Wall-Clock Time for the SSLP

David L. Woodruff
Graduate School of Management
University of California Davis
Davis CA 95616, USA
Email: dlwoodruff@ucdavis.edu

November 30, 2016

1 Preliminaries

The stochastic server location problem (SSLP) [3] continues to serve as a classic vehicle for testing and improving algorithms for optimization under uncertainty. In this note we examine the issue of finding solutions with reasonably tight bounds with as little elapsed time as possible. Of course, parallel computing is needed, but apologies must be extended to readers with an interest in classic research in parallel computing. Our interest here is in the practical question of wall-clock time on a particular computer and little else.

We use Progressive Hedging (PH) [4] with bounds [1] as implemented in the PySP library for stochastic programming [6] that is distributed with the Pyomo [2] Python library for optimization `www.pyomo.org`. For these experiments, PH is terminated when the subproblems agree on first stage variables.

The SSLP instances have names that include values for m, n, and s in that order. The value of m gives the number of potential server locations, which n give the number of potential clients. Together, they determine the size of the problem instance for one scenario. The value of s gives the number of scenarios sampled for the instance.

2 Details

For the most part, we are not even interested in all components of the time, mainly just the time to execute the optimization. In most of the results we leave out the time needed for PySP to process the symbolic language used to describe the instances and report only the algorithm run times.

2.1 Initial Experiments

All experiments are conducted on a 96-core Intel Xeon workstation with 2.1GHz processors and 1TB of RAM.

Table 1 gives results obtained using commands like the following:

```
runef -i sslp_5_50_2000/scenariodata -m model \
--solve --mipgap=0.001 --solver=cplex \
--solver-options="timelimit=14400" --output-solver-log
```

These results are for the entire problem instance, with non-anticipativity constraints added, handed to CPLEX with termination criteria of an 0.001 mipgap or 14400 seconds, whichever comes first. In subsequent tables this is referred to using the label "EF" (Extensive Form).

Instance	Solve Time	Obj	Bound
sslp_5_25_50	2.23	-121.6	-121.6
sslp_5_25_100	4.21	-127.37	-127.37
sslp_5_50_50	3.22	-91 -91	
sslp_5_50_100	7.11	-323.7	-323.7
sslp_5_50_500	49.38	-320.788	-320.81144
sslp_5_50_1000	145.3	-320.139	-320.15372
sslp_5_50_1500	249.15	-323.03467	-323.04611
sslp_5_50_2000	407.72	-320.599	-320.60721
sslp_10_50_50	53.39	-369.84	-370.20983
sslp_10_50_100	128.34	-359.28	-359.63909
sslp_10_50_500	5079.15	-354.078	-354.41479
sslp_10_50_1000	14502.42	-342.611	-357.2986
sslp_10_50_2000	14588.66	-325.985	-355.052
sslp_15_45_5	7.22	-262.4	-262.65465
sslp_15_45_10	5.04	-260.5	-260.71774
sslp_15_45_15	10.67	-253.53333	-253.78128

Table 1: CPLEX version 12.6.3 with a four hour time limit applied to the EF. Times are in seconds and do not include the time for PySP to process the model and give it to CPLEX. The Obj and Bound columns give the objective value of the best solution found and the best lower bound, respectively.

Table 2 shows the winner of a "horse race" between "EF," (shown in Table 1) and three versions of PH. The first is standard PH with each scenario treated as a sub-problem and the parameter ρ simply set to 10, which never wins. The other two are PH, also with ρ set simply to 10, but with sub-problems formed by bundling scenarios in randomly selected groups and treating the resulting extensive form problem as a "super-scenario." The number of bundles was 10 (Bun10) or 50 (Bun50). For straight PH and for Bun50, the number of threads allocated to CPLEX 12.6.3 for each sub-problem was 2. For Bun10, CPLEX

was allowed to use up to 16 threads for each sub-problem. None of the "tricks"described in Watson and Woodruff [5] are used. In particular, a single value of ρ is used for all variables and single mipgap (0.001) is used for all problem and sub-problem solves done by CPLEX. Here is a typical command:

```
mpiexec -np 1 pyomo_ns : -np 1 dispatch_srvr : -np 10 phsolverserver \
: -np 1 runph -i sslp_10_50_2000/scenariodata -m model \
--scenario-mipgap=0.001 --solver=cplex \
--default-rho=10 --disable-gc --max-iterations=200 \
--user-defined-extension=pyomo.pysp.plugins.phboundextension \
--solver-manager=phpyro --create-random-bundles=10 \
--scenario-tree-seed=7734 --scenario-solver-options="threads=16"
```

The bounds extension was controlled so as to run every 10 PH iterations, in addition to the zeroth and last iteration.

Instance	Solve Time	Obj	Bound	Method
5_25_50	2.2	-121.60	-121.60	EF
5_25_100	4.2	-127.37	-127.37	EF
5_50_50	3.2	-91.00	-91.00	EF
5_50_100	7.1	-323.70	-323.70	EF
5_50_500	49.4	-320.79	-320.81	EF
5_50_1000	49.1	-320.15	-320.15	Bun10
5_50_1500	80.5	-323.04	-323.04	Bun10
5_50_2000	105.8	-320.60	-320.60	Bun10
10_50_50	17.9	-369.92	-370.32	Bun10
10_50_100	69.4	-359.33	-359.92	Bun50
10_50_500	205.0	-354.06	-354.36	Bun10
10_50_1000	244.7	-356.43	-356.62	Bun50
10_50_2000	593.9	-352.07	-352.32	Bun50
15_45_5	7.2	-262.40	-262.65	EF
15_45_10	5.0	-260.50	-260.72	EF
15_45_15	10.7	-253.53	-253.78	EF

Table 2: Initial horse race winners. Times are in seconds and do not include the time for PySP to process the model and give it to CPLEX 12.6.3 or to PH, which uses CPLEX as a subproblem solver. The Obj and Bound columns give the objective value of the best solution found and the best lower bound, respectively.

It should be noted that CPLEX often finds somewhat good solutions early in the search and then spends a lot of time improving them a little and improving the bounds. But our main interest is finding instance for further study. For our purposes, that would seem to be the 10_50 family. With sufficient bundling, all of these instances terminate after two PH iterations, namely iteration 0 and iteration 1. (Aside: some instances could be terminated after iteration zero without degraded performance, but also not as much time

	bun100-2				bun100-4				bun50-8	
Scenarios	PH sec	Total sec	Obj	Bound	PH sec	Total sec	Obj	Bound	PH sec	Total sec
500	230	298	-354.1	-355.3	83	153	-354.1	-355.3	210	268
1000	136	304	-356.4	-356.8	158	303	-356.4	-356.8	198	300
2000	405	813	-352.1	-352.3	303	708	-352.1	-352.3	563	792

Table 3: Results for the larger sslp_10_50_s instances. The column groups refer to the number of bundles and the number of threads per subproblem (i.e. bundle). For all but the last column group the subproblem mipgap sent to CPLEX12.6.3 is 0.001, but for the last column group it is 0.01. The column "PH sec" corresponds to the column "Solve time" in the previous tables.

improvement as one might think because of warms starts of the sub-problem solvers after iteration 0). All of the winning horses for this family, except for the 100 scenario case, terminate after two iterations. It should be noted that Bun10 and Bun50 have nearly the same time for 10_50_100, but it is the exception that proves the rule. Bun50 is sightly faster for this instance even though it takes 14 PH iterations because the PH iterations are so much faster than than the two iterations required when the bundle are bigger.

So the lesson is not surprising: bundles should be big enough to ensure rapid convergence of PH, but no so large as to slow down the sub-problem solves too much.

2.2 Work on the 10 50 family a little more

Based on the limited preliminary experiments, it seems that between 5 and 10 scenarios per bundle might be enough to assure rapid convergence of PH. Note that the fact that convergence occurs in two iterations for Bun10 with 50 scenarios does not assure that it would happen that fast with 500 scenarios because there is a greater chance that a few of the scenario bundles are "weird." We conduct an experiment with 100 bundles, even though that is more sub-problems than CPUs by a little bit (and there are a few other users of this computer). Also, we allow each sub-problem four threads, so we could use up to 400 threads on a machine with fewer than 100 cores. However, it is our experience that some of the bundles will solve very quickly. Call this algorithm Bun100-4. We also try 50 bundles, with each allowed up to 8 threads. Finally, we try 100 bundles, four threads each and a sub-problem mipgap of 0.01, which is an order of magnitude looser than for all other experiments. This final configuration is labeled "bun100-4.01."

The results are shown in Table 3 Speedup is limited, in part because of communication bottlenecks. E.g., in these experiments the dispatch server is sometimes the bottleneck.

The difference between PH seconds and Total seconds includes time to process the modeling language, but also time to start up parallel sub-problem solvers and they have sleeps at startup that are fairly long (multiples of 4 seconds) and intentionally somewhat random. The need for sleep depends on communication handshakes and can vary from one run to the next. The main thing to notice is that a huge speedup is possible at the expense

of small quality degradation if a looser mipgap is used by CPLEX on the sub-problems. It should be noted that all runs terminated after 2 PH iterations except the bun100-x configurations for 500 scenarios. For 500 scenarios, with 100 subproblems, 6 iterations were needed for PH convergence with a mipgap of 0.001 and 4 with a mipgap of 0.01; the only explanation for needing fewer iterations with a looser mipgap that I can think of is "randomness" or maybe something to do with heuristics in CPLEX finding the solution versus branch-and-bound.

3 Conclusions

We have described some benchmark results for the venerable sslp instance on a shared memory computer with 96 CPUs. Speedups are possible, of course. The main speedup to PH for these instances would to terminate after iteration zero if the bounds have converged (even if PH has not). Another possibility would be to adapt the mipgap based on PH bounds as well as termination. Finally, one could dedicate processors to bound calculations.

Bibliography

[1] D. Gade, G. Hackebeil, S.M. Ryan, J.P. Watson, R.J.B. Wets, and D.L. Woodruff. Obtaining lower bounds from the progressive hedging algorithm for stochastic mixed-integer programs. *Mathematical Programming, Series B*, 157(1):47–67, 2016.

[2] W.E. Hart, J.P. Watson, and D.L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3), 2011.

[3] Lewis Ntaimo and Suvrajeet Sen. The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, 32:385–400, 2005.

[4] R Tyrrell Rockafellar and Roger J-B Wets. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, 16(1):119–147, 2004.

[5] Jean-Paul Watson and David L Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370, 2011.

[6] J.P. Watson, D.L. Woodruff, and W.E. Hart. PySP: Modeling and solving stochastic programs in Python. *Mathematical Programming Computation*, 4(2), 2012.