Hinweise zur Installation und Benutzung der MIP-Solver

GLPK wikipedia: *Das GNU Linear Programming Kit (GLPK) ist eine im GNU-Projekt entwickelte und in C geschriebene dynamische Programmbibliothek zur Lösung von Problemen der linearen Optimierung und der ganzzahligen linearen Optimierung. GLPK enthält Implementierungen des revidierten Simplex-Verfahrens, des Innere-Punkte-Verfahrens und des Branch-and-Bound-Verfahrens. Außerdem bietet es Funktionen, um ein in GNU MathProg beschriebenes Problem der linearen oder ganzzahligen linearen Optimierung zu lösen, und das eigenständige Programm glpsol zur Lösung linearer und ganzzahlig linearer Probleme.*

Zum Compilieren der C++-Programme aus der Vorlesung muss zum Linken die GLPK-Bibliothek angegeben werden:

```
g++ -Wall -Wextra glp.cpp -lglpk -O2
g++ glpSolver.cpp -lglpk -O2  // Node-Selection = DFS
g++ -DBFS glpSolver.cpp -lglpk -O2  // Node-Selection = BFS
```

Die GLPK-Bibliothek ist in vielen Linux-Distributionen enthalten und kann bspw. unter Ubuntu mittels `sudo apt install libglpk-dev` installiert werden. Nach dem Compilieren der obigen Programme können bspw. die Benchmarks aus der MIPLIB[1][2][3] ausgeführt werden:

```
./a.out miplib2003/misc07.mps.gz
./a.out miplib2003/10teams.mps.gz
```

glpsol Der GLPK-Solver ist in vielen Linux-Distributionen als Paket enthalten und kann bspw. unter Ubuntu mittels `sudo apt install glpk-utils` installiert werden. Dessen Solver `glpsol`[4] kann dann direkt mit den Problemen aus der MIPLIB gestartet werden:

```
glpsol miplib2003/10teams.mps.gz --tmlim 300
glpsol miplib2003/10teams.mps.gz --cuts --pcost
```

Mittels des Parameters `--cuts` werden die Verfahren zur Berechnung von Schnittebenen aktiviert, mittels `--pcost` wird als Branching-Rule die Pseudo-Cost-Heuristik festgelegt, und schließlich begrenzt `--tmlim 300` die Ausführungszeit auf 300 Sekunden, also 5 Minuten.

Um lineare Programme (man spricht auch von Modellen) zu lösen, die mittels GMPL formuliert sind, geht man wie folgt vor. Die Option `-m` legt fest, welches Programm ausgeführt bzw. welches Modell gelöst werden soll, die Option `-d` spezifiziert die Parameter-Datei.

- Brett-Solitär:

```
glpsol -m pegSol.mod -d pegSol.dat --cuts --pcost
glpsol -m pegSol.mod -d pegSol.dat --minisat
```

[1] MIPLIB 2017: `https://miplib.zib.de/`
[2] MIPLIB 2010: `https://miplib2010.zib.de/`
[3] MIPLIB 2003: `https://miplib2010.zib.de/miplib2003/index.php`
[4] Mittels `glpsol -h` wird die Hilfe zu dem Solver angezeigt.

Die Option `--minisat` weist den GLPK-Solver an, das binäre Integer-Programme in einen booleschen Ausdruck in konjunktiver Normalform (CNF-SAT) umzuwandeln und diesen dann mittels des SAT-Solvers `minisat`[5] zu lösen. Das ist bei manchen Problemen effektiver.

- Sudoku:

```
glpsol -m sudoku.mod -d sudoku09.dat
glpsol -m sudoku.mod -d sudoku16.dat
glpsol -m sudoku.mod -d sudoku25.dat --cuts
```

Problem bei GLPK: Es stehen nur sehr wenige Verfahren zur Berechnung von Schnittebenen zur Verfügung, außerdem ist GLPK nur single-threaded. Daher ist GLPK nicht zum Lösen schwerer Probleme geeignet. Leistungsfähiger ist der CBC-Solver von Coin-OR (COmputational INfrastructure for Operations Research).

Coin-OR Branch-and-Cut-Solver wikipedia: *Computational Infrastructure for Operations Research (COIN-OR), is a project that aims to "create for mathematical software what the open literature is for mathematical theory." The open literature (e.g., a research journal) provides the operations research (OR) community with a peer-review process and an archive. Papers in operations research journals on mathematical theory often contain supporting numerical results from computational studies. [...]*

The success of Linux, Apache, and other projects popularized the open-source model of software development and distribution. A group at IBM Research proposed open source as an analogous yet viable means to publish software, models, and data. COIN-OR was conceived as an initiative to promote open source in the computational operations research community and to provide the on-line resources and hosting services required to enable others to run their own open-source software projects.

The COIN-OR website was launched as an experiment in 2000, in conjunction with 17th International Symposium on Math Programming in Atlanta, Georgia. In 2007, COIN-OR had 25 application projects, including tools for linear programming (e.g., COIN-OR CLP), nonlinear programming (e.g., IPOPT), integer programming (e.g., CBC, Bcp and COIN-OR SYMPHONY), algebraic modeling languages (e.g., Coopr) and more. By 2011, this had grown to 48 projects. COIN-OR is hosted by the Institute for Operations Research and the Management Sciences, INFORMS, and run by the educational, non-profit COIN-OR Foundation.

Der CBC-Solver ist in vielen Linux-Distributionen enthalten und kann bspw. unter Ubuntu mittels `sudo apt install coinor-cbc` installiert werden. Möchte man die Funktionalität mittels der API in C++-Programmen nutzen, muss ebenfalls das Paket `coinor-libcbc-dev` installiert werden.

Leider haben solche bereits kompilierten Pakete des Coin-OR CBC-Solvers oft keine GMPL-Unterstützung. In solchen Fällen muss dann der Source-Code kompiliert werden.

- Der Source-Code des Coin-OR CBC-Solvers kann bspw. von der Web-Seite `https://www.coin-or.org/download/source/Cbc/` bezogen werden. In diesem Code sind unter anderem auch Third-Party-Implementierungen von GLPK, Blas (Basic Linear Algebra Subprograms), Lapack (Linear Algebra PACKage) und ASL (Atomic Secured Linux) enthalten.

[5] `http://minisat.se/`

Um die Third-Party-Implementierungen nutzen zu können, muss zunächst in das entsprechende Verzeichnis, bspw. `ThirdParty/Glpk` gewechselt und dann das entsprechende Get-Script, hier `get.Glpk` ausgeführt werden.

- Bevor `./configure -C --enable-cbc-parallel` ausgeführt wird, bitte darauf achten, dass ein Fortran-Compiler wie `gfortran` installiert ist. Außerdem sollten die Pakete `dos2unix` und `doxygen` installiert sein. Die Option `--enable-cbc-parallel` ist wichtig, damit der Solver später multi-threaded ist.
- Anschließend kann das eigentliche Kompilieren erfolgen:

```
make
make test
make install
```

Dann sollte CBC mit GMPL-Unterstützung zur Verfügung stehen. Wir testen das bspw. mittels folgendem Aufruf, der das Modell für Brett-Solitär löst:

```
cbc pegSol.mod%pegSol.dat -threads 4 -solve -gsolu cbc.sol
```

Der erste Parameter `pegSol.mod%pegSol.dat` legt die Modell- und die Parameterdatei fest, wobei die Dateinamen durch ein Prozentzeichen zu trennen sind. Der Parameter `-threads 4` gibt an, dass 4 Threads genutzt werden sollen. Diese Art von Multi-Threading ist allerdings nicht deterministisch, sowohl die Laufzeiten als auch die Ergebnisse können bei verschiedenen Läufen unterschiedlich sein.

Ist ein deterministischer Ablauf gewünscht, wird auf die Anzahl der Threads der Wert 100 addiert. Der Aufruf lautet dann also:

```
cbc pegSol.mod%pegSol.dat -threads 104 -solve -gsolu cbc.sol
```

Wird `cbc` auf der Console gestartet und die Hilfe für den Parameter mittels `threads??` angefordert, erhält man die folgende Auskunft:

```
thread(s) : Number of threads to try and use.
To use multiple threads, set threads to number wanted.  It may be
better to use one or two more than number of cpus available.  If 100+n
then n threads and search is repeatable (maybe be somewhat slower),
if 200+n use threads for root cuts, 400+n threads used in sub-trees.
```

Mittels der Eingabe `quit` kann der Solver verlassen werden.

Coin-OR CBC ohne GMPL-Unterstützung Auch ohne Übersetzen des Source-Codes von CBC können mittels GMPL formulierte Programme ausgeführt bzw. Modelle gelöst werden, allerdings auf einem Umweg und ohne die entsprechende Ausgabe. Mittels

```
glpsol -m pegSol.mod -d pegSol.dat --wlp peg.lp --check
```

wird das lineare Programm bzw. das Modell in das LP-Format umgewandelt. Der Parameter `--wlp filename` steht für „write problem to filename in CPLEX LP format“. Es gibt weitere Umwandlungen bspw. mittels `--wmps` zur Umwandlung ins MPS- oder `--wcnf` zur Umwandlung ins DIMACS CNF-SAT-Format. Der Parameter `--check` bewirkt, dass das vom Benutzer bereitgestellte Modell nur auf seine Beschreibung und Syntax hin überprüft, aber nicht gelöst wird.

Das umgewandelte Modell kann dann mittels CBC und dem folgenden Aufruf

```
cbc peg.lp -threads 4 -solve -printm "M*" -solu sol.txt
```

ausgeführt werden. Dabei gibt `-solu sol.txt` an, dass CBC die Lösung in die Datei `sol.txt` schreiben soll. Mittels des Parameters `printm` kann eine Maske definiert werden, sodass nur Variablen in die Ausgabe geschrieben werden, deren Name zu der angegebenen Maske passt. Da beim Brett-Solitär die Variablen, die die Züge (Moves) bezeichnen, alle mit `M` beginnen, setzen wir die Maske hier auf `M*` und erhalten die folgende (gekürzte) Ausgabe:

```
Optimal - objective value -32.00000000
      105 M(2,4,4,4,1)                1                0
      186 M(3,2,3,4,2)                1                0
      298 M(5,2,3,2,3)                1                0
      381 M(5,4,5,2,4)                1                0
      465 M(5,6,5,4,5)                1                0
      ....
```

Man sieht, dass als erstes der Zug von (2,4) nach (4,4) ausgeführt wird und danach der Zug von (3,2) nach (3,4) folgt. Der letzte Index gibt jeweils an, der wievielte Zug gespielt wird.

IBM ILOG CPLEX Optimization Studio wikipedia: *IBM ILOG CPLEX Optimization Studio (meist nur bezeichnet als CPLEX) ist ein Programmsystem zur Modellierung und Lösung von Optimierungsproblemen mithilfe der mathematischen Optimierung sowie der Constraint-Programmierung. CPLEX stellt neben einem kommandozeilen-basierten Solver auch die Modellierungssprache OPL und umfangreiche Bibliotheken mit Anbindung an die Programmiersprachen C, C++, C#, Java und Python bereit.*

Durch die *Academic Initiative*[6] steht CPLEX auch für Studierende kostenlos zur Verfügung. CPLEX kann ebenfalls Dateien im LP- oder MPS-Format einlesen und die entsprechenden Modelle lösen. Dazu wird CPLEX bspw. in der Konsole gestartet, anschließend sind bspw. die folgenden Befehle auszuführen:

```
CPLEX> read peg.lp
CPLEX> opt
CPLEX> set output writelevel 3
CPLEX> write peg.sol
CPLEX> display solution variables M*
CPLEX> quit
```

Durch den ersten Befehl `read peg.lp` wird das angegebene Modell eingelesen, durch den zweiten Befehl `opt` wird der Optimizer ausgeführt, also das Modell gelöst. Die dritte Anweisung `set` legt den Wert der angegebenen Variablen fest. Die Variable `writelevel` legt den Detaillierungsgrad [7]. fest, der beim Schreiben einer Lösung in eine Datei genutzt wird:

- Wert 1: CPLEX schreibt alle Variablen, sowohl diskret als auch kontinuierlich, mit ihren Werten in die Ausgabedatei.
- Wert 2: CPLEX schreibt nur Werte für diskrete Variablen.
- Wert 3: CPLEX schreibt nur Werte der Variablen nonzero.
- Wert 4: CPLEX schreibt nur Werte für diskrete Variablen.

[6] `https://www.ibm.com/academic/`

[7] `https://www.ibm.com/docs/de/icos/22.1.2?topic=parameters-write-level-mst-sol-files`

Die Ausgabe der Lösung in eine Datei erfolgt durch die `write`-Anweisung.

Anstatt die Lösung in eine Datei zu schreiben, kann die Lösung auch auf dem Bildschirm angezeigt werden. Mittels der `display`-Anweisung werden die Variablen, deren Name mit `M` beginnen, auf dem Bildschirm ausgegeben. Schließlich wird CPLEX mittels `quit` verlassen.

Alternativ kann direkt beim Aufruf des Programms eine Folge von Befehlen definiert werden, die dann von CPLEX in der angegebenen Reihenfolge ausgeführt werden.

```
cplex -c "read peg.lp" "opt" "display solution variables M*"
```

Für Brett-Solitär erhalten wir folgende (gekürzte) Ausgabe:

```
Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.8.0.0
.....
Root node processing (before b&c):
  Real time             =    4.02 sec. (3403.96 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                          ------------
Total (root+branch&cut) =    4.02 sec. (3403.96 ticks)
.....
Variable Name           Solution Value
M(4,2,4,4,1)                 1.000000
M(2,3,4,3,2)                 1.000000
M(3,1,3,3,3)                 1.000000
M(3,4,3,2,4)                 1.000000
M(5,1,3,1,5)                 1.000000
.....
```

Alternativ kann die Modell-Beschreibung mittels OPL erfolgen. Leider ist die Syntax eine andere als bei GMPL. Um OPL zu nutzen, wird der Befehl `oplrun` in einer Konsole genutzt, wie in dem folgenden Beispiel:

```
oplrun pegSolOpl.mod pegSolOpl.dat
```

weitere Solver Es gibt weitere nicht-kommerzielle Solver wie SCIP[8]. Auf der Seite `https://www.scipopt.org/index.php#download` steht bspw. für Ubuntu-Linux ein self-extracting archive zur Verfügung. Zur Installation werden die folgenden Pakete benötigt:

```
gcc g++ gfortran liblapack3 libopenblas0 libtbb12 libcliquer1
libmetis5 libboost-program-options1.83.0 libboost-serialization1.83.0
libgmp10 patchelf
```

Nach der Installation kann SCIP dann interaktiv gestartet werden.

```
SCIP> read peg.lp
SCIP> set parallel
```

[8] `https://www.scipopt.org/`

```
SCIP/set/parallel> maxnthreads 4
SCIP> concurrentopt
SCIP> write solution scip.sol
SCIP> quit
```

Ähnlich wie bei CPLEX kann auch bei SCIP der interaktive Modus durch eine Folge von Befehlen auf der Kommandozeile ersetzt werden. Die obige interaktive Sequenz von Befehlen kann auch wie folgt ausgeführt werden:

```
scip -c "read peg.lp" -c "set parallel" -c "maxnthreads 4" \
     -c "concurrentopt" -c "write solution scip.sol" -c "quit"
```

IBM ist natürlich nicht der einzige Anbieter von MIP-Solvern. Es gibt weitere kommerzielle Produkte wie die Solver von Gurobi[9] oder von FICO[10]. Auch Gurobi[11] und FICO[12] stellen für Universitäten und Hochschulen eine spezielle Lizenz für die Nutzung ihrer Solver zur Verfügung.

Diese Liste der MIP-Solver ist weder vollständig noch repräsentativ. Die sehr interessante Seite `https://plato.asu.edu/bench.html` von Hans Mittelmann vergleicht einige MIP-Solver miteinander.

[9] `https://www.gurobi.com/`
[10] `https://www.fico.com/de/products/fico-xpress-optimization`
[11] `https://www.gurobi.com/academia/academic-program-and-licenses/`
[12] `https://community.fico.com/s/academic-programs`