Any work node.

8.2. Output provided. The code returns a list of variables in the `information` entry with index `hard variables` as a `vector<unsigned int>`.

There is no useful statistical information collected, the efficiency is always 1.

8.3. Control parameters.

`excl_ratio`: This parameter is of type `double` and specifies the ratio of exclusion box size vs. box size at which the exclusion box coordinates are used for trisection instead of bisection. The default is `1`.

`number of splits`: With this parameter of type `int` the number of alternative splits is specified. The default is `1`.

`multiplier`: Using this `vector<double>` the user can specify an estimate for the Lagrange multipliers. These are then use as weights for the constraints. Default: no multiplier.

8.4. Termination reason. There is no termination reason except `'SUCCESS'` (0), `'no valid split coordinate'` (-2), and `'NO MODEL DEFINED'` (-1).

9. Statistical information collected

No useful statistical information is collected.

CHAPTER 8

UWien Best Point

H. SCHICHL
University of Vienna, Department of Mathematics
A-1090 Wien, Austria
email: Hermann.Schichl@esi.ac.at

1. Copyright

2. Module name

Module name is `Best Point`

3. Type of module

`Best Point` is a constraint generator and info engine.

4. Description of the module

`Best Point` is a module which manages the best point found and the upper bound on the objective function.

5. Limits

There are no hardcoded limits.

6. Service *update*

`Best Point` finds the best point in the `point` table of the search database and updates the upper bound of the objective function according to its objective function value.

6.1. Kind of input. Any work node.

6.2. Output provided. The code returns a `bound_delta` changing the bounds on the objective function.

There is no useful statistical information collected, the efficiency is always 1.

6.3. Control parameters.

`use unverified`: This parameter is of type `bool` and specifies whether unverified points may be used to reduce the range of the objective function. The default is `false`.

6.4. Termination reason. There is no termination reason except `'SUCCESS'` (0), `'no valid point'` (-1), and `'NO MODEL DEFINED'` (-2).

7. Statistical information collected

No useful statistical information is collected.

8. Service *find bound*

`Best Point` finds the best point in the `point` table of the search database and returns its objective function value.

8.1. Kind of input. Any work node.

8.2. Output provided. The code returns an information entry `upper bound` containing the obective function value of the best point found.

There is no useful statistical information collected, the efficiency is always 1.

8.3. Control parameters.

`use unverified`: This parameter is of type `bool` and specifies whether unverified points may be used to reduce the range of the objective function. The default is `false`.

8.4. Termination reason. There is no termination reason except `'SUCCESS'` (0), `'no valid point'` (-1), and `'NO MODEL DEFINED'` (-2).

9. Statistical information collected

No useful statistical information is collected.

10. Service *set*

`Best Point` sets the upper bound of the objective function to a specified value.

10.1. Kind of input. Any work node.

10.2. Output provided. The code returns a `bound_delta` changing the bounds on the objective function.

There is no useful statistical information collected, the efficiency is always 1.

10.3. Control parameters.

`upper bound`: This parameter is of type `double` specifies the upper bound of the objective function.

10.4. Termination reason. There is no termination reason except `'SUCCESS'` (0) and `'NO MODEL DEFINED'` (-2).

CHAPTER 9

UWien Check Box

H. SCHICHL
University of Vienna, Department of Mathematics
A-1090 Wien, Austria
email: Hermann.Schichl@esi.ac.at

1. Copyright

2. Module name

Module name is `Check Box`

3. Type of module

`Check Box` is a checker.

4. Description of the module

`Check Box` is a module which checks whether a box can be considered tiny.

5. Limits

There are no hardcoded limits.

6. Service *check*

`Check Box` tests whether the box represented by the work node is tiny in all components.

6.1. Kind of input. Any work node.

6.2. Output provided. The return value is passed as termination reason. 100 means that the box is tiny, 0 means that it is not tiny.

There is no useful statistical information collected, the efficiency is always 1.

6.3. Control parameters.

`eps`: This parameter is of type `double` and specifies the maximal allowed width in every node. Default is `1.0e-6`.

`scale`: Using this `vector<double>` the user can specify a scaling for all nodes to determine a scaled width. Default: all ones vector.

6.4. Termination reason. There is no termination reason except `'SUCCESS'` (0 or 100).

7. Statistical information collected

No useful statistical information is collected.

CHAPTER 10

UWien Check Infeasibility

H. SCHICHL
University of Vienna, Department of Mathematics
A-1090 Wien, Austria
email: Hermann.Schichl@esi.ac.at

1. Copyright

2. Module name

Module name is `Check Infeasibility`

3. Type of module

`Check Infeasibility` is a checker.

4. Description of the module

`Check Infeasibility` is a module which checks whether a model has been detected infeasible.

5. Limits

There are no hardcoded limits.

6. Service *check*

`Check Infeasibility` tests whether the model in the work node has been detected infeasible.

6.1. Kind of input. Any work node.

6.2. Output provided. The return value is passed as termination reason. 100 means that the model is infeasible, 0 means that it is not infeasible.

There is no useful statistical information collected, the efficiency is always 1.

6.3. Control parameters. None.

6.4. Termination reason. There is no termination reason except `'SUCCESS'` (0 or 100).

7. Statistical information collected

No useful statistical information is collected.

CHAPTER 11

UWien Check Number

H. SCHICHL
University of Vienna, Department of Mathematics
A-1090 Wien, Austria
email: Hermann.Schichl@esi.ac.at

1. Copyright

2. Module name

Module name is `Check Number`

3. Type of module

`Check Number` is a checker.

4. Description of the module

`Check Number` is a module which checks whether a model fulfills certain requirements.

5. Limits

There are no hardcoded limits.

6. Service *check*

`Check Number` checks whether the constraints and the objective function fulfill certain requirements.

6.1. Kind of input. Any work node.

6.2. Output provided. The return value is passed as termination reason. 100 means that the requirements are fulfilled, 0 means that they are not.

There is no useful statistical information collected, the efficiency is always 1.

6.3. Control parameters. There should be one comparison parameter set to an `int`, either `">="`, `"<="`, `">"`, `"<"`, or `"="`. If none is specified, `">="` is chosen.

Setting, e.g., `"<="` to 7 means that there must not be more than 7 constraints of the type specified by the other parameters.

`type`: This parameter is of type `unsigned int` specifies the semantics bitmap as defined in `expression.h`.

The `bool` parameters can be used to turn on, or off all expression types, as specified in `expression.h`.

```
bound: ex_bound
linear: ex_linear
quadratic: ex_quadratic
polynomial: ex_polynomial
other: ex_other
kj: ex_kj
org: ex_org
redundant: ex_redundant
restrictive: ex_restrictive
active: ex_active
inactive: ex_inactive
integer: ex_integer
exists: ex_exists
forall: ex_forall
free: ex_free
stochastic: ex_stochastic
convex: ex_convex
concave: ex_concave
equality: ex_equality
leftbound: ex_leftbound
rightbound: ex_rightbound
atmlin: ex_atmlin
atmquad: ex_atmquad
atmpoly: ex_atmpoly
nonlin: ex_nonlin
nonbnd: ex_nonbnd
any: ex_any
bothbound: ex_bothbound
```

6.4. Termination reason. There is no termination reason except `'SUCCESS'` (0 or 100).

7. Statistical information collected

No useful statistical information is collected.

CHAPTER 14

TU Darmstadt module DONLP2-INTV

1. Name

`donlp2_intv`.

2. Module Type

nonlinear local optimizer

3. Description

`donlp2_intv` is a general purpose nonlinear optimizer for continuous variables. It is intended for the minimization of an (in general nonlinear) differentiable real function f subject to (in general nonlinear) inequality and equality constraints . Specifically the problem is accepted in the following form:

$$\begin{aligned} f(x^*) &= \min\{f(x) : \quad x \in \mathcal{S}\} \\ \mathcal{S} &= \{x \in \mathbb{R}^n : \quad x_u \le x \le x_o , \\ & \qquad\qquad\qquad b_u \le Ax \le b_o , \\ & \qquad\qquad\qquad c_u \le c(x) \le c_o\} . \end{aligned}$$

Here A is a matrix of dimension `nlin`$\times$`n` and c is a vectorvalued function of dimension `nonlin`. The lower and upper bounds $x_u,\ x_o,\ b_u,\ b_o,\ c_u,\ c_o$ may have components equal to $-\infty$ resp. ∞ and, if finite, may also be equal, indicating an equality constraint. This applies also for x_u and x_o, $(x_o)_i = (x_u)_i$ indicating a fixed variable. Since there is no internal preprocessor, in that case an explicit equality constraint is used internally. Under weak assumptions on the problem one can show that the solver determines a point satisfying the first order optimality conditions from arbitrary initial guesses.

4. Dependencies

None

5. Overall limit size

Unknown

6. Services

6.1. 1 Name: `solve`

Input:

(1) *Type*: `const gptr<work_node>&`.
Meaning: pointer to a work node

Output: The code returns with an approximate local minimizer, the multipliers from the multiplier rule and the corresponding function values. It also collects some statistical information and gives an indicator of the problems sensitivity.

Description: `donlp2` computes a local solution to a nonlinear programming problem satisfying the first order necessary optimality conditions.

Control Parameters:

(1) *Name*: `tau0`
Type: `double`
Description: bound for constraint violation during run
Default value: `1.0`

(2) *Name*: `del0`
Type: `double`
Description: bound for considering an inequality as active
Default value: `1.0`

(3) *Name*: `delmin`
Type: `double`
Description: tolerated infeasibility in solution
Default value: `1.0e-6`

(4) *Name*: `epsx`
Type: `double`
Description: tolerated error in multiplier rule
Default value: `1.0e-5`

(5) *Name*: `cold`
Type: `bool`
Description: cold start indicator
Default value: `true`

(6) *Name*: `nreset`
Type: `int`
Description: tolerated number of small inefficient steps
Default value: `n`

Reason for termination:

Integer: from –16 to –13 and from –9 to 3
String:

- `'CONSTRAINT EVALUATION RETURNS ERROR WITH CURRENT POINT'` (–9)
- `'OBJECTIVE EVALUATION RETURNS ERROR WITH CURRENT POINT'` (–8)
- `'QPSOLVE: EXTENDED QP-PROBLEM SEEMINGLY INFEASIBLE '` (–7)

* `'QPSOLVER: NO DESCENT FOR INFEAS FROM QP FOR TAU=TAU_MAX'` (–6)
* `'QPSOLVER: ON EXIT CORRECTION SMALL, INFEASIBLE POINT'` (–5)
* `'STEPSIZESELECTION: COMPUTED D FROM QP NOT A DIR. OF DESCENT'` (–4)
* `'MORE THAN MAXIT ITERATION STEPS'` (–3)
* `'STEPSIZESELECTION: NO ACCEPTABLE STEPSIZE IN [SIGSM,SIGLA]'` (–2)
* `'SMALL CORRECTION FROM QP, INFEASIBLE POINT'` (–1)
* `'KT-CONDITIONS SATISFIED, NO FURTHER CORRECTION COMPUTED'` (=0)
* `'COMPUTED CORRECTION SMALL, REGULAR CASE '` (1)
* `'STEPSIZESELECTION: X ALMOST FEASIBLE, DIR. DERIV. VERY SMALL'` (2)
* `'KT-CONDITIONS (RELAXED) SATISFIED, SINGULAR POINT'` (3)
* `'VERY SLOW PRIMAL PROGRESS, SINGULAR OR ILLCONDITONED PROBLEM'` (–16)
* `'MORE THAN NRESET SMALL CORRECTIONS IN X '` (–15)
* `'CORRECTION FROM QP VERY SMALL, ALMOST FEASIBLE, SINGULAR '` (–14)
* `'NUMSM SMALL DIFFERENCES IN PENALTY FUNCTION,TERMINATE'` (–13)

Description: For the explanation of the status values, see the user guide.

7. Input

Description: Given a work node the code evaluates the model DAG and extracts the necessary information from this. This is individual information about the coefficients of the linear constraints, the bounds on the variables and the constraints and the values of the objective function and the nonlinear constraints as well as their gradients.

The model is required to represent a differentiable function.

8. Output

Description: The code returns with an approximate local minimizer, the multipliers from the multiplier rule and the corresponding function values. It also collects some statistical information and gives an indicator of the problems sensitivity.
Description of non-standard delta's: None

9. Statistics

Description:

(1) `num_of_iter` : number of steps total .
(2) `num_of_f` : number of objective function evaluations .
(3) `num_of_gradf`: number of objective gradient evaluations.
(4) `num_of_con` : number of nonlinear constraint evaluations.
(5) `num_of_gradcon`: number of nonlinear constraints gradients evaluations.
(6) `num_of_restarts`: number of restarts for the quasi Newton Hessian update.
(7) `num_of_full_upd`: number of quasi Newton updates without modification.
(8) `num_of_upd` : number of all quasi Newton updates.
(9) `num_of_full_sqp` : number of steps with problem regularization through addition of slacks.

CHAPTER 15

UWien Exclusion boxes using Karush-John conditions

1. Name

`Exclusion box KJ`

2. Module Type

Annotation generator

3. Description

`Exclusion box KJ` computes two boxes B_i and B_x which have the following property: B_i contains at least one (or exactly one) Karush-John point for the presented model, $B_i \subset B_x$, and $B_x \backslash B_i$ does not contain a Karush-John point.

Given an approximate Karush-John point z we iteratively produce boxes B_r centered at z with radius r. In these boxes we compute the interval slope $F[z, B_r]$ for all equality constraints, an approximate midpoint inverse $C(r)$, and the matrix $H(r) = F[z, B_r]C(r)$. Then we find a zero $\tilde{r}$ for $f(r) = \|H(r) - I\| - 1$ using a secant method, and set $B_x = B_{\tilde{r}}$.

B_i for uniqueness is computed by using the same test with the interval derivative $F'[\mathbf{x}]$ inside $\frac{1}{2}B_r$ and applying the Krawczyk operator iteratively.

4. Dependencies

None

5. Overall limit size

Unknown

6. Services

6.1. 1 Name: `generate`

Input:

(1) *Type*: Any work node containing the Karush-John conditions and an (approximate) Karush-John point.
Meaning: None

Output: The code returns an `exclusion_box_delta` containing B_x and a `uniqueness_area_delta` containing B_i.
Description: This service generates exclusion regions.
Control Parameters:

(1) *Name*: `exclusion_only`
Type: `bool`
Description: Only construct an exclusion box B_x.
Default value: `false`

(2) *Name*: `with_uniqueness`
Type: `bool`
Description: Construct B_i using interval derivatives such that uniqueness of the Karush-John point is guaranteed.
Default value: `true`

Reason for termination:

Integer: 0 or –1
String:
- ∗ "SUCCESS"
- ∗ "NO MODEL DEFINED"

Description: None

7. Statistics

Description: The ratios of the radii $\mathrm{rad}(B_x)/\mathrm{rad}(B_i)$, and of the radii $\mathrm{rad}(\mathbf{x})/\mathrm{rad}(B_x)$.

CHAPTER 17

UWien Karush–John conditions generator

1. Name

`KJ generator`

2. Module Type

Constraint generator

3. Description

`KJ generator` computes a directed acyclic graph representing all operations and constraints needed to code the Karush–John conditions.

Consider the optimization problem

$$\begin{aligned}
&\min f(x)\\
&\text{s.t.}\quad A_B x \in \mathbf{b_B},\quad A_L x \geq b_L,\quad A_U x \leq b_U,\quad A_E x = b_E,\\
&\quad F_B(x) \in \mathbf{F_B},\quad F_L(x) \geq F_L,\quad F_U(x) \leq F_U,\quad F_E(x) = F_E
\end{aligned} \tag{4}$$

If this problem is represented in the work node passed, the KJ generator produces the corresponding Karuhsh–John conditions. In the generated DAG, for every new variable (Lagrange multiplier) and every new node the `ex_kj` flag is set.

The KJ conditions are generated by adding the following constraints to the existing DAG.

$$\begin{gathered}
\eta g(x) - A_B^T y_B - A_L^T y_L + A_U^T y_U +\\
A_E^T y_E - F_B'(x)^T z_B - F_L'(x)^T z_L + F_U'(x)^T z_U + F_E'(x) z_E = 0\\
z_L * (F_L(x) - F_L) = 0,\quad z_L \geq 0,\\
z_U * (F_U(x) - F_U) = 0,\quad z_U \geq 0,\\
z_B * (F_B(x) - \underline{F_B}) * (F_B(x) - \overline{F_B}) = 0,\\
z_B * (F_B(x) - \underline{F_B}) \leq 0,\quad z_B * (F_B(x) - \overline{F_B}) \leq 0,\\
y_L * (A_L x - b_L) = 0,\quad y_L \geq 0,\quad y_U * (A_U x - b_U) = 0,\quad y_U \geq 0,\\
y_B * (A_B x - \underline{b_B}) * (A_B x - \overline{b_B}) = 0,\\
y_B * (A_B x - \underline{b_B}) \leq 0,\\
y_B * (A_B x - \overline{b_B}) \leq 0\\
\eta^2 + z_B^T z_B + z_L^T z_L + z_U^T z_U + z_E^T z_E = 1,
\end{gathered} \tag{5}$$

where $*$ denotes componentwise multiplication of two vectors.

4. Dependencies

None

5. Overall limit size

Unknown

6. Services

6.1. 1 Name: `generate`

Input:
(1) *Type*: Any work node *Meaning*: The input problem.
Output: The code returns with a `dag_delta` containing all additional variables (the multipliers), operations and constraints needed to represent the Kuhn-Tucker conditions.
Description: This service generates the Karush–John conditions for the model contained in the work node.
Control Parameters: None
Reason for termination:
Integer: 0 or –1
String:
- “SUCCESS”
- “NO MODEL DEFINED”

Description: None

7. Statistics

Description: None

CHAPTER 18

UWien Linear relaxation generator using slopes

1. Name

`Linear Relaxation SLP`

2. Module Type

Constraint generator

3. Description

`Linear Relaxation SLP` computes a directed acyclic graph representing a linear relaxation of the model represented by the work node passed.

For every constraint $f(x) \in \mathbf{b}$ we compute an interval slope enclosure of the form

$$f(x) \in \mathbf{f} + \sum \mathbf{s}_i(\mathbf{x}_i - z_i).$$

Then we compute a linear underestimator by

$$\underline{l}(x) = \underline{f} + \sum \overline{s}_i(\underline{x}_i - z_i) + \frac{\underline{s}_i(\overline{x}_i - z_i) - \overline{s}_i(\underline{x}_i - z_i)}{\overline{x}_i - \underline{x}_i}(x_i - \underline{x}_i).$$

Analogously, we compute a linear overestimator $\overline{l}(x)$.

Then we add the two new constraints

$$\overline{l}(x) \geq \underline{f}$$
$$\underline{l}(x) \leq \overline{f}.$$

All constraints which would involve infinities are left out.

4. Dependencies

None

5. Overall limit size

Unknown

6. Services

6.1. 1 Name: `generate`

Input:

(1) *Type*: `work_node`.
Meaning: the specification of the problem

Output: The code returns with a `dag_delta` containing all constraints needed to represent the linear relaxation.

Description: This service generates a linear relaxation of the original problem using interval slopes.

Control Parameters:

(1) *Name*: `with_kj`
Type: `bool`
Description: If this is `false` use only the original problem to generate a relaxation. If this parameter is set to `true`, also the Karush-John conditions are used to construct the relaxation.
Default value: `true`

(2) *Name*: `full_delta`
Type: `bool`
Description: This decides whether the `dag_delta` adds constraints to the original problem (`false`) or if the generated constraints replace the original constraints in the model (`true`).
Default value: `true`

(3) *Name*: `center`
Type: `vector<double>`
Description: Specifies the center at which the slopes are being computed. The default value for this parameter is the safeguarded midpoint of the enclosures of all the variables.
Default value: safeguarded midpoint of the box.

Reason for termination:

Integer: 0 or –1
String:
* “SUCCESS”
* “NO MODEL DEFINED”

Description: None

7. Statistics

Description: None

CHAPTER 20

UWien Simple Convexity

H. SCHICHL
University of Vienna, Department of Mathematics
A-1090 Wien, Austria
email: Hermann.Schichl@esi.ac.at

1. Copyright

2. Module name

Module name is `Simple Convexity`

3. Type of module

`Simple Convexity` is an annotation generator.

4. Description of the module

`Simple Convexity` is a module which performs a simple convexity test to all constraints and the objective function.

5. Limits

There are no hardcoded limits.

6. Service *convexity test*

`Simple Convexity` tries to determine the convexity information for all nodes of the DAG. It uses a simple propagation algorithm, as specified in deliverable D12.

6.1. Kind of input. Any work node.

6.2. Output provided. The code returns with a `semantics_delta` changing the convexity information of all nodes and setting the convexity algorithm type to 1.

There is no useful statistical information collected, the efficiency is always 1.

6.3. Control parameters. None.

6.4. Termination reason. There is no termination reason except `'SUCCESS'` (0), and `'NO MODEL DEFINED'` (-1).

7. Statistical information collected

No useful statistical information is collected.

CHAPTER 24

Template for description of modules

1. Name

2. Module Type

e.g., solver, checker, filter

3. Description

of the module

4. Dependencies

on other modules, software, libraries, etc.

5. Overall limit size

of what can be processed by the module

6. Services

for each service

6.1. 1 Name: of the service

Input: for each input parameter of the service
(1) *Type*: the type of the parameter in terms of C++
Meaning: short comment
(2) idem
Output: produced by the service
Description: of what the service does
Control Parameters:
(1) *Name*
Type: the C++ type of the control parameter
Description: short comment
Default value
(2) idem
Reason for termination:

Integer: 0 (success, if non-negative; module-specific termination code otherwise)
String: the messages accompaniying different termination codes
Description: explanations, if necessary

6.2. 2 Name: idem

7. Input

Description: of what is required from the input constraints and objective, e.g. the degree of smoothness

8. Output

Description: of the data produced by all the services
Description of non-standard delta's: if such are generated by some services

9. Statistics

Description: of the statictical data generated by the module

Bibliography

[1] Anthony Brooke, David Kendrick, and Alexander Meeraus. *GAMS - A User's Guide (Release 2.25)*. Boyd & Fraser Publishing Company, Danvers, Massachusetts, 1992. [11]

[2] The COCONUT project home page. http://www.mat.univie.ac.at/coconut. [7]

[3] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL — A Mathematical Programming Language*. Thomson, second edition, 2003. [11]

[4] Josef Kallrath, editor. *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers, Boston Dordrecht London, 2003. [11]

[5] M. Lerch, G. Tischler, and J. Wolff von Gudenberg. *filib++–Interval Library, Specification and Reference Manual*. Informatik, Universität Würzburg, techn. report 279 edition, August 2001. [7]

[6] R.E. Moore and S.T. Jones. Safe Starting Regions for Iterative Methods In *SIAM Journal on Numerical Analysis*, 14: 1051-1065, 1977 [104]

[7] Hermann Schichl and Arnold Neumaier. Exclusion regions for systems of equations. *SIAM J. Num. Analysis*, 2003. to appear. [9, 11]

[8] Hermann Schichl and Arnold Neumaier. Interval analysis on directed acyclic graphs for global optimization. preprint (2003). [8, 42, 44]

[9] J. Siek, A. Lumsdaine, and L.-Q. Lee. Generic programming for high performance numerical linear algebra. In *Proceedings of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98)*. SIAM Press, 1999. [7]

[10] M.-C. Silaghi, D. Sam-Haroud, and B. Faltings. Search Techniques for Non-linear CSPs with Inequalities. In *Proceedings of the 14th Canadian Conference on AI*, 2001. [54, 55]

[11] X.-H. Vu, D. Sam-Haroud, and M.-C. Silaghi. Numerical Constraint Satisfaction Problems with Non-isolated Solutions. In *Post-Proceedings of the 1st International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS'2002)*, France, October 2002. To appear. [54, 55]