Exercise: Column Generation for Binpacking

This exercise shows how to implement a solver for the binpacking problem using a formulation with exponentially many variables, each representing a feasible packing of a bin. Therefore, we use a pricer which dynamically generates new variables with negative reduced costs and adds them to the master problem. The pricer has to solve a combinatorial optimization problem to generate a new variable (feasible packing) with negative reduced cost.

Problem description

The binpacking problem consists of the task to distribute a given set of items $[n] := \{1, \ldots, n\}$ with nonnegative size s_i to a minimal number of bins, all of the same capacity κ. One possible formulation is the following integer program:

$$\begin{aligned} \min \quad & \sum_{s \in \mathcal{S}} x_s & \\ \text{subject to} \quad & \sum_{s \in \mathcal{S}} (\lambda_s)_i \, x_s \geq 1 & \forall i \in \mathcal{I} \\ & x_s \in \mathbb{Z}^+ & \forall s \in \mathcal{S} \end{aligned}$$

Here, the set $\mathcal{S} := \{S \subseteq [n] \mid \sum_{i:i \in S} s_i \leq \kappa\}$ contains all feasible packing patterns.

Since $\mathcal{S}$ can be of exponential size, we will use a column generation approach to solve this problem. We initialize the problem with a set of n variables representing packings of a single item per bin.

Now, we have to iteratively search for variables representing "better" packings, i.e., a packing pattern which reduces the overall costs. For a given solution y^* of the (restricted) dual linear program, we have to find a variable λ_S for which the reduced costs

$$c_S - \sum_{i \in S} y_i^* < 0 \Leftrightarrow \sum_{i \in S} y_i^* > 1$$

since all variables λ_S have an objective coefficient $c_S = 1$.

Getting started

The binpacking project you already used in the previous exercise contains all necessary files for this exercise. Amongst others, it contains:

`src/reader_bpa.{c,h}` This reader parses the binpacking problem files and creates the master set covering formulation. It is **already** implemented.

`src/pricer_binpacking{c,h}` These files contain an almost completed version of the pricer you have to implement.

The pricer

In the following we give a road map through this exercise.

(a) Formulate the pricing problem of the set covering formulation as a combinatorial optimization problem.

(b) Open the file `COatWork-Binpacking/src/pricer_binpacking.c` and search for the callback method `pricerRedcostBinpacking`.

(c) Implement the missing parts of this method. These are indicated by `TODO` in the source file. The following methods might help:

- `SCIPpricerGetData(SCIP_PRICER*)` gives you the `SCIP_PRICERDATA*` structure which is defined at the beginning of `pricer_binpacking.c`. The pricer data structure consists of four members. This are:
  - `SCIP_CONS** conss` set covering constraints for the items
  - `SCIP_Longint* weights` size of the items
  - `int nweights` number of weights (items) to be packed
  - `SCIP_Real capacity` capacity of the bins
- The dual solution of a (LP row corresponding to a) set covering constraint is given by `SCIPgetDualsolSetppc(SCIP*, SCIP_CONS*)`.
- Use `SCIPsolveKnapsackExactly()` to solve the knapsack problem. This method is located and documented in the constraint handler `cons_knapsack.h`.
  Note, that this method will resort the `weights`, `profits`, and `items` arrays.
- `SCIPcreateVar()` to create a new variable. Note, that the last four parameters of this method are `NULL` in our case and the two `SCIP_Bool` parameters should be set to `TRUE`.
- `SCIPaddPricedVar(SCIP*, SCIP_VAR*, SCIP_Real)` passes a new variable to SCIP.
- `SCIP_RETCODE SCIPaddCoefSetppc(SCIP*, SCIP_CONS*, SCIP_VAR*)` adds a variable to set covering constraint.
- `SCIPreleaseVar(SCIP*, SCIP_VAR**)` releases a variable.

Hints:

- The pricing score can always be set to 1.0.

Final test

Finally, you should compile and test your solution. For testing, open the binary at `bin/binpacking` and `read` one of the smaller test instances which you find in the directory `data/`.

If everything works fine, you should run a automated test using the command `make test` in the main directory of the project.

Good luck!