Solving the Simple Assembly Line Balancing Problem with CP Optimizer

Published on December 3, 2020 Edit article View stats

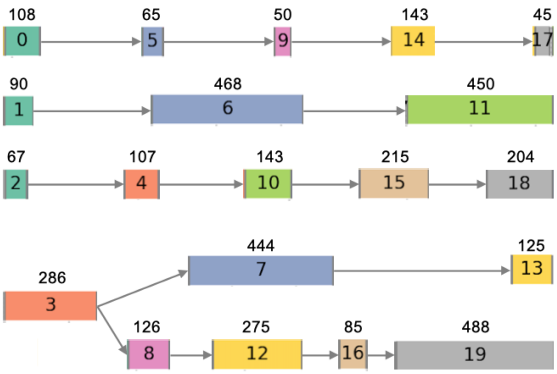
Philippe Laborie
Software Developer: Operations Research, Artificial Intelligence, Optimization, Planning & Scheduling

3 articles

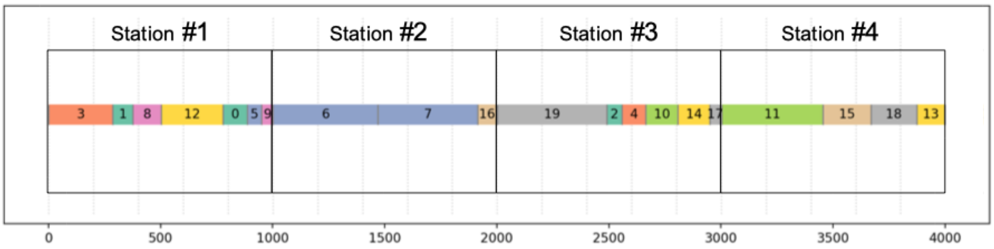
The problem

The Simple Assembly Line Balancing Problem (**SALBP**) is the basic optimization problem in assembly line balancing research. Given is a set of operations each of which has a deterministic duration. The operations are partially ordered by precedence relations defining a precedence graph. The paced assembly line consists of a sequence of (work) stations. In each station a subset of the operations is performed by a single operator. The resulting station time (sum of the respective operation times) must not exceed the cycle time. Concerning the precedence relations, no task is allowed to be executed in an earlier station than any of its predecessors. We consider the version where the cycle time is given and we want to minimize the number of stations (SALBP-1).

Here is an example of precedence graph with 20 operations. The length of the rectangle representing the operations is proportional to the operation's duration given above the rectangle.



Assuming a cycle time of 1000, an example of optimal solution is shown below. The minimal number of stations is 4.



About CP Optimizer

CP Optimizer is available in **CPLEX Optimization Studio**. It provides a modeling language for Combinatorial Optimization Problems that extends Integer Linear Programming (and classical Constraint Programming) with some algebra on intervals and functions allowing compact and maintainable formulations for complex scheduling problems.

CP Optimizer has an automatic solution search process. This search of CP Optimizer is :

- **Complete** : it provides optimality proofs and lower bounds
- **Anytime** : feasible solutions are in general produced quickly
- **Efficient** : it is usually competitive with problem-specific algorithms on classical problems, and the performance improves from version to version
- **Scalable**: the engine can handle problems involving up to several hundreds of thousands of tasks
- **Parallel**: it can exploit all the compute cores available on the machine
- **Deterministic**: solving the same problem twice on the same machine will produce the same result

You can get a free and unlimited version of IBM ILOG CPLEX Optimization Studio, including CP Optimizer, through **Academic Initiative** if you are in academia. CP Optimizer is available in **Python**, Java, C++ (native code) and OPL. I will use Python and OPL here for illustration.

An overview of CP Optimizer (modeling concepts, applications, examples, tools, performance, etc.) is available **here**.

You can get an idea how CP Optimizer is used in the academia or the industry by searching for the references on **GoogleScholar**.

The CP Optimizer formulation for SALBP

Here is all the Python code you need to write in order to (1) read the data, (2) formulate the SALBP in CP Optimizer and (3) solve the problem using the automatic search. The explanation of the formulation is given after the code.

```
# 1. READING THE DATA

import json
with open("instance.json") as file:
    data = json.load(file)
n = data["nb_operations"]
c = data["cycle_time"]
D = data["durations"]
S = data["successors"]
N = range(n)

# 2. MODELING THE PROBLEM WITH CP-OPTIMIZER

from docplex.cp.model import *
model = CpoModel()

# Decision variables: operations and station boundaries
op = [interval_var(size=D[i]) for i in N ]
sb = [interval_var(size=1,start=k*(1+c)) for k in N ]

# Objective: minimize project makespan
model.add(minimize(max(end_of(op[i]) for i in N)))

# Constraints: precedence between operations
model.add([end_before_start(op[i],op[j]) for [i,j] in S])

# Constraints: operations ans station boundaries do not overlap
model.add(no_overlap(op + sb))

# 3. SOLVING THE PROBLEM

sol = model.solve(TimeLimit=30)
nstations = (sol.get_objective_values()[0]+c) // (1+c)
```

Yes, that's all!

The model creates one interval variable *op[i]* per operation. There are at most n stations. The time boundaries of stations is modeled by fixed interval variables of length 1 (the value 1 could be any positive integer, it represents the time taken to move from one station to the next one, the value of this duration has no impact on the problem). Precedence relations are posted using *end_before_start* constraints between operations and a global *no_overlap* constraint is posted to ensure that operations do not overlap each other and do not overlap station boundaries. Finally, the objective function is to minimize the makespan (end time of the last operation) as the number of stations is directly related with the makespan: *nstations = ceil(makespan/(1+c))*.

The instance file "instance.json" of the introductory example above reads like this:

```
{
 "nb_operations" : 20,
 "cycle_time"    : 1000,
 "durations"     : [ 108, 90, 67, 286, 107, 65, 468, 444, 126, 50, 143, 450, 275, 125, 143, 215, 85, 45, 204, 488 ],
 "successors"    : [ [0,5], [1,6], [2,4], [3,7], [3,8], [4,10], [5,9], [6,11], [7,13], [8,12], [9,14],
                     [10,15], [12,16], [14,17], [15,18], [16,19] ]
}
```

Here is the same model formulated in OPL:

```
using CP;

tuple P { int i; int j; }

// 1. READING DATA

int   n    = ...;
range N    = 0..n-1;
int   c    = ...;
int   D[N] = ...;
{ P } S    = ...;

// 2. MODELING THE PROBLEM WITH CP-OPTIMIZER

dvar interval op[i in N] size D[i];
dvar interval sb[k in N] in k*(1+c)..k*(1+c)+1 size 1;

execute { cp.param.TimeLimit = 30; }

dexpr int makespan = max(i in N) endOf(op[i]);
minimize makespan;
subject to {
  forall(p in S)
    endBeforeStart(op[p.i],op[p.j]);
  noOverlap(append(op,sb));
}
```


Note that there are several ways to formulate the fact the operations do not overlap the stations boundaries. We will use a slightly different one with similar efficiency in an extension of the problem in the end of the article.

Results

We run CP Optimizer on instances of the largest instance sets 'large' and 'very large' for respectively 100 and 1000 operations, from the **benchmark data sets** of [Otto & al, 2013] considering that the small problems with 50 operations or less are not really representative of realistic problem size.

Experiments were run on a MacBook Pro, Processor 2.5 GHz Quad-Core Intel Core i7, 16 GB RAM Memory.

Results on problems with 100 operations

We run all the 525 instances with 100 operations of [Otto & al, 2013] with the above CP Optimizer formulation with a time limit of ***30 sec.*** and performed a similar comparison as the one done with LocalSolver in a recent **post**. The average gap to the best known solutions is -0.67%, it is compared to the gaps of LocalSolver on Table 1.

	30 sec	5 min	1 hour
LocalSolver	6.4%	3.7%	2.8%
CP Optimizer	-0.67%		

TABLE 1 – Average gap to best known solution of [Otto & al, 2013] on problems with 100 operations

On this benchmark with 100 operations, LocalSolver reported improving **37** best known solutions over the results of [Otto & al, 2013]. Using the same reference, the CP Optimizer model (with a time limit of *30 sec.*) improves **131** solutions. If we take into account the 37 improvements by LocalSolver, CP Optimizer still enhances **115** solutions. The new solutions are reported on Table 2, the ones enhancing LocalSolver solutions are in yellow. Column 'PB' is the problem instance number. Column 'S+L' is the number of stations in the best solution from [Otto & al, 2013] taking into account LocalSolver improvements. Column 'CPO' is the solution found by CP Optimizer within the *30 sec.* time limit.

PB	S+L	CPO		PB	S+L	CPO		PB	S+L	CPO
51	51	50		52	54	52		53	53	52
54	52	51		55	54	53		56	53	52
57	56	54		58	58	57		60	55	54
62	53	51		66	52	51		67	56	55
69	55	53		70	56	53		72	56	54
73	58	56		74	52	51		93	28	27
126	54	51		127	54	53		128	59	57
129	56	55		130	56	55		131	54	53
132	59	58		134	58	55		135	58	55
136	54	53		137	55	54		138	57	56
140	57	55		142	57	55		144	49	48
145	58	57		147	62	59		148	54	53
149	57	55		150	58	57		201	54	53
202	62	61		203	53	52		205	58	57
206	54	51		207	52	51		208	58	56
210	53	52		211	53	51		212	53	52
213	54	53		214	55	54		217	53	52
218	54	53		219	53	52		222	55	53
223	53	51		225	55	53		276	62	61
279	55	54		280	57	55		281	64	62
283	57	55		284	56	55		285	56	55
286	59	57		287	56	55		289	63	62
290	56	54		293	54	53		295	58	57
296	58	55		299	56	55		300	56	55
351	60	59		353	54	51		354	54	52
356	61	59		358	54	52		359	54	53
360	57	55		364	53	52		365	54	53
367	57	55		369	52	51		371	54	53
373	52	51		374	53	51		426	63	61
427	57	56		428	57	54		431	55	54
432	57	56		433	54	52		435	58	56
436	53	52		437	56	53		438	57	55
439	58	55		440	55	54		441	54	53
443	58	56		444	54	53		445	57	56
446	57	56		447	55	54		448	57	56
501	63	62		503	62	60		509	59	58
512	61	60		513	63	62		515	63	61
518	58	57		519	63	61		520	62	60
523	56	55								

TABLE 2 – Better solutions found by CP Optimizer
on problems with 100 operations

Results on problems with 1000 operations

Problems with 1000 operations are probably more representative of the size of actual industrial problems. On these problems, we ran CP Optimizer on all the 525 instances with a time limit of **2 min**. The average gap to best known solutions is **-0.05%**. A total of **180** solutions were improved. These new solutions are reported on Table 3. Column 'PB' is the problem instance number. Column 'S' is the best solution from [Otto & al, 2013]. Column 'CPO' is the solution found by CP Optimizer within the *2 min.* time limit.

Note that LocalSolver did not report any result on this benchmark with 1000 operations.

PB	S	CPO	PB	S	CPO	PB	S	CPO
30	548	547	33	531	530	35	529	527
36	527	522	38	546	542	42	524	517
43	533	524	45	512	511	47	528	526
57	226	225	63	229	228	70	230	229
71	232	231	73	223	222	101	550	545
102	551	548	103	556	555	104	555	539
105	541	540	106	557	546	107	539	533
108	553	540	109	556	543	110	555	547
111	541	537	112	550	543	113	549	535
114	553	540	115	550	534	116	542	534
117	550	544	118	570	556	119	538	525
120	561	543	121	536	532	122	528	523
123	548	545	124	541	536	125	546	535
126	231	230	128	225	224	132	217	216
135	228	227	138	224	223	139	227	226
141	219	217	142	223	222	143	216	215
144	220	219	145	223	222	149	240	239
150	225	224	178	547	545	179	546	545
181	553	547	187	556	551	191	541	538
199	525	520	200	537	529	203	232	231
252	558	556	253	557	554	254	555	552
256	546	542	257	558	556	258	549	545
259	544	542	260	542	538	262	538	532
264	551	544	265	568	567	267	566	559
269	553	547	271	545	536	275	563	560
281	223	222	294	234	233	296	211	210
300	232	231	327	536	534	328	532	525
331	531	527	332	524	518	336	528	518
338	540	534	339	542	541	341	539	537
343	540	539	344	531	530	345	538	537
346	528	525	347	531	530	348	553	552
350	526	518	366	230	229	375	229	228
401	547	541	402	565	552	403	557	545
404	550	539	405	557	551	406	542	534
407	548	544	408	567	554	409	559	547
412	547	545	413	547	546	414	557	546
415	553	546	416	561	554	417	585	579
418	548	545	419	574	568	420	551	548
421	546	541	422	546	540	423	557	551
424	545	534	425	563	559	426	227	226
428	227	226	430	223	222	433	233	232
435	230	229	436	230	229	437	225	224
439	228	227	441	225	224	442	234	233
443	220	219	444	225	224	446	231	230
447	225	224	449	236	235	476	585	573
477	597	581	478	607	598	479	588	578
480	582	566	481	594	579	482	604	592
483	588	566	484	598	588	485	594	579
486	582	570	487	597	583	488	581	567
489	578	563	490	594	573	491	585	570
492	606	582	493	571	558	494	583	567
495	606	590	496	569	556	497	580	563
498	593	582	499	579	563	500	591	570
501	234	232	502	230	228	503	231	229
504	234	233	506	230	228	507	227	226
508	225	223	509	232	230	510	234	232
511	237	235	512	227	224	513	226	224
515	227	225	516	236	235	517	228	226
518	226	224	521	236	234	522	221	220
523	226	225	524	233	231	525	227	226

TABLE 3 – Better solutions found by CP Optimizer on problems with 1000 operations

Flexibility of the CP Optimizer formulation

Toward more complex stations

In fact, the CP Optimizer formulation we have seen does more than just allocating operations to stations (which turns out to be sufficient in the simple version of the problem): it computes actual start and end times of each operation within each station. In many industrial line balancing problems, this is necessary because the resources available at the stations are more complex that just a single operator: there may be several available operators, some equipments or renewable resources can be used by the operations, there may be constraints due to the limited space at the station, incompatibilities between some pairs of operations to run in parallel, etc. Stated otherwise, finding the right timing of operations within a given station may be a complex scheduling problem in itself (somehow similar to the Resource Constrained Project Scheduling Problem, see **my article** on a formulation of this problem with CP Optimizer).

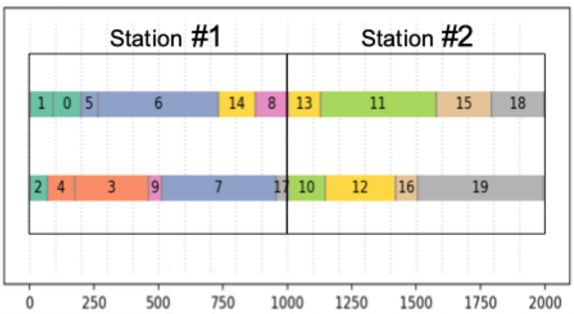
Because the CP Optimizer model also consider the scheduling of operations within the work stations, it can be very easily extended to more realistic versions of the problem.

Let's see a simple extension where we suppose that there is more than a single operator at a given station. Let *W* denote the number of operators at each station. The CP Optimizer model can easily be adapted to use a *cumul function* instead of a *no-overlap* constraint. The extended formulation is given below in Python.

```
# 2. MODELING THE PROBLEM WITH CP-OPTIMIZER

from docplex.cp.model import *
model = CpoModel()

# Decision variables: operations and station boundaries
op = [interval_var(size=D[i]) for i in N ]

# Decision expression: number of operations over time
load = sum([pulse(op[i],1) for i in N])

# Objective: minimize project makespan
model.add(minimize(max(end_of(op[i]) for i in N)))

# Constraints: precedence between operations
model.add([end_before_start(op[i],op[j]) for [i,j] in S])

# Constraints: time values of station boundaries
model.add([always_in(load,k*(1+c),k*(1+c)+1,0,0) for k in N])

# Constraints: maximal number of workers
model.add(load <= W)
```

This formulation does not need the fixed interval variables for the station boundaries. Instead, it uses a *cumul function* 'load' that represents the evolution over time *t* of the number of operations currently executing at time *t*. Constraints are posted on the possible values of this function 'load': it must be lower than the number of workers *W* and it must be 0 at stations boundaries.

An optimal solution for a version of the example using two operators (*W=2*) is shown below. It only uses 2 stations.



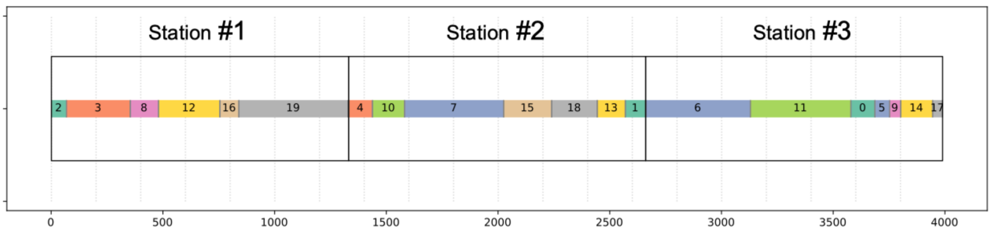
Toward other objective functions

It is also pretty easy to extend the model to handle different objective functions. For instance a classical variant (SALBP-2) is to minimize the cycle time c given a fixed number of stations m. If M denotes the range $\{0,\ldots, m\}$, the model can be adapted as follows:

```
# 2. MODELING THE PROBLEM WITH CP-OPTIMIZER

# Decision variables: operations and station boundaries
op = [interval_var(size=D[i]) for i in N ]
sb = [interval_var(size=1) for k in M ]
c  = integer_var(max([D[i] for i in N]), sum([D[i] for i in N]))

# Objective: minimize cycle time
model.add(minimize(c))

# Constraints: precedence between operations
model.add([end_before_start(op[i],op[j]) for [i,j] in S])

# Constraints: operations finish before end time of last station
model.add([end_before_start(op[i],sb[m]) for i in N])

# Constraints: cycle time of each station
model.add([start_of(sb[k]) == k*(1+c) for k in M])

# Constraints: operations and station boundaries do not overlap
model.add(no_overlap(op + sb))
```

The main different with the original model is that the interval variables representing the station boundaries 'sb' are now not fixed but constrained to execute at some multiple of the cycle time. And the objective is to minimize cycle time 'c'.

An optimal solution for an SALBP-2 version of the example using three stations ($m=3$) is shown below. The optimal cycle time is $c=1329$.



References

[Otto & al, 2013] Otto, A.; Otto, C.; Scholl, A. (2013): Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. European Journal of Operational Research 228/1, 33-45.

Report this

Published by

Philippe Laborie
Software Developer: Operations Research, Artificial Intelligence, Optimization, Planning & Scheduling
Published • 3mo

3 articles

#optimization #operationsresearch #datascience #scheduling #cpoptimizer #localsolver