Vienna University of Technology

Modeling and Solving Constrained
Optimization Problems

Energy-Cost Aware Scheduling

Jonas Ferdigg BSc. (01226597)

September 6, 2019

4 Compile and Run Instructions

In the Makefile located in the root folder of the project the values of the variables *CPLEX_VERSION* and *CPLEX_DIR* in line 9 and 10 have to be adjusted.

Running *make* in the root folder of the project builds it. Running *make run* builds the project and runs it with the first JSON file (sample01.json).

The executable needs the path of a JSON file as an argument. The usage is as following:

USAGE: ./ecas *instance_file*
EXAMPLE: ./ecas ./data/sample01.json

5 Introduction

I chose to use IBM CP Optimizer with a C++ environment to solve the assignment. It took mi some time to set up the makefile and get all the linkerflags right to compile the project, but I got there. I wrote a parser for the JSON data files (I had to run a JSON-Formatter on them to get the linebreaks right) and a C++ representation for the data in the form of three classes included in the project:

- Machine - Representation of a *machine* in the JSON files
- Task - Representation of a *task* in the JSON files
- Instance - Complete representation of the problem instance

This way, I could easily access the data when writing the code for the model. I added the class *ECAS* (Energy-Cost Aware Scheduling) inheriting the *initCP* class to function as the core piece of the model. In the method *initModel* the model for the assignment is defined which I will elaborate in the next chapter.

6 Model

I had some trouble setting up the model as I was not familiar withe the framework at all and it is not really the most intuitive one. I ran into more than one odyssey while doing so but this is the final result. I am fairly sure that the model is correct, but I didn't get it to terminate once. This might be due to the slow hardware I am using or the lack of optimizing constraints which I did not have time to implement.

6.1 Structure

The core piece of my model are the following and will be explained in detail later on:

- *IloIntervalVarArray* tasks

- *IloIntervalVarArray2* taskOnMachine
- *IloIntervalVarArray2* machineHasTask
- *IloCumulFunctionExprArray* machineCPURes
- *IloCumulFunctionExprArray* machineMemRes
- *IloCumulFunctionExprArray* machineIORes
- *IloIntervalVarArray2* machinePowerOnIntervals
- *IloCumulFunctionExprArray* machinePowerOn
- *IloNumToNumStepFunction* energyPrices
- *IloNumExpr* energyCosts

6.1.1 *IloIntervalVarArray* tasks

The model representation of the tasks listen in the JSON data files. They are of course initialized with a preset size and as non-optional.

6.1.2 *IloIntervalVarArray2* taskOnMachine

An array of interval variables with the *task.ID* as a row index and *machine.ID* as column index and intervals with the same size set as in the previously defined *tasks* array but set as optional (with *setOptional*). With the function *Alternative* the tasks defined in the previous array are given the option to run on any machine.

6.1.3 *IloIntervalVarArray2* machineOnTask

The transposed *taskOnMachine* matrix. This representation is often more convenient to loop over but is not inherently necessary.

6.1.4 *IloCumulFunctionExprArray* machineCPURes

The cumulative functions defined in this array are summing up the resources needed by tasks running on the corresponding machine. They are used in combination with the constraint *machineCPURes[m.id] ¡= m.cpu_cap()* to ensure that the resources needed by the tasks running on a machine remain lower than the capacity of the machine. The same applies for the cumulative functions *machineMemRes* and *machineIORes*.

6.1.5 *IloIntervalVarArray2* machinePowerOnIntervals

The intervals defined in this matrix represent the time in which a machine is running. There are Tasks + 1 such intervals per machine. They are defined as optional and with no preset duration. They are constrained by the fact that in each row of the matrix *machinePowerOnIntervals[i][j]*, *machinePowerOnIntervals[i-1][j]* must be present for *machinePowerOnIntervals[i][j]* to be present (except for i=0 of course) and that *machinePowerOnIntervals[i-1][j]* must have ended before *machinePowerOnIntervals[i][j]* can start (and there has to be 1 time-tick in between). This method of chained optional intervals is also discussed in the CP Optimizer tutorial (Introduction to CP Optimizer for Scheduling) on slide 109.

6.1.6 *IloCumulFunctionExprArray* machinePowerOn

This cumulative function is the sum of pulses of the intervals *machinePowerOnIntervals* with height 1. This cumulative function binds the presence of one or more *machineHasTask* intervals and a *machinePowerOnIntervals* together when the *IloAlwaysIn(env, machinePowerOn[m.id], machineHasTask[m.id][t.id], 1, 1)* constrained is applied. This forces a machinePowerOnInterval to be present if a machine is assigned a task at that time.

6.1.7 *IloNumToNumStepFunction* energyPrices

The piece-wise linear function *energyPrices* is the time dependent representation of the evolving energy prices as defined in the JSON data file.

6.1.8 *IloNumExpr* energyCosts

This numerical expression is used as the optimization function for the solver. It adds up energy expenses from tasks and machines as defined the project assignment. For tasks it includes the exact energy costs for executing it. For machines I did not figure out how to include an exact calculation of the idle energy consumption as the duration of the *machinePowerOnIntervals* is not known at compile time so I could not apply the same method as for the tasks. The calculation I use as an approximation evaluates both start and end time of *machinePowerOnIntervals* using the *StartEval* and *EndEval* functions provided by the framework. I calculated the estimated energy costs by adding the energy prices $E(t)$ at the start and end of the intervals a_i multiplied by the size $size(a_i)$ of the interval and multiplied by the power consumption $C(a_i)$ of the interval divided by 2 to get an average between the price at the start and the end of the interval:

$$\sum_i \frac{(E(startOf(a_i)) - E(endOf(a_i))) * size(a_i) * C(a_i)}{2}$$

Of course power down and power up costs are also included in the objective function.

7 Pitfalls

One pitfall in which i fell into was the assumption that the power state of the machines has to be modeled with *IloStateFunctions*. This seemed natural in my head (power state => state function) so this was the first method I tried. I kept skimming through the CP Optimizer documentation and code examples trying to find a way of including an *IloStateFunction* in the objective function but did not find one in the end. I lost a lot of time on that one.

Another one was the assumption that I could use an ordinary C++ *double* array in the objective function which of course, did not work. I only found out today that I had to convert it to an *IloNumArray* so it can be used in the objective function. Before today I thought it was not possible to use an array in the objective function.