Submitted to *Operations Research*
manuscript (Please, provide the mansucript number!)

Fast sample average approximation for minimizing Conditional-Value-at-Risk

Daniel Espinoza
Department of Industrial Engineering, Universidad de Chile, daespino@dii.uchile.cl

Eduardo Moreno
Faculty of Engineering and Science, Universidad Adolfo Ibañez, eduardo.moreno@uai.cl

Recent years have seen growing interest in coherent risk measures, such as Conditional Value-at-Risk (CVaR). One reason is that they fulfill many desirable properties for risk measures. A second reason is the recent algorithmic results for solving stochastic optimization problems, mainly through the sample average approximation (SAA) approach, that allows to solve stochastic optimization problems where the underlying distribution is more general than the classic closed-formula solutions known today for special cases. There is however one disadvantage of the SAA technique, even when used with convex risk measures such as CVaR: there are some cases where the number of samples required to get good approximated solutions tends to be very large. In this paper we propose an automatic aggregation scheme to exactly solve linear programs with CVaR objective function using sample average approximations with a very large number of scenarios. Extensive computational experiments are performed on Netlib LP instances. Our results show that this aggregation scheme is in average between 3 (1,000 scenarios) and 152 (1,000,000 scenarios) times faster than the standard LP-equivalent formulation. Moreover, the proposed aggregation scheme reduce the memory requirements to solve the problem, allowing to exactly solve minimization CVaR problems with tens of millions of scenarios.

Key words: Conditional Value at Risk, Aggregation techniques, Approximation methods, Sample Average Approximation

1. Introduction

Conditional Value-at-Risk (also known as CVaR, or Expected Shortfall) is a risk measure that has gained major relevance in recent years. This relevance is explained by different reasons. CVaR is a "coherent" risk measure (Acerbi and Tasche 2002), which is a desired property that other risk measures (such as Value-at-Risk) do not have. Moreover, CVaR can be formulated as a convex function, so it is (in principle) more suitable to be used in optimization problems either as an objective function or as a constraint. On the other hand, the Sample Average Approximation method (Kleywegt et al. 2002, Linderoth et al. 2006) allows to pose any linear programing problem with CVaR objective and/or constraints, under any distribution, as a linear programming problem. This explain why computational portfolio optimization problems minimizing CVaR has been extensively studied in recent years (Lim et al. 2011, Ogryczak and Śliwiński 2011).

From the computational point of view, approximations of CVaR may require many scenarios to obtain good estimations of the real optimum, requiring to solve a large scale LP optimization problem. In Beliakov and Bagirov (2006), authors affirm that LP formulation of CVaR "does not scale well" for a portfolio optimization problem, and they propose a non-smooth optimization method of the discrete gradient. Also, in Künzi-Bay and Mayer (2006), different linear and non-linear programming formulations of CVaR are studied, and authors also propose a new method based on the L-shaped method.

In this paper, we propose an algorithm to compute the "traditional" sample average approximation of CVaR, using an automatically generated aggregation of scenarios. Instead of solving the full problem, we solve a smaller problem using aggregated scenarios, and iteratively we disaggregate them to obtain a better approximation. This algorithm is based on an algorithm for scheduling of open pit mines, that appears in Bienstock and Zuckerberg (2010).

In Section 2 we formally define CVaR and the LP formulation based on SAA. In Section 3 we present our algorithm, and we prove its correctness in Section 4. Finally, in Section 5 we show computational results comparing our algorithm with the full formulation.

2. Problem description

Given $x \in \mathbb{R}^n$, a linear *loss*-function $z(x) = \hat{c}x$, where $\hat{c}$ is a random parameter, and a probability distribution function $F_x(\lambda) = \mathbb{P}(\hat{c}x \leq \lambda)$, we define the Value-at-Risk (VaR) at level ε as

$$\mathrm{VaR}_\varepsilon(x) = \min\{\lambda | F_x(\lambda) \geq \varepsilon\}$$

Analogously, the conditional Value-at-Risk (CVaR) at level ε is defined as

$$\mathrm{CVaR}_\varepsilon(x) = \mathbb{E}(\hat{c}x | \hat{c}x \geq \mathrm{VaR}_\varepsilon(x)) = \min_t \left[t + \frac{1}{\varepsilon}\mathbb{E}((\hat{c}x - t)^+) \right]$$

Note that exact computation of CVaR is not possible except for some particulars distribution functions F_x. In this paper, we are interested in a general linear programming problem minimizing CVaR. Let us consider the following problem:

$$(P) \min \mathrm{CVaR}_\varepsilon(\hat{c}x) \tag{1a}$$
$$s.t.\ Ax = b \tag{1b}$$
$$x \geq 0, \tag{1c}$$

where $\hat{c}$ is a random variable and $\varepsilon \in]0,1]$.

In Rockafellar and Uryasev (2000), a linear programming approach is proposed, based on sample average approximation (SAA) of the expectation function. Given a large enough sample $\{c^i\}_{i=1}^N$ of $\hat{c}$, each of them with probability p_i, (1) can be well approximated by the following problem:

$$(P_N) \min t + \frac{1}{\varepsilon}\sum_{i=1}^N p_i \eta_i \tag{2a}$$
$$s.t.\ Ax = b \tag{2b}$$
$$c^i x + t + \eta_i \geq 0 \quad \forall i \in \{1, \ldots, N\} \tag{2c}$$
$$x, \eta \geq 0 \tag{2d}$$

However, although optimal objective values $z_{P_N} \rightarrow_{N\rightarrow\infty} z_P$, for problems with many random parameters, the number of scenarios required to obtain good approximations tends to be too large.

3. Our algorithm

Consider $\mathcal{N} = \{P^j\}_{j=1}^k$ a partition of $\{1, \ldots, N\}$, and the problem

$$(\underline{P}_{\mathcal{N}}) \min t + \frac{1}{\varepsilon}\sum_{j=1}^k \hat{p}_j \hat{\eta}_j \tag{3a}$$
$$s.t.\ Ax = b \tag{3b}$$
$$\bar{c}^j x + t + \hat{\eta}_j \geq 0 \quad \forall j \in \{1, \ldots, k\} \tag{3c}$$
$$x, \hat{\eta} \geq 0 \tag{3d}$$

where $\hat{p}_j = \sum_{i\in P^j} p_i$ and $\bar{c}^j = \frac{1}{\hat{p}_j}\sum_{i\in P^j} p_i c^i$. Note that (3) is a relaxation of (2), from where $z_{\underline{P}_{\mathcal{N}}} \leq z_{P_N}$. On the other hand, given $\tilde{x}\geq 0$ satisfying (3b), we can formulate the problem

$$(\overline{P}_{\tilde{x}})\min t + \frac{1}{\varepsilon}\sum_{i=1}^{N} p_i\eta_i \tag{4a}$$

$$s.t.\, c^i\tilde{x} + t + \eta_i \geq 0 \quad \forall i\in\{1,\dots,N\} \tag{4b}$$

$$\eta \geq 0 \tag{4c}$$

Let $w_i = c^i x\ \forall i = 1\dots N$ and reorder these values such that $w_{(1)}\leq w_{(2)}\leq \dots w_{(n)}$. Let $i^* = \max\{l : \sum_{i=1\dots l} p_{(i)} \leq \varepsilon\}$. Then, it is easy to see that the optimal solution of this problem is $t = -w_{(i^*+1)}$ and $\eta_{(i)} = w_{(i^*+1)} - w_{(i)}$ for $i\leq i^*$ and $\eta_{(i)} = 0$ for $i > i^*$. Hence, (4) can be solved in time $\mathcal{O}(N\log(N))$. Moreover, for any $\tilde{x}\geq 0$ satisfying (3b), we have that

$$z_{\underline{P}_{\mathcal{N}}} \leq z_{P_N} \leq z_{\overline{P}_{\tilde{x}}} \tag{5}$$

This suggest the following scheme to solve (2). Start with $\mathcal{N} = \{\{1,\dots,N\}\}$, solve (3), get $\tilde{x}$, solve (4). If $z_{\underline{P}_{\mathcal{N}}} = z_{\overline{P}_{\tilde{x}}}$ then stop. Otherwise, refine partition $\mathcal{N}$. A detailed description of this procedure is presented in Algorithm 1.

Algorithm 1 Iterative procedure to solve P_N

```
1: Start with $\mathcal{N} = \{\{1,\dots,N\}\}$
2: loop
3:    Solve $(\underline{P}_{\mathcal{N}})$. Let $(\tilde{x},\tilde{\eta})$ be its solution, and $z_{\underline{P}_{\mathcal{N}}}$ be its objective value.
4:    Solve $(\overline{P}_{\tilde{x}})$. Let $(\eta^*)$ be its solution and let $z_{\overline{P}_{\tilde{x}}}$ be the optimal objective value.
5:    if $z_{\underline{P}_{\mathcal{N}}} = z_{\overline{P}_{\tilde{x}}}$ then
6:       return Solution $(\tilde{x},\eta^*)$
7:    else
8:       Let $N^+ = \{i : \eta^*_i > 0\}$
         Refine partition $\mathcal{N} \leftarrow \bigcup_{j=1..k}\{(P^j\setminus\{i^*+1\})\cap N^+, (P^j\setminus\{i^*+1\})\setminus N^+\}\cup\{i^*+1\}$
```

Lemma 1. *Algorithm 1 stops and returns the optimal solution of* P_N.

4. Proof of Lemma 1

Since the number of scenarios is finite, we only need to prove that the size of $\mathcal{N}$ increase in each step. Let us rewrite the dual of (2) as following

$$(DP_N)\max y^t b \tag{6a}$$

$$s.t.\, y^t A + \frac{1}{\varepsilon}\sum_{i=1}^{N} p_i\lambda_i c^i \leq 0 \tag{6b}$$

$$\sum_{i=1}^{N} p_i\lambda_i = \varepsilon \tag{6c}$$

$$0\leq \lambda_i \leq 1 \quad \forall i\in\{1\dots N\} \tag{6d}$$

In the same way, the dual of (3) can be rewritten as following:

$$(D\underline{P}_{\mathcal{N}})\ \max y^t b \tag{7a}$$

$$s.t.\, y^t A + \frac{1}{\varepsilon}\sum_{j=1}^{k} \hat{p}_j \hat{\lambda}_j \bar{c}^j \le 0 \tag{7b}$$

$$\sum_{j=1}^{k} \hat{p}_j \hat{\lambda}_j = \varepsilon \tag{7c}$$

$$0 \le \hat{\lambda}_j \le 1 \quad \forall j \in \{1 \dots K\} \tag{7d}$$

Note that this problem, is equivalent to (DP_N) plus constraints $\lambda_i = \lambda_j$ for all i, j in a same element of partition $\mathcal{N}$. Denote $L(D\underline{P}_{\mathcal{N}}, \mu)$ the Lagrangian relaxation of $D\underline{P}_{\mathcal{N}}$ obtained by penalizing constraints (7b) by μ. That is,

$$L(D\underline{P}_{\mathcal{N}}, \mu):\ \max\left\{ y^t b + (-y^t A - \frac{1}{\varepsilon}\sum_{j=1}^{k} \hat{p}_j \hat{\lambda}_j \bar{c}^j)\mu : \sum_{j=1}^{k} \hat{p}_j \hat{\lambda}_j = \varepsilon, 0 \le \hat{\lambda}_j \le 1 \forall j = 1 \dots k \right\}$$

Finally, the dual of (4) is

$$(D\overline{P}_{\tilde{x}}) : \max\left\{ -\frac{1}{\varepsilon}\sum_{i=1}^{N} p_i \lambda_i c^i \tilde{x} : \sum_{i=1}^{N} p_i \lambda_i = \varepsilon, 0 \le \lambda_i \le 1 \forall i \right\}$$

Note that this problem can be solved by sorting the values of $c^i\tilde{x}$. Specifically, an optimal solution of this problem is given by $\lambda_{(i)} = 1$ for all $i \le i^*$, $\lambda_{(i^*+1)} = \varepsilon - \sum_{i=1}^{i^*} p_{(i)}$ and $\lambda_{(i)} = 0$ for all $i > i^* + 1$.

From Equation (5), we know that

$$z_{D\underline{P}_{\mathcal{N}}} \le z_{DP_N} \le z_{D\overline{P}_{\tilde{x}}}$$

Suppose that the size of $\mathcal{N}$ doesn't increase after refining partition $\mathcal{N}$, and let λ^* be the optimal solution of $(D\overline{P}_{\tilde{x}})$. Hence, $\lambda^*_i = \lambda^*_{i'}$ for all $i, i' \in P^j\ \forall j = 1 \dots k$. Therefore, we can construct $\hat{\lambda}^*_j = \lambda^*_i$ for some $i \in P^j$, and $\hat{\lambda}^*$ is feasible for $L(D\underline{P}_{\mathcal{N}}, \mu)$ for any $\mu \ge 0$. However, since $\tilde{x}$ is the optimal dual of constraints (7b), we know that the optimal of value of $L(D\underline{P}_{\mathcal{N}}, \tilde{x})$ is equal to $z_{D\underline{P}_{\mathcal{N}}}$, proving the result.

5. Implementation and computational results

All runs where made using a single thread with an address space limit and data limit of 4Gb and 20 hours running time limit. The machines were running Linux 2.6.18 under x86_64 architecture, with two quad-core Intel® Xeon® E5620 processors and with 48Gb of RAM. To obtain better timing measures, the machines where configured (in BIOS) with the following technologies disabled: Intel® Turbo Boost Technology, Intel® Hyper-Threading Technology, and Intel® Virtualization Technology (VT-x). These settings, allowed us to run up to eight instances in a machine without (much) interference between processes.

All our code is implemented in the C programming language. We implement Algorithm 1 by solving $(D\underline{P}_{\mathcal{N}})$ and $(D\overline{P}_{\tilde{x}})$ respectively, the main advantage of this is that the problem has, in general, no more than $m + 2n + 1$ constraints (other than bounds), and $m + 2n + |\mathcal{N}|$ variables for the reduced problem and $m + 2n + 1$ constraints and $m + 2n + N$ variables for the full problem. All linear programming problems where solved using CPLEX 12.2 dual simplex algorithm.

For generating random numbers, we use the implementation by Pierre L'Ecuyer of the algorithm published in L'Ecuyer (1999). One of the advantages of this random number generator, besides architecture Independence and a period length of $\rho \approx 2^{191}$, is that we can *jump* in the sequence of randomly generated numbers easily; this allows to generate, on the fly, any sub-sequence of random numbers, and thus permitting to work without storing all random numbers in memory, but to re-generate them as needed.

To test the performance of this algorithm, we use the Netlib (Gay 1985) LP problem collection. To generate random objective functions, we introduce uncertainty to the original objective function by multiplying independently each coefficient of the objective function with a random variable χ. We test two random variables:

- $\chi \sim U[0,1]$, i.e. χ is a uniform distribution in [0,1].
- $\chi \sim \begin{cases} \mathcal{N}(1, 0.4) & \text{with probability } 0.95 \\ \exp(10) & \text{otherwise} \end{cases}$, i.e. 95% of the time, χ follows a normal distribution, and in 5% of the time, it follows an exponential distribution. This is one of the distributions used in Lim et al. (2011).

Furthermore, we test different risk levels for CVaR, namely $\varepsilon \in \{0.01, 0.10, 0.50, 0.75\}$.

We test the algorithm in two forms:

In memory: We generate all random coefficients once at the beginning of the algorithm, storing them in an array whose length is Nn', where n' is the number of non-zero coefficients in the objective function.

Out of memory: Each time we need a random coefficient, we re-generate the proper random number, in this way we avoid storing in memory all the data, but it increases the computation time.

In the first form, we test problems with 10^2, 10^3, 10^4, 10^5 and 10^6 samples and both random distribution, and report on the instances where both, our algorithm and the full formulation, solve the same instance within the running-time limit and the memory limit described before.

In the second form, we only show the running times for our algorithm for problems with 10^5, 10^6 and 10^7 samples and the uniform random distribution.

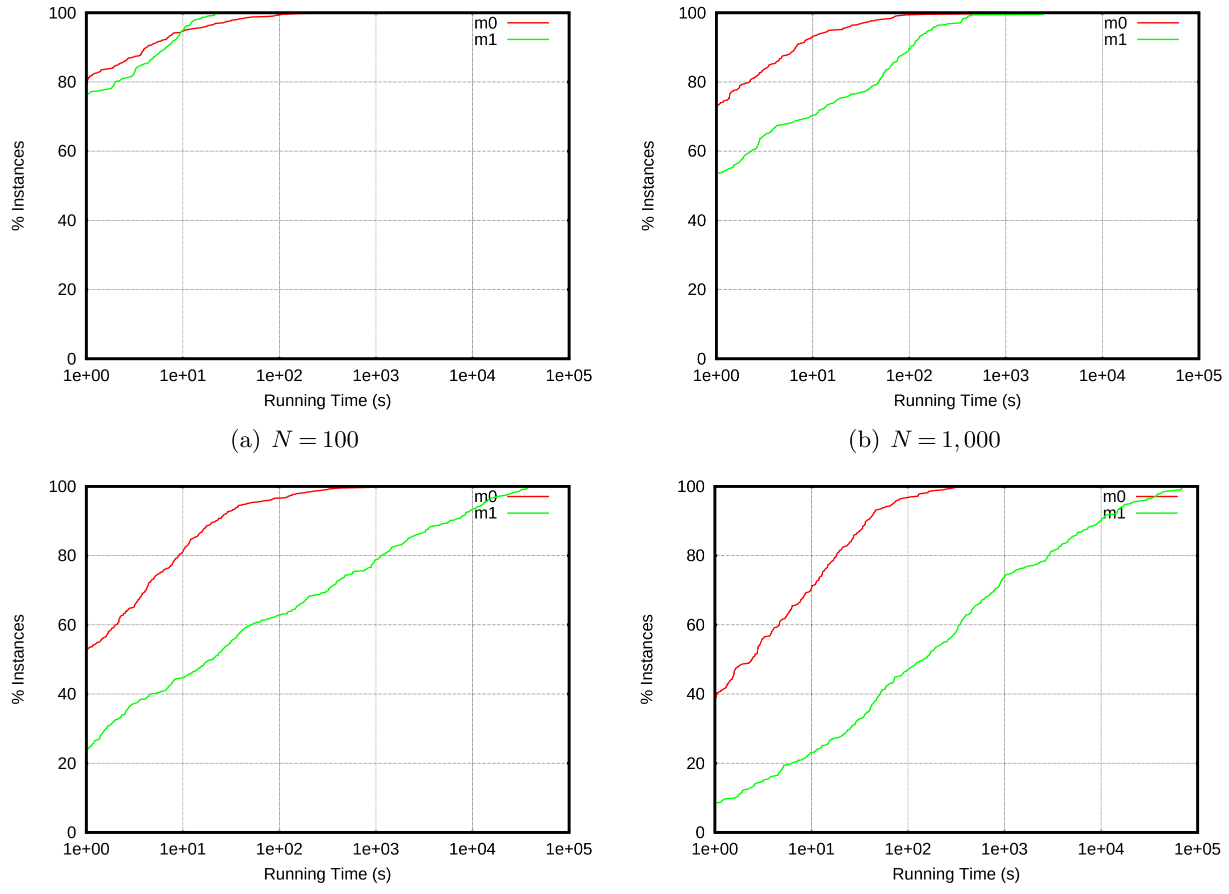
A final detail is in the termination condition of the algorithm. In theory, both upper and lower bounds should be the same; however, due to the floating-point representation, we stop our algorithm whenever

$$\frac{z_{D\underline{P}_{\mathcal{N}}} - z_{D\overline{P}_{\bar{x}}}}{\max\{1, |z_{D\underline{P}_{\mathcal{N}}}|\}} \leq 10^{-6}.$$

The value 10^{-6} was chosen because is the default reduced cost tolerance for CPLEX simplex algorithm, which should make both approaches comparable.

In Figure 1 we show performance profiles (Dolan and Mor 2002) of both methods for the different values of N. Figure 1 shows the empirical accumulated probability of the required time to solve these instances, using the full formulation (label $m0$) and our algorithm (label $m1$). These graphs include all NetLib instance that could be solved "In memory" by both methods, with the different levels of ε and probability distributions. As we can see, both methods requires a similar time to solve instance with $N = 100$ samples. However, for larger sample sizes Algorithm 1 outperforms the full formulation. In fact, our algorithm has a geometric mean time 3 times faster for $N = 10^3$, 15 times faster for $N = 10^4$, 42 times faster for $N = 10^5$ and 152 times faster for $N = 10^6$.

For the "Out of memory" form, we show the results of Algorithm 1 in Table 1. For each instance, we show the time (in seconds) required to solve the problem, the number of iterations required (column Iter) and the size of the final partition (column $|\mathcal{N}|$). We note that we are unable to compare these results with the full formulation because CPLEX can not solve many of them due to the large memory requirement. As we can see, using Algorithm 1 we are able to solve problems with a large number of scenarios not only faster than the original full formulation, but also with a smaller memory requirement. In fact, we can see that the size of the optimal partition $\mathcal{N}$ is, in the worst case, two orders of magnitude smaller than the number of scenarios N. Moreover, several



(a) $N = 100$ (b) $N = 1,000$ (c) $N = 10,000$ (d) $N = 100,000$

Figure 1 Performance Profiles of both methods on Netlib instances.

problems can be solved in a few seconds, because the size of the optimal partition contains less than 10 subsets.

6. Conclusions

We show a simple aggregation technique to exactly solve minimization of CVaR problems using sample average approximations of arbitrary probabilistic distributions, with a very large number of scenarios. Computational experiments show that this method is much more efficient in running time and, together with the use of modern random number generators, allow us to solve problems with several millions of scenarios. This is possible because the optimal solution of these problems usually requires a far smaller number of *representative* scenarios. An interesting open question is if there is a better selection of these representative scenarios allows to obtain even smaller aggregations, further improving the performance of this algorithm. Finally, we note that this aggregation technique could be also applied to solve more general two-stage stochastic programming problems, opening a new direction for future research.

Acknowledgments

Daniel Espinoza was partially funded by FONDECYT grant 1110024 and Millenium Nucleus Information and Coordination in Networks ICM/FIC P10-024F. Eduardo Moreno acknowledges the financial support of grants Anillo ACT-88 and Basal Center CMM-UCh.

References

Acerbi, Carlo, Dirk Tasche. 2002. On the coherence of expected shortfall. *Journal of Banking & Finance* **26**(7) 1487 – 1503. doi:10.1016/S0378-4266(02)00283-2.

Beliakov, Gleb, Adil Bagirov. 2006. Non-smooth optimization methods for computation of the conditional value-at-risk and portfolio optimization. *Optimization* **55**(5-6) 459–479. doi:10.1080/02331930600816353.

Bienstock, D., M. Zuckerberg. 2010. Solving lp relaxations of large-scale precedence constrained problems. *IPCO*. 1–14. doi:10.1007/978-3-642-13036-6_1.

Dolan, Elizabeth D., Jorge J. Mor. 2002. Benchmarking optimization software with performance profiles. *Mathematical Programming* **91** 201–213. doi:10.1007/s101070100263.

Gay, David M. 1985. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Bulletin* **13** 10–12. Data available at http://www.netlib.org/netlib/lp.

Kleywegt, A.J., A. Shapiro, T. Homem-de Mello. 2002. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization* **12**(2) 479–502.

Künzi-Bay, Alexandra, Jnos Mayer. 2006. Computational aspects of minimizing conditional value-at-risk. *Computational Management Science* **3**(1) 3–27.

L'Ecuyer, Pierre. 1999. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research* **47**(1) 159–164.

Lim, Andrew E.B., J. George Shanthikumar, Gah-Yi Vahn. 2011. Conditional value-at-risk in portfolio optimization: Coherent but fragile. *Operations Research Letters* **39**(3) 163 – 171. doi:10.1016/j.orl.2011.03.004.

Linderoth, J., A. Shapiro, S. Wright. 2006. The empirical behavior of sampling methods for stochastic programming. *Annals of Operations Research* **142**(1) 215–241.

Ogryczak, Włodzimierz, Tomasz Śliwiński. 2011. On solving the dual for portfolio selection by optimizing conditional value at risk. *Computational Optimization and Applications* **50** 591–595. doi:10.1007/s10589-010-9321-y.

Rockafellar, R. Tyrrell, Stanislav Uryasev. 2000. Optimization of conditional value-at-risk. *Journal of Risk* **2** 21–41.

Table 1 Results for Netlib instances for 10^5, 10^6 and 10^7 scenarios (average values for $\varepsilon = 1\%$, 10%, 50% and 75%.

	$N = 100,000$			$N = 1,000,000$			$N = 10,000,000$		
Name	time	Iter.	$\lvert\mathcal{N}\rvert$	time	Iter.	$\lvert\mathcal{N}\rvert$	time	Iter.	$\lvert\mathcal{N}\rvert$
adlittle	85	17.3	5232.8	2080	19.8	32763.5	56684	22.3	218034.0
afiro	2	2.0	2.0	22	2.0	2.0	210	2.0	2.0
agg	45	13.8	924.8	568	15.8	4272.3	6270	17.5	19766.3
bandm	57	14.8	1353.8	720	17.0	7072.8	7854	18.8	29525.3
beaconfd	21	7.8	44.5	260	8.8	79.3	2462	8.5	91.0
boeing2	54	15.0	1579.5	653	16.8	8781.8	7318	18.3	37770.0
bore3d	5	2.0	2.0	54	2.3	2.5	488	2.3	2.5
brandy	2	2.5	3.0	23	2.5	3.0	225	2.5	3.0
capri	31	14.8	2517.0	399	16.3	6927.0	4674	17.8	15675.3
e226	76	16.5	3576.3	1013	18.8	23025.3	13197	21.5	149284.5
etamacro	10	4.8	12.0	135	5.5	20.5	1040	4.8	15.5
fffff800	14	10.5	118.5	158	10.8	252.8	1499	10.3	354.0
ganges	37	12.8	815.0	471	14.8	4581.5	5380	17.3	30458.5
grow15	8	4.8	87.3	99	5.3	412.0	831	5.5	1740.5
grow22	21	9.5	135.8	218	9.5	493.8	1999	9.5	2329.3
grow7	16	9.5	146.5	164	9.3	199.5	1283	8.3	284.8
israel	38	13.8	544.5	440	14.5	2331.0	4745	16.3	9594.3
kb2	3	3.3	5.0	47	4.3	9.0	435	4.5	12.5
lotfi	0	0.0	0.0	50	5.0	14.8	535	5.5	17.8
perold	3	2.5	3.0	30	2.8	3.5	268	2.8	4.0
pilot.ja	4	2.8	3.5	32	2.8	3.5	300	2.8	3.5
pilot	45	14.3	1488.5	467	16.5	8316.5	5257	18.5	47483.0
pilot4	3	3.8	6.0	37	3.5	5.5	342	3.5	5.8
pilotnov	47	13.3	2589.8	539	14.8	12104.3	5199	16.3	19192.8
recipe	5	2.0	2.0	50	2.0	2.0	454	2.0	2.0
sc105	1	2.0	2.0	13	2.0	2.0	130	2.0	2.0
sc205	1	2.0	2.0	13	2.0	2.0	130	2.0	2.0
sc50a	1	2.0	2.0	14	2.0	2.0	130	2.0	2.0
sc50b	1	2.0	2.0	13	2.0	2.0	129	2.0	2.0
scagr7	20	6.8	29.8	231	7.5	78.8	2225	7.8	276.0
scfxm1	5	3.0	4.0	52	3.0	4.0	466	3.0	4.0
scfxm2	6	3.0	4.0	63	3.0	4.0	557	3.0	4.0
scfxm3	8	3.5	6.3	81	3.5	7.0	700	3.3	5.0
share1b	16	8.3	21.8	189	8.5	27.3	1167	5.8	22.0
share2b	16	9.0	36.3	194	9.8	67.8	1073	5.8	28.8
stair	1	2.0	2.0	14	2.0	2.0	129	2.0	2.0
standata	3	3.0	4.0	38	3.0	4.0	255	2.5	3.0
standgub	3	3.0	4.0	38	3.0	4.0	257	2.5	3.0
standmps	2	2.0	2.0	23	2.0	2.0	209	2.0	2.0
stocfor1	19	10.5	78.3	206	10.8	178.0	1305	7.5	139.5
tuff	2	2.0	2.0	19	2.0	2.0	172	2.0	2.0
vtp.base	3	3.0	4.0	36	3.0	4.3	213	2.3	2.5
woodw	5	4.5	12.0	49	4.3	12.3	287	3.0	4.5