Exercise: Modeling the Binpacking Problem

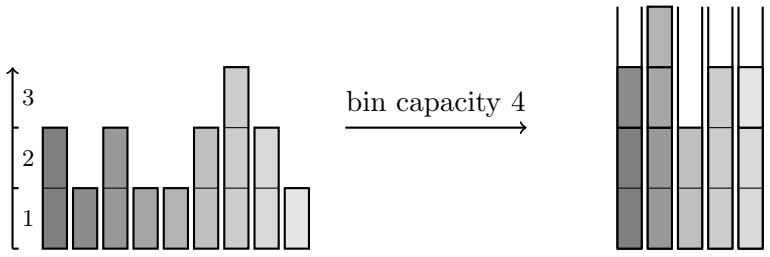
In this exercise, we consider the binpacking problem. We will model this problem (using zimpl) with an assignment formulation and with a formulation problem.

Problem description

The *binpacking problem* is to assign a given set of $n \in \mathbb{N}$ items, each of a size $s_i \in \mathbb{N}$, to bins with capacity $\kappa \in \mathbb{N}$, such that the used bins are not overloaded. The goal is, the minimize the number of used bins.

Example

Given 9 items with sizes: 2, 1, 2, 1, 1, 2, 3, 2, and 1 and a bin capacity of $\kappa = 4$, the following pictures show a feasible solution which needs 5 bins.



Items Assignment

Integer programming formulations

In this exercise, we will consider two formulations of the binpacking problem.

One way of formulating this problem as an integer program, is to use binary variables y_{ij} for each item i and bin j. If and only if y_{ij} is one, item i is assigned to bin j. This formulation we call *assignment* formulation.

Another way is to use a binary variable x_s for each feasible packing s which states if this packing is used or not. A *packing* is a subset of items. It is *feasible*, if and only if the total size of the items contained in this set is not greater than the given capacity κ.

Exercise

Model the binpacking problem as an assignment and as a set covering formulation using zimpl. Afterwards, run automated tests that check which of these two formulations performs better. Therefore, you get a test set, the scripts required for the automated test run, and instruction how to run these tests.

Documentation about zimpl can be found in at `http://zimpl.zib.de`. The following instruction will guide you through this exercise.

Getting started

(a) First you have to fix a bug in SCIP. Therefore, you have to download the file `reader_zpl.c`, put it into the source directory `src/scip`, and recompile SCIP using the command `make` in the home directory of SCIP.
`http://co-at-work.zib.de/berlin2009/exercises/0930/`

(b) Extract the binpacking project `COatWork-Binpacking.tgz`
(`tar xzf COatWork-Binpacking.tgz`). Amongst others, it contains:

`zimpl/assignment.zpl` You should use this file to model the assignment formulation for the binpacking problem. It already includes the parsing part.

`zimpl/setcovering.zpl` You should use this file to model the set covering formulation for the binpacking problem. It already includes the parsing part.

`Makefile` Makefile for the binpacking project. In particular, it includes the functionality to start automated test runs for two models.

`data/*.bpa` Some binpacking instances in the binpacking format.

`check/*` In this directory you find all scripts which are used to perform and evaluate automated tests run for SCIP and CPLEX.

(c) In the folder `COatWork-Binpacking/lib`, create a softlink to your SCIP directory, e.g.
`ln -s /home/coatwork/software/ziboptsuite-1.2.0/scip-1.2.0 scip`

**For each section, please read all given instructions,
before you start working on it.**

Modeling

In the first part of this exercise, you should model the binpacking problem using ZIMPL in two different ways.

Assignment formulation

(a) Open the file `COatWork-Binpacking/zimpl/assignment.zpl` and model the binpacking problem as an assignment formulation.

(b) The data parsing part is already given.

- `capacity` this parameter stores the bin capacity which is the same for all bins
- `nitems` gives you the number of items to pack
- `I` is an index set for the items. This mean, `I` $= \{1, \ldots,$ `nitems`$\}$.
- `sizes` is a map which maps the item index to the size of the item

(c) First, you should define the required decision variables. You need (**at least**) binary variables y_{ij} which determine whether item i is assigned to bin j.

(d) Finally, the constraints have to be formulated. Note, that `ord(sizes[i],1,1))` gives you the size of item i.

(e) Test your model on one of the smaller test instances, such as `u20_00.bpa`. To do so, you can run ZIMPL on its own in the main directory of the project as follows:
`zimpl -DDATAFILE=data/u20_00.bpa zimpl/assignment.zpl`
This creates an LP file with the name `assignment.lp` which can solve with CPLEX or SCIP.
Or, you can directly read the ZIMPL model into SCIP. This can be done as follows:

```
SCIP> set reading zplreader parameter -DDATAFILE=../data/u20_00.bpa
SCIP> read zimpl/assignment.zpl
SCIP> optimize
```

Set covering formulation

(a) Open the file `COatWork-Binpacking/zimpl/setcovering.zpl` and model the bin-packing problem as a set covering formulation.

(b) The data parsing part is already given. It is the same as for the assignment formulation (see above).

(c) First, you should define the set of feasible packings. Therefore, the ZIMPL functions `subsets` and `indexset` might be of interest.

- `subsets(I,n,m)` generates all subsets of I of a cardinality between n and m. That is, $\{X \mid X \subseteq I \wedge n \leq |I| \leq m\}$.
- For some set `I`, `indexset(I)` is the indexset of I. That is $\{1, \ldots, |I|\}$.

(d) Define the required decision variables for each feasible packing.

(e) Finally, the constraints have to be formulated. To determine if i is an element of P you can use the following ZIMPL expression: `card({i} inter P) != 0`.
If this expression evaluates to `true`, element i is part of the set P.

(f) Test your model on one of the smaller test instances. Follow the instruction from above and use the ZIMPL model `setcovering.zpl` instead of `assignment.zpl`.

Automated test runs

After you have implemented the assignment and the set covering formulation, you can run automated tests to check which of the two formulations performs better. All required scripts are included in this project. You can find them in the `check/` folder. This section explains you how to run these tests for the binpacking models and how to compare the results.

All test runs are started in the main directory of the project and get initialized via the `Makefile`. Depending on the requested solver (CPLEX or SCIP) a bash script is called which sequentially schedules all instances to be run. That is for CPLEX `check/check_cplex.sh` and for SCIP `check/check_scip.sh`. After that, a solver specific `awk` scripts parses the generated log files and produces a summary table.

SCIP test run

In order to start a SCIP test run you just have to use the target `testscip` of the `Makefile`. This means, you call in the main directory of the project `make testscip`. This will start

an automated test run using SCIP as solver engine and the default values for the test set, time limit, memory limit, and the ZIMPL model. Besides other, the following options are available:

- `TEST` - test set name [default: binpacking]
- `TIME` - time limit for per instance in seconds [default: 30]
- `MEM` - memory limit in MB [default: 1536]
- `MODEL` - used ZIMPL model [default: assignment]

For example, to run a test using 512 MB of memory, a time limit of 60 seconds, and the set covering model, you should call:

```
> make testscip TIME=60 MEM=512 MODEL=setcovering
```

The test run will generate three files which are located in the directory `check/results/`.

- `*.out` - Includes the output generated by the solver via the stdout
- `*.err` - Includes the output generated by the solver via the stderr, such as error methods
- `*.res` - ASCII table containing a summary of the computational results

The base name of all these files is the same and has the following structure which allows to reconstruct the test run.

`check.<model name>.<test name>.scip`

- "model name" indicates to used ZIMPL model, e.g., "assignment"
- "test name" indicates the name of the the test file, e.g., "binpacking"

CPLEX test run

In order to start a test run with CPLEX, you just have to use the `Makefile` target `testcplex` instead of `testscip`. The rest remains the same as in the SCIP test run case.

Exercise

Perform a test run for both models using the SCIP and CPLEX solver engine. This gives you four different test runs.

Compare results

The scripts which are needed to compare the results are also included in the project and located in the `check/` folder. To use them, you have to switch to the `check/` folder and apply the following command:

```
./allcmpres.sh results/check.*.res
```

This produces a overall table comparing all four runs. On the web page

`http://scip.zib.de/doc/html/TEST.html`

you find a "HowTo" the explanation to these result tables as well as the testing environment which comes with the SCIP release.

Good luck!