LP solver support

Some configurations of the search component of Fast Downward, such as optimal cost partitioning for landmark heuristics, require a linear programming (LP) solver and will complain if the planner has not been built with support for such a solver. Running an LP configuration requires three steps, explained below:

1. Installing one or more LP solvers.
2. Building Fast Downward with LP support.
3. Choosing a solver with command-line arguments.

Step 1. Installing one or more LP solvers

Fast Downward uses a generic interface for accessing LP solvers and hence can be used together with different LP solvers. Currently, CPLEX and SoPlex are supported. You can install one, or both solvers without causing conflicts. Installation varies by solver and operating system.

Installing CPLEX on Linux/macOS

IBM offers a free academic license that includes access to CPLEX. Once you are registered, you find the software under Technology -> Data Science. Choose the right version and switch to HTTP download unless you have the IBM download manager installed. If you have problems using their website with Firefox, try Chrome instead. Execute the downloaded binary and follow the guided installation. If you want to install in a global location, you have to execute the installer as `root`.

After the installation, set the following environment variable. The installer is for ILOG Studio, which contains more than just CPLEX, so the variable points to the subdirectory `/cplex` of the installation. Adapt the path if you installed another version or did not install in the default location:

```
export DOWNWARD_CPLEX_ROOT=/opt/ibm/ILOG/CPLEX_Studio2211/cplex
```

If you don't want to permanently modify your environment, you can also set these variables directly when calling CMake. The variable needs to be set when building Fast Downward's search component (Step 2.).

Installing CPLEX on Windows

Follow the Linux instructions to acquire a license and access the Windows-version of the CPLEX installer. Please install CPLEX into a directory without spaces. For a silent installation, please consult: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.studio.help/Optimization_Studio/topics/td_silent_install.html

⚠ **Important Note:** Setting up environment variables might require using / instead of the

more Windows-common \ to work correctly.

Installing !SoPlex on Linux/macOS

SoPlex is available under the Apache License from Github. To be compatible with C++-20, we require a version of SoPlex more recent than 6.0.3. At the time of this writing, 6.0.3 is the latest release, so we have to build from the unreleased version at the tip of the main branch.

You can install SoPlex as follows (adapt the path if you install another version or want to use another location):

```
git clone https://github.com/scipopt/soplex.git
sudo apt install libgmp3-dev # The library is optional but important
for SoPlex's performance
export DOWNWARD_SOPLEX_ROOT=/opt/soplex-6.0.3x
export CXXFLAGS="$CXXFLAGS -Wno-use-after-free" # Ignore compiler
warnings about use-after-free
cmake -S soplex -B build
cmake --build build
cmake --install build --prefix $DOWNWARD_SOPLEX_ROOT
rm -rf soplex build
```

After installation, permanently set the environment variable `DOWNWARD_SOPLEX_ROOT` to the value you used in installation.

⚠ **Note:** Once version 6.0.4 or later is released, we can update this and can recommend the SoPlex homepage for downloads instead.

Installing SoPlex on the grid in Basel

To build SoPlex on the grid, you should load a module with the GMP library and a compatible compiler module. The following setup should work:

```
module purge
module load GCC/11.3.0.lua
module load CMake/3.23.1-GCCcore-11.3.0.lua
module load Python/3.10.4-GCCcore-11.3.0.lua
module load GMP/6.2.1-GCCcore-11.3.0
```

Because the library is loaded from a module, it is not in a default directory, so change the CMake call to

```
cmake -DCMAKE_INSTALL_PREFIX="$DOWNWARD_SOPLEX_ROOT"
-DGMP_DIR="$EBROOTGMP" ..
```

Step 2. Installing the Open Solver Interface

The Open Solver Interface (OSI) provides a common interface to different LP solvers. OSI must be compiled **after** installing the LP solver(s) in Step 1. If you install a solver later, repeat the installation steps of OSI. We assume in the following that CPLEX and SoPlex are installed and the environment variables `DOWNWARD_CPLEX_ROOT` and

DOWNWARD_SOPLEX_ROOT are set up correctly.

These instructions apply to COIN/OSI 0.107.9, newer versions from http://www.coin-or.org/download/source/Osi/ should work, too. If you experience problems, please let us know.

Installing the Open Solver Interface on Linux

The way we install OSI depends on zlib, so make sure to install it first with

```
sudo apt install zlib1g-dev
```

Run the following commands. If you have chosen not to install CPLEX or SoPlex, **omit** the options related to the corresponding solver from the call to `./configure` (options for solvers that are not installed can lead to very cryptic error messages). If you installed other solvers like CLP or Gurobi, also add the corresponding options for them. You may have to include the path to shared libraries in LDFLAGS, i.e. `LDFLAGS="-L$DOWNWARD_CPLEX_ROOT/lib -L/usr/lib/gcc/i686-linux-gnu/4.8"`. If there are other problems, see the troubleshooting section.

Zeilennummern ein/ausschalten Zeilennummern ein/ausschalten

```
wget http://www.coin-or.org/download/source/Osi/Osi-0.107.9.tgz
tar xvzf Osi-0.107.9.tgz
cd Osi-0.107.9

export DOWNWARD_COIN_ROOT=/opt/coin/Osi-0.107.9


./configure CC="gcc"  CFLAGS="-pthread -Wno-long-long" \
            CXX="g++" CXXFLAGS="-pthread -Wno-long-long" \
            LDFLAGS="-L$DOWNWARD_CPLEX_ROOT/lib/x86-64_linux
/static_pic \
                     -L$DOWNWARD_SOPLEX_ROOT/lib" \
            --without-lapack --enable-static=no \
            --prefix="$DOWNWARD_COIN_ROOT" \
            --disable-bzlib \
            --with-soplex-incdir=$DOWNWARD_SOPLEX_ROOT/include \
            --with-soplex-lib="-lsoplex" \
            --with-cplex-incdir=$DOWNWARD_CPLEX_ROOT/include
/ilcplex \
            --with-cplex-lib="-lcplex -lm -ldl" # -ldl is only
needed for CPLEX >= 12.8

make
make install
cd ..
rm -rf Osi-0.107.9
rm Osi-0.107.9.tgz
```

After installation, permanently set the environment variable DOWNWARD_COIN_ROOT to the value you used in installation.

If you don't want to permanently modify your environment, you can also set these variables directly when calling CMake, or in `src/build.py`. The variables need to be set for when installing the Open Solver Interface (Step 2.) and when building Fast Downward's

search component (Step 3.).

Installing the Open Solver Interface on macOS

Follow the Linux instructions above with the following change:

- Building `CoinUtils` can create both static and dynamic libraries and there have been reported problems with both.
  - To use static libraries, replace `--enable-static=no` with `--enable-static=yes` and delete the dynamic libraries from `$(DOWNWARD_COIN_ROOT)/lib/` after building `CoinUtils`.
  - **or**
  - To use dynamic libraries, keep using `--enable-static=no` and change the Fast Downward CMakefile to look for dynamic (*.dylib) instead of static (*.a) libraries of libOsiCpx, libOsiGrb, and libOsiClp.

Installation of version 0.103.0 has been reported to fail on macOS due to a bug in the `CoinUtils` configure script. (See issue295.) If you run into this problem, try downloading the patched file attached to this page, make it executable, and replace the file `Osi-0.103.0/CoinUtils/configure` with it:

```
    cp Dowloads/coinutils-configure.patched Osi-0.103.0/CoinUtils
/configure
    chmod +x Osi-0.103.0/CoinUtils/configure
```

Installing the Open Solver Interface on Windows

We managed to install OSI natively with support for CPLEX on Windows using the following steps:

1. We assume CPLEX is installed in `C:\cplex` and
2. Open the command line and execute `vcvarsall.bat x64` of Visual Studio (see ObtainingAndRunningFastDownward#Compiling_on_Windows)
3. Download the zlib source code (https://www.zlib.net/). We use the current (Jan 2021) version 1.2.11. We assume your unpacked directory is called `zlib_sources`. Next compile it and set up the include and lib directories within `zlib\`. Let the variable `ZLIB_ROOT` point to `zlib\`.

   Zeilennummern ein/ausschalten Zeilennummern ein/ausschalten

   ```
   1  mkdir zlib
   2  mv zlib_sources zlib\include
   3  cd zlib\include
   4  nmake /f win32/Makefile.msc
   5  mkdir ..\lib
   6  move zdll.lib ..\lib\zdll.lib
   7  move zlib.lib ..\lib\zlib.lib
   8  move zlib1.dll ..\lib\zlib1.dll
   ```

4. Download OSI (we use 0.107.9, newer versions should work but are untested). We assume your unpacked directory is called `osi`. On Windows, we have to create two directories for the libraries, one for the debug and one for the release mode. First, we

create the `include` directories:

```
Zeilennummern ein/ausschalten  Zeilennummern ein/ausschalten

   1     mkdir osi_release\include
   2     cd osi
   3     copy CoinUtils\src\*.hpp ..\osi_release\include
   4     copy CoinUtils\src\*.h ..\osi_release\include
   5     copy Osi\src\Osi\*.hpp ..\osi_release\include
   6     copy Osi\src\Osi\*.h ..\osi_release\include
   7     copy Osi\src\OsiCpx\*.hpp ..\osi_release\include
   8     copy Osi\src\OsiCpx\*.h ..\osi_release\include
   9     Xcopy /E /I ..\osi_release\include ..\osi_debug
\include
```

5. To statically link the LP solvers, we have to compile the libraries with static linking. At the time being (Jan 2021), setting this parameter required the Visual Studio GUI. We wrote our own script which workes without a GUI. For easier access, we create a variable to call the script.

```
Zeilennummern ein/ausschalten  Zeilennummern ein/ausschalten

   1     set SET_RUNTIME_LIBRARY=python PATH\TO\REPOSITORY
\github\workflows\misc\set-visual-studio-static-runtime-
libraries.py
```

6. Next, we compile libOsi. We use `devenv` to update the Visual Studio project file for your Visual Studio version.

```
Zeilennummern ein/ausschalten  Zeilennummern ein/ausschalten

   1     mkdir ..\osi_release\lib
   2     mkdir ..\osi_debug\lib
   3
   4     cd Osi\MSVisualStudio\v10\
   5     devenv Osi.sln /Upgrade
   6     cd libOsi\
   7     %SET_RUNTIME_LIBRARY% libOsi.vcxproj
libOsi.vcxproj
   8     msbuild libOsi.vcxproj /p:Configuration=Release
/p:Platform=x64
/p:DefaultWindowsSDKVersion=%WindowsSDKVersion%
/p:OutDir=lib
   9     move lib\* ..\..\..\..\..\osi_release\lib\
  10     msbuild libOsi.vcxproj /p:Configuration=Debug
/p:Platform=x64
/p:DefaultWindowsSDKVersion=%WindowsSDKVersion%
/p:OutDir=lib
  11     move lib\* ..\..\..\..\..\osi_debug\lib\
  12
```

7. To use CPLEX with Osi, we compile the binding library. There is no Visual Studio project for this, thus, we compile it manually.

```
Zeilennummern ein/ausschalten  Zeilennummern ein/ausschalten

   1     cd ..\..\..\src\OsiCpx
   2     cl /EHsc OsiCpxSolverInterface.cpp /I ..\Osi /I
..\..\..\CoinUtils\src /I "%DOWNWARD_CPLEX_ROOT%\include
\ilcplex" /c
   3     lib OsiCpxSolverInterface.obj
   4     move OsiCpxSolverInterface.lib ..\..\..
```

```
\..\osi_release\lib\libOsiCpx.lib
   5     cl /EHsc OsiCpxSolverInterface.cpp /I ..\Osi /I
..\..\..\CoinUtils\src /I "%DOWNWARD_CPLEX_ROOT%\include
\ilcplex" /c /MTd
   6     lib OsiCpxSolverInterface.obj
   7     move OsiCpxSolverInterface.lib ..\..\..
\..\osi_debug\lib\libOsiCpx.lib
```

8. At last, we compile libCoinUtils.

Zeilennummern ein/ausschalten Zeilennummern ein/ausschalten

```
   1     cd ..\..\..\CoinUtils\MSVisualStudio\v10
   2     devenv CoinUtils.sln /Upgrade
   3     cd libCoinUtils
   4     %SET_RUNTIME_LIBRARY% libCoinUtils.vcxproj
libCoinUtils.vcxproj
   5     msbuild libCoinUtils.vcxproj
/p:Configuration=Release /p:Platform=x64
/p:DefaultWindowsSDKVersion=%WindowsSDKVersion%
/p:OutDir=lib
   6     move lib\* ..\..\..\..\..\osi_release\lib\
   7     msbuild libCoinUtils.vcxproj /p:Configuration=Debug
/p:Platform=x64
/p:DefaultWindowsSDKVersion=%WindowsSDKVersion%
/p:OutDir=lib
   8     move lib\* ..\..\..\..\..\osi_debug\lib\
   9
```

9. Finally, set the `DOWNWARD_COIN_ROOT_RELEASE` and `DOWNWARD_COIN_ROOT_DEBUG` variables to point to `osi_release` and `osi_debug`.

Step 3. Building Fast Downward with LP support

Once OSI is installed, you can build Fast Downward's search component with LP support by calling `./build.py`. Remove your previous build first:

```
rm -rf builds
```

Fast Downward automatically includes an LP Solver in the build, if it is needed, and the solver and the necessary OSI adapter are detected on the system. If you want to explicitly build without the LP solver that is installed on your system, disable the LP solver plugin (with `-DPLUGIN_LP_SOLVER_ENABLED=FALSE`) and all plug-ins that require it, such as `POTENTIAL_HEURISTICS` (see ObtainingAndRunningFastDownward#Manual_Builds).

Troubleshooting

The LP-related libraries have a number of dependencies which might not be installed on your system. If for some reason one of the above steps fails, we would appreciate if you could attempt to troubleshoot it yourself.

If the configure step of COIN/OSI fails with the error
`Cannot find symbol(s) CPXgetstat with CPX`, check the path to the

CPLEX lib directory. Depending on your CPLEX installation, the libraries might be stored in a different place, e.g., in `$DOWNWARD_CPLEX_ROOT/lib/x86-64_sles10_4.1/static_pic`. Check that the directory exists and contains the file `libcplex.a`. In some cases with the same error message, the library `dl` has to be added (see this question on stackoverflow). If the configure step fails for another reason, have a look at their troubleshooting page or their page of current issues.

If you get warnings about unresolved references with CPLEX, visit their help pages.

If Fast Downward gives you compilation errors mentioning the `isnan` function, you are likely affected by an OSI/compiler incompatibility (see http://issues.fast-downward.org/issue681). In this case try the following:

1. Add `export DOWNWARD_USE_ISNAN_WORKAROUND=1` to your `~/.bashrc` file (or other shell configuration file).
2. Make sure the changed shell configuration takes effect by logging out and then logging in again.
3. In the main directory of the repository, delete all cmake build artifacts with `rm -r builds`.
4. Rebuild.

If you compiled Fast Downward on Windows (especially on GitHub Actions) and cannot execute the binary in a new command line, then it might be unable to find a dynamically linked library. Use `dumpbin /dependents PATH\TO\DOWNWARD\BINARY` to list all required libraries and ensure that they can be found in your `PATH` variable.

If after troubleshooting you can get the LP package to work, please do let us know of your problem and its solution so that we can improve these instructions. If you still cannot get it to work, we may be able to provide some help, but note that the LP solvers and OSI library are external packages not developed by us.