CS 517: A tool for Job Shop Problem
Final Report

Wei-Jen Chen, Weijie Mo

Abstract

Job Shop Scheduling problem is a complicated scheduling problem. In JSS problem, there are many different tasks which can be processed by many different machine at many different time. In this document, we will show the reduction to 3-SAT problem and create a tool to solve this problem by z3 sat-solver. We will evaluate the tool by comparing the behavior between our tool and Google OR tool.

Preface

At first we are planning to do the research in the topic that not many people did before. Therefore we choose to do research in Kirkman's schoolgirl problem. Kirkman's schoolgirl problem(1850s) is a combinatorial optimisation scheduling problem, and it can be transformed to another well-known social golfer problem(1998). However, in "An Improved SAT Formulation for the Social Golfer Problem"[1], we learn that this problem is NP-complete only if when deciding whether a partially filled schedule can be completed to conflict-free one, and the computational complexity of this problem is currently unknown. Therefore, we plan to do another scheduling problem we found in Google OR-Tools project, which is Job Shop Scheduling problem(JSS)[2]. At the meantime, we found a very detailed tutorial for z3 SAT-solver written by two Microsoft employee[3]. Therefore, our new plan is that we will build a tool to solve JSS by using z3 SAT solver in python, and we can compare our tool to Google OR-Tool in evaluation section if necessary.

Background

JSS is an optimization problem in CS and operation research, and it can be extended to many different problem depend on the given "constraints". In this document, we will follow the Google Project's version[2]: Suppose there are many jobs which are processed by several machines, Job shop scheduling problem is to minimize the length of the schedule under the following constraints:

1. Each job consists of a sequence of tasks, which must be performed in a given order
2. Each task must be processed on a specific(correspond) machine
3. No task can be started until the previous task is completed

4. A machine can only work on one task at one time

5. Once a machine start a task, it need to run to complete

Here is the example from Google's project: Each task below is labeled by a pair of numbers (m, p) where m is the number of the machine the task must be processed on and p is the processing time of the task — the amount of time it requires. (The numbering of jobs and machines starts at 0.)

- job 0 = [(0, 3), (1, 2), (2, 2)]
- job 1 = [(0, 2), (2, 1), (1, 4)]
- job 2 = [(1, 4), (2, 3)]

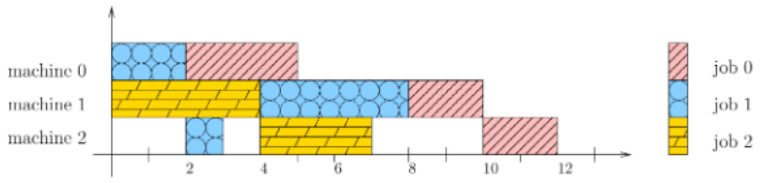

Figure 1: one possible solution for Google JSS example

Reduction

First, we define some variable for JSS problem. We have start ($S_{i,j}$) for the start time of j-th task for job i. Each task start at $S_{i,j}$ and run $T_{i,j}$ units of time, where $T_{i,j}$ denotes the duration of j-th task of job i. In classic JSS problem, there are only 2 constraints:

1. Precedence
   A precedence constraint between two consecutive tasks $T_{i,j}$ and $T_{i,j+1}$ is encoded using the inequality $S_{i,j+1} \geq S_{i,j} + S_{i,j}$. This inequality states that the start-time of task $j+1$ must be greater than or equal to the start-time of task j plus its duration.

2. Resource Capacity
   A resource constraint between two tasks from different jobs i and i' requiring the same machine j is encoded using the formula
   $(S_{i,j} \geq S_{i',j} + T_{i',j}) \vee (S_{i',j} \geq S_{i,j} + T_{i,j})$, which states that the two tasks do not overlap.

Moreover, other than the 2 main constraints above, we need some other constraint in programming.

3. Start-time for the first task of each job need to be greater or equal to zero, that is $S_{i,0} \geq 0$

4. At the end, we define a variable max(makespan). And we want to decide if there is a schedule such that end-time of the last task is less or equal max time, that is $S_{i,n} + T_{i,n} \leq max$(makespan), where n is the index number of last task for each job.

We conclude the constraints in arbitrary sentence in the following table.

Constraint #number	satisfaction sentence	Description
Constraint #1 Precedence	$S_{i,j+1} \geq S_{i,j} + T_{i,j}$	No task can be started until the previous task is completed
Constraint #2 Resource capacity	$(S_{i',j} \geq S_{i,j} + T_{i,j}) \vee (S_{i,j} \geq S_{i',j} + T_{i',j})$	A machine can only work on one task at one time

We will continue using the example from Google project to generate the SMT formula:

duration time / $T_{i,j}$	Machine 0	Machine 1	Machine 2
Job 0	3 / $T_{0,0}$	2 / $T_{0,1}$	2 / $T_{0,2}$
Job 1	2 / $T_{1,0}$	4 / $T_{1,1}$	1 / $T_{1,2}$
Job 2		3 / $T_{2,1}$	3 / $T_{2,2}$

Let max = 10

Encoding:

$(S_{0,0} \geq 0) \wedge (S_{0,1} \geq S_{0,0} + 3) \wedge (S_{0,2} \geq S_{0,1} + 2) \wedge (S_{0,2} + 2 \leq 10) \wedge$
$(S_{1,0} \geq 0) \wedge (S_{1,1} \geq S_{1,0} + 2) \wedge (S_{1,2} \geq S_{1,1} + 4) \wedge (S_{1,2} + 1 \leq 10) \wedge$
$(S_{2,1} \geq 0) \wedge (S_{2,2} \geq S_{2,1} + 3) \wedge (S_{2,2} + 3 \leq 10) \wedge$
$((S_{0,0} \geq S_{1,0} + 2) \vee (S_{1,0} \geq S_{0,0} + 3)) \wedge$
$((S_{0,1} \geq S_{1,1} + 4) \vee (S_{1,1} \geq S_{0,1} + 2)) \wedge$
$((S_{0,1} \geq S_{2,1} + 3) \vee (S_{2,1} \geq S_{0,1} + 2)) \wedge$
$((S_{1,1} \geq S_{2,1} + 3) \vee (S_{2,1} \geq S_{1,1} + 4)) \wedge$
$((S_{0,2} \geq S_{1,2} + 1) \vee (S_{1,2} \geq S_{0,2} + 2)) \wedge$
$((S_{0,2} \geq S_{2,2} + 3) \vee (S_{2,2} \geq S_{0,2} + 2)) \wedge$
$((S_{1,2} \geq S_{2,2} + 3) \vee (S_{2,2} \geq S_{1,2} + 1))$

Code

We will show our partial code in the following table. For completed code, please check our github at the end of this chapter.

```
# items can not overlap:
def no_items_overlap (s, lst):
    for pair in itertools.combinations(lst, r = 2):
        no_interval_overlap(s, (pair[0][1], pair[0][2]), (pair[1][1], pair[1][2]))
# intervals can not overlap:
def no_interval_overlap (s, i1, i2):
    (i1_begin, i1_end) = i1
    (i2_begin, i2_end) = i2
    s.add(Or(i1_begin ≥ i2_end, i1_end ¡= i2_begin))
for job in range(len(jobs)):
    former_task_end = None
    jobs_list_tmp = []
    for t in jobs[job]:
        machine = t[0]
        time_used = t[1]
#set variables:
    begin = Int('j_%d_task_%d_%d_begin' % (job, machine, time_used))
    end = Int('j_%d_task_%d_%d_end' % (job, machine, time_used))
#check whether it is avaliable to add tasks
    if (begin,end) not in tasks_machines[machine]:
        tasks_machines[machine].append((job,begin,end))
#check whether it is avaliable to add jobs
    if (begin,end) not in jobs_list_tmp:
        jobs_list_tmp.append((job,begin,end))
# task start from time ≥ 0
    sol.add(begin ≥ 0)
# no task end after makespan:
    sol.add(end ≤ makespan)
# end time is fixed with begin time:
    sol.add(end == begin+time_used)
# no task begin before the last task end:
    if former_task_end != None:
        sol.add(begin ≥ former_task_end)
        former_task_end = end
        jobs_list.append(jobs_list_tmp)
```

```
# no tasks overlap on machines:
for tasks_for_machine in tasks_machines:
    no_items_overlap(sol, tasks_for_machine)
# no tasks overlap on each jobs:
for jobs_list_tmp in jobs_list:
    no_items_overlap(sol, jobs_list_tmp)
    min = sol.minimize(makespan)
#unknown = CheckSatResult(Z3_L_UNDEF)
if sol.check() == unknown:
    print ("the result is unknown")
    exit(0)
#sat = CheckSatResult(Z3_L_TRUE)
#unsat = CheckSatResult(Z3_L_FALSE)
elif sol.check() == unsat:
    print ("the result is unsat")
    exit(0)
#sol.lower is to checkensure result returned by maximize/minimize
    sol.lower(min)
#m_sol is to return optimized machine using schedule
    m_sol = sol.model()
```

Github: https://github.com/javamore/SAT-jobshop/blob/master/jobshop.py

Evaluation

We have done several sets of tests by using different size of data(*machine* x *job*), and we let the number of tasks equal to the number of jobs(tasks can be less than jobs). We use the data set retrieved from SAS web site[6], one of the test result is shown as follow:

Test result: 10x10

Machine Number: 10

- Job0: [(2,44),(3,5),(5,58),(4,97),(0,9),(7,84),(8,77),(9,96),(1,58),(6,89)],
- Job1: [(4,15),(7,1),(1,87),(8,57),(0,77),(3,85),(2,81),(5,39),(9,73),(6,21)],
- job2: [(9,82),(6,22),(4,10),(3,70),(1,49),(0,40),(8,34),(2,48),(7,80),(5,71)],
- job3: [(1,91),(2,17),(7,62),(5,75),(8,47),(4,11),(3,7),(6,72),(9,35),(0,55)],
- job4: [(6,71),(1,90),(3,75),(0,64),(2,94),(8,15),(4,12),(7,67),(9,20),(5,50)],
- job5: [(7,70),(5,93),(8,77),(2,29),(4,58),(6,93),(3,68),(1,57),(9,7),(0,52)],

- job6: [(6,87),(1,63),(4,26),(5,6),(2,82),(3,27),(7,56),(8,48),(9,36),(0,95)],
- job7: [(0,36),(5,15),(8,41),(9,78),(3,76),(6,84),(4,30),(7,76),(2,36),(1,8)],
- job8: [(5,88),(2,81),(3,13),(6,82),(4,54),(7,13),(8,29),(9,40),(1,78),(0,75)],
- job9: [(9,88),(4,54),(6,64),(7,32),(0,52),(2,6),(8,54),(5,82),(3,6),(1,26)]

Result of google:

```
Optimal Schedule Length: 842
Machine 0: job_7_0   job_9_4   job_0_4   job_1_4   job_4_3   job_2_5   job_3_9   job_8_9   job_6_9   job_5_9
           [8,44]    [255,307] [308,317] [317,394] [394,458] [495,535] [557,612] [612,687] [689,784] [789,841]
Machine 1: job_3_0   job_1_2   job_4_1   job_6_1   job_2_4   job_8_8   job_0_8   job_5_7   job_9_9   job_7_9
           [0,91]    [91,178]  [178,268] [268,331] [446,495] [514,592] [653,711] [711,768] [788,814] [829,837]
Machine 2: job_0_0   job_3_1   job_8_1   job_9_5   job_5_3   job_6_4   job_4_4   job_1_6   job_2_7   job_7_8
           [0,44]    [91,108]  [108,189] [307,313] [331,360] [363,445] [458,552] [562,643] [643,691] [782,818]
Machine 3: job_0_1   job_8_2   job_7_4   job_4_2   job_2_3   job_3_6   job_6_5   job_1_5   job_5_6   job_9_8
           [44,49]   [189,202] [222,298] [298,373] [373,443] [443,450] [450,477] [477,562] [615,683] [782,788]
Machine 4: job_1_0   job_9_1   job_0_3   job_6_2   job_2_2   job_8_4   job_3_5   job_7_6   job_5_4   job_4_6
           [0,15]    [93,147]  [161,258] [331,357] [357,367] [367,421] [421,432] [432,462] [462,520] [594,606]
Machine 5: job_8_0   job_7_1   job_0_2   job_5_1   job_3_3   job_6_3   job_9_7   job_1_7   job_4_9   job_2_9
           [0,88]    [88,103]  [103,161] [161,254] [254,329] [357,363] [432,514] [643,682] [709,759] [771,842]
Machine 6: job_4_0   job_6_0   job_9_2   job_8_3   job_2_1   job_7_5   job_3_7   job_5_5   job_0_9   job_1_9
           [1,72]    [72,159]  [159,223] [223,305] [305,327] [327,411] [450,522] [522,615] [711,800] [800,821]
Machine 7: job_1_1   job_5_0   job_3_2   job_9_3   job_0_5   job_8_5   job_7_7   job_6_6   job_4_7   job_2_8
           [15,46]   [46,116]  [117,179] [223,255] [337,421] [421,434] [462,538] [538,594] [606,673] [691,771]
Machine 8: job_7_2   job_1_3   job_5_2   job_3_4   job_9_6   job_8_6   job_0_6   job_2_6   job_4_5   job_6_7
           [103,144] [178,235] [254,331] [331,378] [378,432] [439,468] [468,545] [545,579] [579,594] [594,642]
Machine 9: job_9_0   job_7_3   job_2_0   job_8_7   job_3_8   job_0_7   job_6_8   job_4_8   job_1_8   job_5_8
           [0,88]    [144,222] [222,304] [469,509] [522,557] [557,653] [653,689] [689,709] [709,782] [782,789]

Time used: 1.1636631488800049
```

Figure 2: 10x10 testing result of Google-OR-tool

Result of ours:

```
machines used :
0
1
2
3
4
5
6
7
8
9

-------------
time schedule=  0  :
7       3       0       .       1       8       4       .       .       9
time schedule=  1  :
7       3       0       .       1       8       4       .       .       9
time schedule=  2  :
7       3       0       .       1       8       4       .       .       9
```

```
time schedule=833  :
5       .       .       .       2       .       .       .       .
time schedule=834  :
5       .       .       .       2       .       .       .       .
time schedule=835  :
5       .       .       .       2       .       .       .       .
time schedule=836  :
5       .       .       .       2       .       .       .       .
time schedule=837  :
5       .       .       .       2       .       .       .       .
time schedule=838  :
5       .       .       .       2       .       .       .       .
time schedule=839  :
5       .       .       .       2       .       .       .       .
time schedule=840  :
5       .       .       .       2       .       .       .       .
time schedule=841  :
.       .       .       .       2       .       .       .       .
Time used: 2.820807933807373
```

Figure 3: 10x10 testing result of our Z3-tool

By testing the data set 6x6, 8x8, 10x10, we learn that our Z3-tool can generate same answer as Google-OR-tool.(Note that our schedule length start with 0 while Google-OR-tool start with 1, therefore our result will be 1 unit time less than google's) However, our performance is not as good as Google's. While the data set comes to 15x15, our tool seems will run forever(more than 15 minutes) while Google's only run 41.51(sec). When the data set comes to 20x20, our tool will run more than 15 minutes, while Google's only run 93.17(sec).

machine x job	6x6	8x8	10x10	15x15	20x20
Google-OR-tool	0.0302(sec)	0.1477 (sec)	1.164 (sec)	41.5129 (sec)	93.1707 (sec)
Our Z3-tool	0.6392(sec)	1.177 (sec)	2.8208 (sec)	N/A	N/A

Figure 4: Result Table

Conclusion

During our research in JSS problem, we learned that there are some other new technics come out recently. For example, in 2004, Reduced Sat Formula (RSF)[4] was published. RSF is a new codification of JSS problem that need fewer clauses in the SAT formula for JSS problem instances than our work. Moreover, this same group come out a more efficient algorithm, which is RandTaunt [5]. However, because we change the topic 2 weeks ago before the deadline, we don't have much time to get familiar with those new idea. Therefore, we use relatively old algorithm to build our tool. In the future, we will try to implement those new ideas to our tool, and make our tool more efficient.

References

[1] Markus Triska, Nysret Musliu. February 2010. *An Improved SAT Formulation for the Social Golfer Problem.* Springer Science+Business Media, LLC 2010

[2] Google-OR-Tool. *The Job Shop Problem.* Goole Developer

[3] Leonardo de Moura, Nikolaj Bjørner. *Z3 - a Tutorial.* Microsoft Research

[4] Solís, Juan Frausto and Marco Antonio Cruz-Chávez. *A Reduced Codification for the Logical Representation of Job Shop Scheduling Problems.* ICCSA (2004).

[5] Frausto-Solís, Juan et al. *A Very Efficient Algorithm to Representing Job Shop Scheduling as Satisfiability Problem. (2008). An Improved SAT Formulation for the Social Golfer Problem.*

[6] SAS website. *SAS/OR(R)9.3 User's Guide: Constraint Programming* http://support.sas.com/documentation/cdl/en/orcpug/63973/HTML/default/viewer.htm#orcpug_clp_sect048.htm