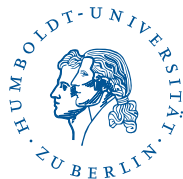


# Interfacing COIN-OR solvers by GAMS

Michael Bussieck

Stefan Vigerske



Operations Research 2007  
Saarbrücken, September 5, 2007

# Overview

## The COIN-OR / GAMSLinks project

Introduction

Download and Installation

Write your own GAMS interface

Quality assurance and Benchmarking

Testing a GAMS solver

Benchmarking with GAMS

A few benchmarks

LP and MIP Solver

NLP Solver

MINLP Solver

Future

## Background

General Algebraic Modeling System = modeling system + solver library

Goals of the GAMSLinks project:

- easy access to COIN-OR solvers via GAMS
  - ⇒ broadening the audience of COIN-OR
  - ⇒ broadening the audience of GAMS
- help developers to connect their solvers to GAMS
- provide access to GAMS benchmarking and quality assurance tools

<https://projects.coin-or.org/GAMSLinks>

# Timeline

2004 Michael Bussieck: links to [GLPK](#) and [COIN-OR/CBC](#) (code is public)

2004 GAMS 21.4: GAMS ships GLPK and CBC (incl. CLP) within the GAMS distribution (free of charge)

2006 Steven Dirkse: link to [COIN-OR/Ipopt](#) (not public)

Jan. '07 start of COIN-OR/GAMSlinks project, Ipopt link becomes public

Feb. '07 GAMS 22.4: GAMS ships GLPK, CBC, and Ipopt

Apr. '07 link to [COIN-OR/Bonmin](#)

Jun. '07 GAMS 22.5: GAMS ships GLPK, CBC, Ipopt, and Bonmin

Aug. '07 link to general [OSI](#) solver (rudimentary links to Symphony, DyLP, Volume)

		Linux (Intel 32+64bit)	Solaris (Intel 64bit)	Windows (32bit)	MacOS (Darwin)
now	OSI	x	x	x	x
	CBC	x	x	x	x
	GLPK	x	x	x	x
	IPOPT	x	x	x	
	BONMIN	x	x	x	

## Download and Installation

1. Get [GAMSLinks code](#) with all referenced COIN-OR projects from <http://www.coin-or.org/download/source/GAMSLinks> or checkout via Subversion from <https://projects.coin-or.org/svn/GAMSLinks/trunk>
2. Get Symphony, DyLP, or Volume code if you want to use them
3. Get [3rd-party codes](#) using `get.XXX` scripts:  
GAMSIO, Blas, Lapack, Mumps, GLPK
4. Apply a small [patch](#):  

```
patch -p0 < osiglpk.patch
```
5. Let the [Autotools](#) and [Compilers](#) do their work:  

```
configure -C --with-gamssystem=...
make
make install
```
6. Copy `gmsXX_.zip`-files from `bin` subdirectory into GAMS distribution and call [gamsinst](#).
7. Use them: `MIP=GLPK`, `MIP=CBC`, `NLP=IPOPT`, `MINLP=BONMIN`, ...

## GAMS I/O libraries

- download scripts (get.XXX) for [precompiled GAMS I/O libraries](#) at [GAMSLinks/ThirdParty/GAMSI0](#)
- supported [architectures](#):
  - Linux on 32- and 64-bit x86-CPU's
  - Solaris on 64-bit x86-CPU's
  - Windows 32-bit x86 with MS C++ Compiler
  - MacOS Darwin
- two ways of accessing GAMS models:

IOLib	SMAGLib
used for many years	a few years old
all architectures supported	only common architectures
all GAMS-features available	some GAMS-features not yet available (SOS, branching prior., =X=)
sometimes difficult to use	quite easy to use (e.g., automatic reform. of objective func.)



LPs and MIPs



NLPs and MINLPs

# The GamsModel class

## IOLib interface:

- hide IOLib functions, allow easy access to a **linear** GAMS-model
- provides a **COIN-OR/OSI - compatible** problem representation
- access to **specialities**: SOS type 1 and 2, semicontinuous variables, branching priorities, user-defined scaling parameters
- access to **GAMS options** and **option file reader**
- writing of **GAMS solution file** given primal/dual values, basis status, status codes, ...
- finalize routine to **process solution from OSI-solver**
- GamsMessageHandler to redirect output into **GAMS output channels** (logfile, statusfile)
- **documentation** generated by doxygen available

# The SMAG library

## SMAG library:

- interface to GAMS model via **C functions**
- get **sparsity** of gradients, Jacobian, and Hessian
- evaluation of **functions, gradients, Jacobian, Hessian** of Lagrangian
- **reformulation of objective function** constraint  $z = \text{objfunc}$
- access to **GAMS options** (reslim, iterlim, ...)
- writing of **GAMS solution file**
- writing to **GAMS status- and logfile**

## In COIN-OR/GAMSLinks for Ipopt and Bonmin interfaces:

- **SMAG\_TNLP** and **SMAG\_TMINLP** classes to represent (MI)NLP
- **SmagJournal** to redirect Ipopt output into status- and logfile
- **SmagMessageHandler** to redirect Bonmin output



## Testing a solver with the GAMS Quality Assurance tools

- the GAMS distribution brings the GAMS [Testlib Library](#)
- developed for [quality control and testing](#)
- models designed to check [solver correctness](#), special functions, ...
- E.g. lp03.gms: examine behaviour of LP solver on model with many free variables and when it restarts from an optimal basis
- quality.gms: driver for [quality tests of all sorts](#)

```
> gams quality.gms --solver=Ipopt --suite=QCP
...
Congratulations! All 4 tests passed.
```

```
> gams quality.gms --solver=Ipopt --suite=LP
...
There were errors: 6 out of 11 tests failed.
See the file failures.gms to reproduce the failed runs
```

## Testing a solver with the GAMS Quality Assurance tools

- the GAMS distribution brings the GAMS [Testlib Library](#)
- developed for [quality control and testing](#)
- models designed to check [solver correctness](#), special functions, ...
- E.g. `lp03.gms`: examine behaviour of LP solver on model with many free variables and when it restarts from an optimal basis
- `quality.gms`: driver for [quality tests of all sorts](#)
- Number of [failed tests](#) of GAMS / COIN-OR solvers on testsuites:

	LP	MIP	QCP	NLP
CBC/CLP	0	4*	–	–
GLPK	0	4*	–	–
IPOPT	6	–	0	0
BONMIN	6	4*	0	0

(date: 01.09.2007)

\* – tests on handling of SOS and semicontinuous variables

# The GAMS World ([www.gamsworld.org](http://www.gamsworld.org))

A collection of many worlds:

- CONE World: Conic Optimization
- GLOBAL World: Global Optimization of NLPs
- MINLP World: Mixed Integer Nonlinear Programming
- MPEC World: Mathematical Programs with Equilibrium Constraints
- MPSGE World: Mathematical Programming System for General Equilibrium
- Translation: translate GAMS models into other languages
- **Performance World**: performance testing of solvers
  - PerformanceLib: Libraries of test problems, e.g., [LINLib](#) (LP, MIP, QCP)
  - **Performance Tools**: simplifying performance data collection, measurement, postprocessing, and visualization
  - **PAVER**: Server for Automated Performance Analysis & Visualization

## Using Performance Tools

1. Create a **list of models**  $\Rightarrow$  LPs.list

2. Create **batch files** using crbatch.gms  $\Rightarrow$  LPs\_CPLEX.gms:

```
gams crbatch.gms --batfile=LPs_batch --modelfile=LPs.list --type=LP
  --solver=cplex --iterlim=999999 --reslim=3600
  --gmsopts="trace=cplex.trc traceopt=3"
```

...

3. Start benchmarks to create **trace files**  $\Rightarrow$  cplex.trc, cbc.trc, ...

```
gams LPs_CPLEX.gms
gams LPs_CBC.gms
gams LPs_GLPK.gms
```

4. **Compare** solvers, generate performance profiles using pprocess.gms:

```
gams pprocess --trace1=cplex.trc --trace2=cbc.trc --trace3=glpk.trc
```

(or let the PAVER server do this job)

$\Rightarrow$  performance profiles, comparison of solver outcomes, comparison of solver resource times

## Performance Profiles

E.D. Dolan and J.J. More, Mathematical Programming, 91, 2002:

- compare performance of solver  $s \in \mathcal{S}$  on problem  $p \in \mathcal{P}$  with best performance by any solver on problem  $p$ :

$$\rho(p, s) := \frac{t_{p,s}}{\min_{s' \in \mathcal{S}} t_{p,s'}}$$

- $t_{p,s}$  = time solver  $s$  spend on  $p$ ,  $t_{p,s} = \infty$  if  $s$  did not solve  $p$
- $P_s(\tau)$  = probability that performance ratio  $\rho(p, s)$  within factor of  $\tau$  of best possible ratio:

$$P_s(\tau) := \frac{|\{p \in \mathcal{P} : \rho(p, s) \leq \tau\}|}{|\mathcal{P}|}$$

- percentage of models that solver  $s$  will solve if for each model,  $s$  can have a maximum resource time of  $\tau$  times the minimum time
- $s$  solved  $p$ : found feasible point or found best possible solution

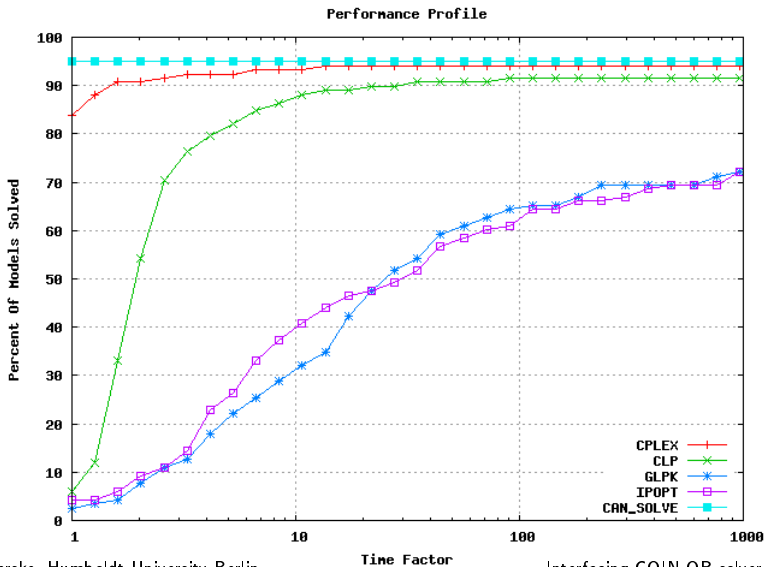
## LPs from LINLib

- LINLib: [performance library](#) from GAMSWorld with both theoretical and practical test models (LP, MIP, QCP, includes netlib and miplib)
- for our test: LPs with at least 50.000 nonzeros  $\Rightarrow$  [118 models](#)
- Competitors: [CPLEX](#) 10.20, [CLP](#) (Aug. '07), [GLPK](#) 4.20, [IPOPT](#) 3.3
- timelimit: [1 hour](#)

Results:

	CPLEX	CLP	GLPK	IPOPT
solved to optimality	112	109	86	93
infeasible reported	4	4	4	
timelimit exceeded	2	5	26	14
failed			2	11

## LPs from LINLib - Performance profile



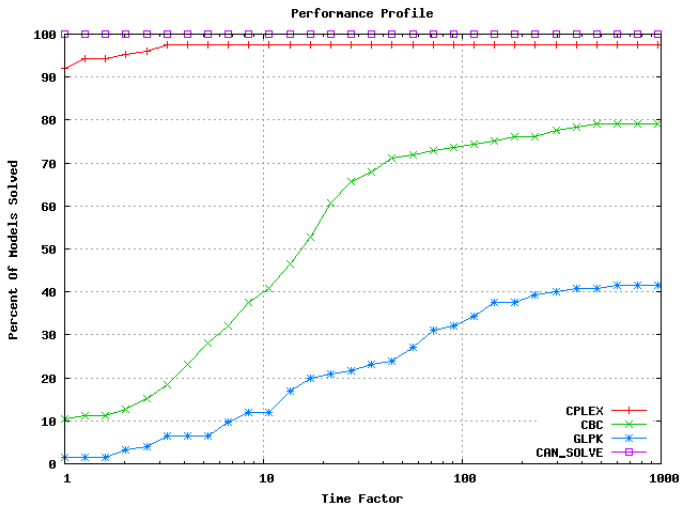
## MIPs from LINLib

- all MIPs from LINLib  $\Rightarrow$  125 models
- Competitors: CPLEX 10.20, CBC (August 2007), GLPK 4.20
- timelimit: 1 hour
- gap tolerance: 0.01%

	CPLEX	CBC	GLPK
integer feasible solution found	124	121	60
failed	1	4	65



# MIPs from LINLib - Performance profile



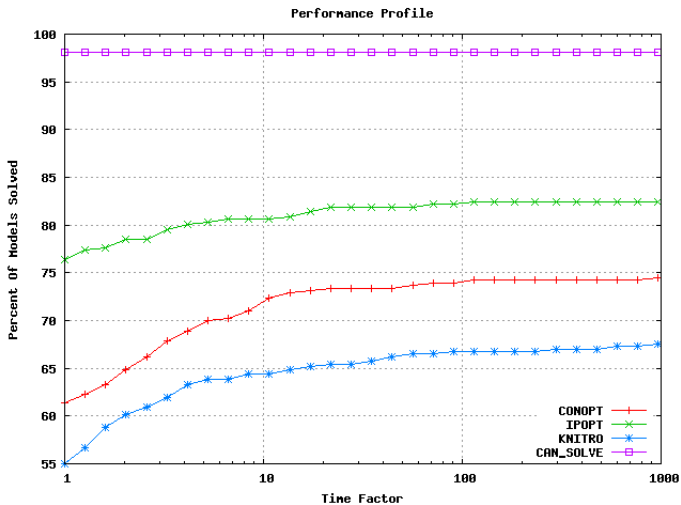
solved = found best solution among all solvers

## NLPs from GlobalLib

- all NLPs from GlobalLib  $\Rightarrow$  379 models
- Competitors: CONOPT 3.14r, KNITRO 5.1, IPOPT 3.3
- timelimit: 1 hour

	CONOPT	KNITRO	IPOPT
(local) optimal	340	341	366
feasible	3	3	5
infeasible	21	1	6
unbounded	5	0	2
timelimit or other failure	5	31	5

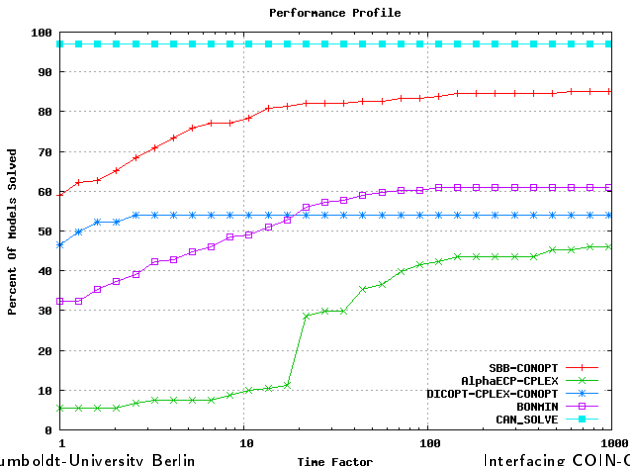
# NLPs from GlobalLib - Performance profile



solved = found best solution among all solvers

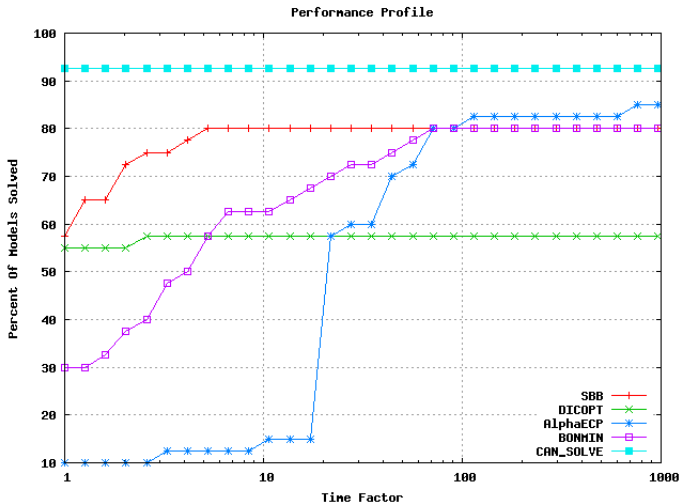
# MINLPs from MINLPLib

- selected MINLPs from MINLPLib  $\Rightarrow$  167 models
- Competitors: SBB, DICOPT, AlphaECP 1.30, BONMIN (Aug. '07)
- timelimit: 1 hour, gap tolerance: 1%



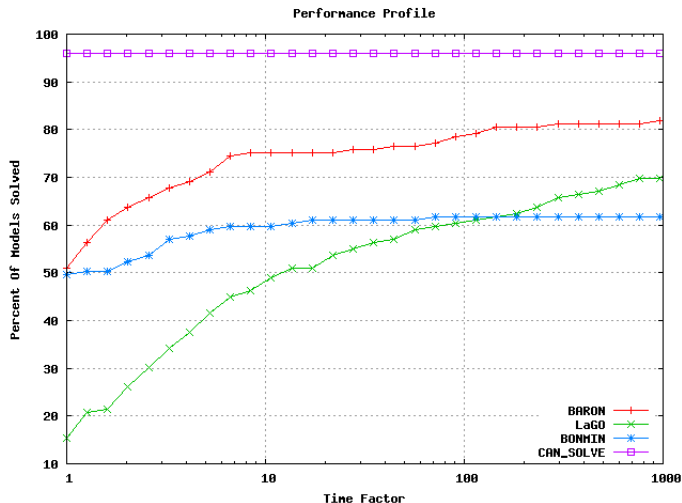
## convex MINLPs from MINLPlib

- only convex MINLPs  $\Rightarrow$  40 models



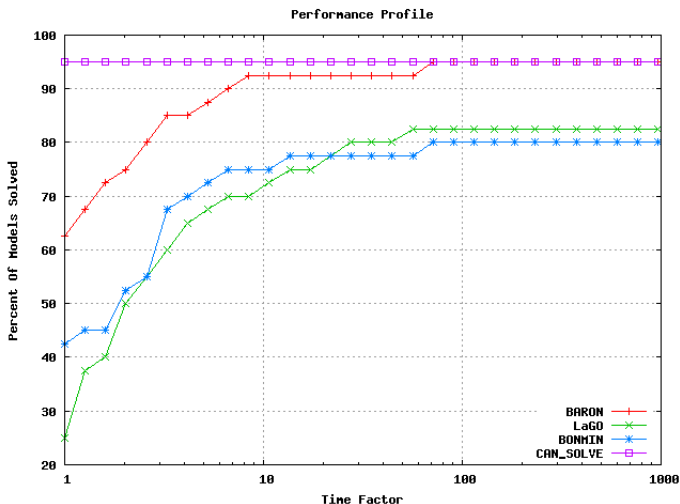
## MINLPs from MINLPlib - other solvers

- Competitors: **BARON**, **LaGO** (Aug. '07), **BONMIN** (Aug. '07)



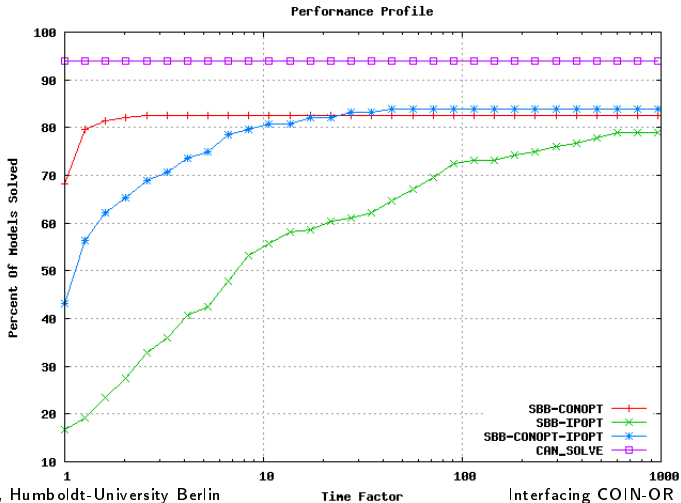
## convex MINLPs from MINLPlib - other solvers

- only convex MINLPs  $\Rightarrow$  40 models



## SBB on MINLPs with different NLP subsolver

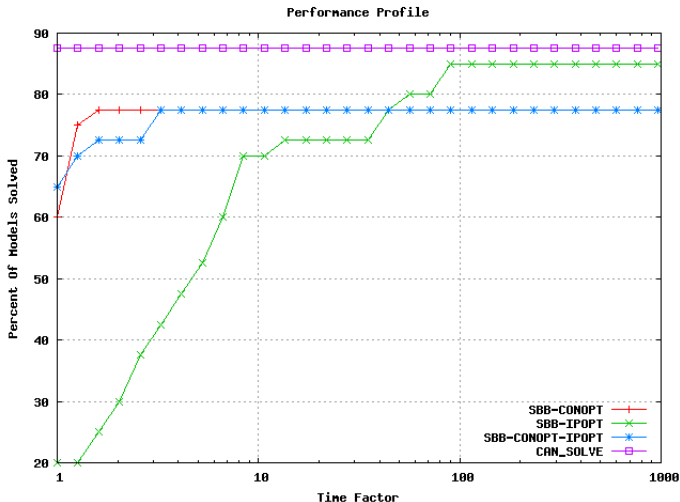
- benchmark **CONOPT** vs. **IPOPT** as SBB subsolver
- using SBB option `infeasseq`: let SBB call **IPOPT** if **CONOPT** reports infeasible and SBB is at most at depth 50 of the search tree





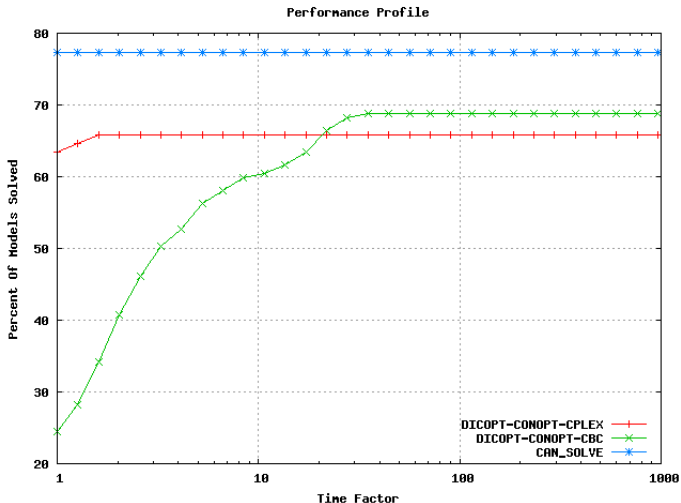
# SBB on convex MINLPs with different NLP subsolver

- only convex MINLPs  $\Rightarrow$  40 models



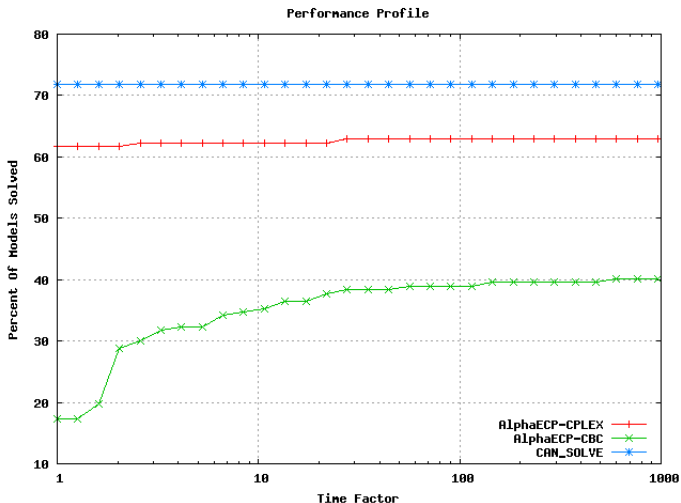
# DICOPT on MINLPs with different MIP subsolver

- benchmark CPLEX vs. CBC as DICOPT subsolver



# AlphaECP on MINLPs with different MIP subsolver

- benchmark **CPLEX** vs. **CBC** as AlphaECP subsolver



## Further Developments

**Improve** existing links:

- GLPK: access to interior-point solver
- CLP: support quadratic objective function
- GAMS Branch-and-Cut-and-Heuristic Facility for CBC and Bonmin
- Bonmin: GAMS NLP-solver as alternative subsolver
- ...

**Create** more interfaces:

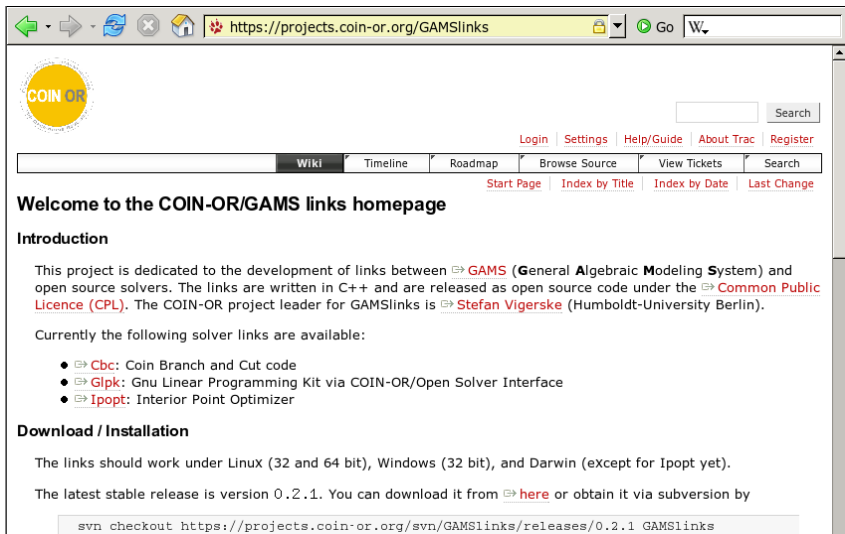
- GAMS to OSiL (requires access to GAMS expression trees)
- Symphony, LaGO, ...

**Help** people to hook up their solver to GAMS:

- documentation

# Thank you!

PS: **Your** contribution is welcome at



The screenshot shows a web browser window with the address bar containing `https://projects.coin-or.org/GAMSLinks`. The page features the COIN-OR logo, a search bar, and navigation links such as [Login](#), [Settings](#), [Help/Guide](#), [About Trac](#), and [Register](#). Below the navigation is a menu with options like [Wiki](#), [Timeline](#), [Roadmap](#), [Browse Source](#), [View Tickets](#), and [Search](#). The main content area is titled "Welcome to the COIN-OR/GAMS links homepage" and includes an "Introduction" section. The introduction states that the project is dedicated to developing links between GAMS (General Algebraic Modeling System) and open source solvers, written in C++ and released under the Common Public Licence (CPL). It also mentions that the COIN-OR project leader for GAMSLinks is Stefan Vigerske (Humboldt-University Berlin). A list of currently available solver links is provided, including Cbc, Glpk, and Ipopt. A "Download / Installation" section follows, stating that the links should work under Linux (32 and 64 bit), Windows (32 bit), and Darwin (except for Ipopt yet). The latest stable release is version 0.2.1, which can be downloaded from [here](#) or obtained via subversion. A code block at the bottom shows the subversion checkout command: `svn checkout https://projects.coin-or.org/svn/GAMSLinks/releases/0.2.1 GAMSLinks`.

[Start Page](#) | [Index by Title](#) | [Index by Date](#) | [Last Change](#)

## Welcome to the COIN-OR/GAMS links homepage

### Introduction

This project is dedicated to the development of links between [GAMS](#) (**General Algebraic Modeling System**) and open source solvers. The links are written in C++ and are released as open source code under the [Common Public Licence \(CPL\)](#). The COIN-OR project leader for GAMSLinks is [Stefan Vigerske](#) (Humboldt-University Berlin).

Currently the following solver links are available:

- [Cbc](#): Coin Branch and Cut code
- [Glpk](#): Gnu Linear Programming Kit via COIN-OR/Open Solver Interface
- [Ipopt](#): Interior Point Optimizer

### Download / Installation

The links should work under Linux (32 and 64 bit), Windows (32 bit), and Darwin (except for Ipopt yet).

The latest stable release is version 0.2.1. You can download it from [here](#) or obtain it via subversion by

```
svn checkout https://projects.coin-or.org/svn/GAMSLinks/releases/0.2.1 GAMSLinks
```

## GamsModel usage (1/2)

```
OsiClpSolverInterface solver;
GamsModel gm(<controlfile-name>);
gm.setInfinity(-solver.getInfinity(), solver.getInfinity());

gm.ReadOptionsDefinitions("cbc");
gm.ReadOptionsFile();
solver.setHintParam(OsiDoScale, gm.optGetBool("scaling"));

gm.readMatrix();

solver.setObjSense(gm.ObjSense());
solver.setDblParam(OsiObjOffset, gm.ObjConstant());
double* rowrng=CoinCopyOfArray(NULL, gm.nRows(), 0.);
solver.loadProblem(gm.nCols(), gm.nRows(), gm.matStart(),
  gm.matRowIdx(), gm.matValue(), gm.ColLb(), gm.ColUb(),
  gm.ObjCoef(), gm.RowSense(), gm.RowRhs(), rowrng);
for (j=0; j<gm.nCols(); ++j)
  if (gm.Coldisc()[j]) solver.setInteger(j);
```

## GamsModel usage (2/2)

```
CbcModel model(solver);
GamsMessageHandler slvout(&gm);
model.passInMessageHandler(&slvout);

gm.TimerStart();
model.initialSolve();

if ((0==gm.nDCols()) || !model.solver()->isProvenOptimal()) {
    gm.setIterUsed(model.solver()->getIterationCount());
    gm.setResUsed(gm.SecondsSinceStart());
    gm.setObjVal(gm.ObjSense()*model.solver()->getObjValue());
    GamsFinalizeOsi(&gm, &slvout, model.solver(), 0);
    return EXIT_SUCCESS;
}
CbcStrategyGams strategy(gm);
model.setStrategy(strategy);
model.branchAndBound();

GamsFinalizeOsi(&gm, &slvout, model.solver(), presolve_infeas);
```

## SMAG library usage (1/2)

```
smagHandle_t prob = smagInit(<controlfile-name>);
smagStdOutputStart(prob, SMAG_STATUS_OVERWRITE_IFDUMMY, buffer, buf_size);

smagReadModelStats(prob);
smagSetObjFlavor(prob, OBJ_FUNCTION);
smagSetSqueezeFreeRows(prob, 1); /* don't show me =n= rows */
smagReadModel(prob);
smagHessInit(prob);

SmartPtr<TNLP> smagnlp = new SMAG_NLP(prob);
SmartPtr<IpoptApplication> app = new IpoptApplication(false);

SmartPtr<Journal> jrnl=new SmagJournal(prob, "console", J_ITERSUMMARY);
jrnl->SetPrintLevel(J_DBG, J_NONE);
app->Jnlst()->AddJournal(jrnl);

if (prob->gms.useopt) app->Initialize(prob->gms.optFileName);
else app->Initialize("");
app->OptimizeTNLP(smagnlp);
```



## SMAG library usage (2/2)

- evaluation of **objective and constraints**:

```
smagEvalObjFunc(prob, x, &obj_value);  
smagEvalConFunc(prob, x, gradient);
```

- evaluation of **Jacobian**:

```
smagEvalConGrad(prob, x);    k=0;  
for (Index row = 0; row < m; ++row)  
    for (smagConGradRec_t* cGrad = prob->conGrad[row]; cGrad;  
         cGrad = cGrad->next, ++k) {  
        values[k] = cGrad->dcdj;  
        jCol[k]   = cGrad->j;  
        iRow[k]   = row;  
    }  
}
```

- writing a **solution file**:

```
smagReportSolBrief(prob, modelStatus, solverStatus);  
smagReportSolStats(prob, modelStatus, solverStatus, iterations, time,  
                   objValue, domviolations);  
smagReportSolFull (prob, modelStatus, solverStatus, iterations, time,  
                  objValue, domviolations, rowValues, rowDuals, rowBasisStatus,  
                  rowIndicator, colValues, colDuals, colBasisStatus, colIndicator);
```