Discrete Optimization
Assignment 2

Mehdi Nadif & Viktor Hansen

October 27, 2017

Abstract

This is the second weekly assignment for the Discrete Optimization course offered at The Department of Computer Science, Uni. Copenhagen.

Theoretical part

Let $X_a = 1$ if a is covered by C and $X_a = 0$ otherwise and let $X = \sum_{a \in V} X_a$ denote the total number of elements covered by C. We wish to bound the probability $Pr\left[X \geq n/2\right]$ using Chernoff Bound, since the bound yielded by Markov's inequality on this expression is too weak to say anything meaningful. X is a sum of Bernoulli trials with parameters $p_a \geq p \geq 1 - \frac{1}{e}$. Using this bound for p_a we can apply applying Chernoff bound for sum of Bernoulli variables [1] and get:

$$\Pr[X \geq n/2] \geq 1 - e^{-\frac{n}{2p}\left(p-\frac{1}{2}\right)^2} \geq 1 - e^{-\frac{n}{2-\frac{2}{e}}\left(1-\frac{1}{e}-\frac{1}{2}\right)^2} \geq 1 - e^{-\frac{1}{2-\frac{2}{e}}\left(1-\frac{1}{e}-\frac{1}{2}\right)^2} \approx 0.014$$

Now, consider a rounded set-cover C produced by the LP program. We wish to bound $\Pr[\text{cost}(C) \leq c \cdot OPT] = 1 - \Pr[\text{cost}(C) > c \cdot OPT]$. From Markov's inequality we get

$$\Pr[\text{cost}(C) > c \cdot OPT] \leq \Pr[\text{cost}(C) \geq c \cdot OPT_f] = \Pr[\text{cost}(C) \geq c \cdot \text{E}[\text{cost}(C)]] \leq 1/c$$

So $\Pr[\text{cost}(C) \leq c \cdot OPT] \geq 1 - 1/c$. Finally by applying union bound we get the following lower bound

$$\begin{aligned}
\Pr[X \geq n/2 \ \wedge \ c \cdot OPT \geq \text{cost}(C)] &= 1 - \Pr[X < n/2 \ \vee \ c \cdot OPT < \text{cost}(C)] \\
&\geq 1 - (1 - 0.014) - 1/c \\
&= 0.014 - 1/c \\
&= \Omega(1)
\end{aligned}$$

Which is positive for $c \gtrsim 0.014^{-1}$. For the upper bound we have $\Pr[X \geq n/2 \ \wedge \ c \cdot OPT \geq \text{cost}(C)] \leq \min\left\{0.014, 1/c\right\} = O(1)$ showing that the event occurs with probability $\Theta(1)$.

[1] `https://en.wikipedia.org/wiki/Chernoff_bound#Example`

Implementation part

All the experiments were run on a Lenovo Y500, i7 2.4 GHZ Quad core processor, with 8 GB RAM and a GeForce GT 650M. All times were computed with Java built-in System.currentTimeMillis().

2.1 CPLEX

We used the `ModelSparse1.mod` file provided for the implementation. It provides an OPL-model for CPLEX to solve the ILP for set cover with costs for each set. The results of the ILP using CPLEX can be reproduced by running `ass2op1.java`, in which we used the ilog.opl library in java to interpret the model. Then, CPLEX is run with the model, to finally get the objective values found in Table 1.

After 20 minutes of running on the file instance `scpnrg5` without results, we gave up on retrieving them. Therefore, it is marked as N/A. All other instances were solved, with timing beginning when CPLEX was set to solve and ended immediately thereafter.

2.2 Rounding

The ILP relaxation of the model used in 2.1 was created with `ModelSparse1relax.mod`, where the only change applied was to set the decision variables to float+. Since the relaxation will never have $x_s > 1$, as all costs are positive, this corresponds to relaxing so that $x_s \in [0, 1]$. All results of the relaxations can be reproduced by running `ass2op2.java`, where the approaches to each rounding was as follows:

Simple Rounding: To minimize f, we started with $f = 1$, and saw whether the rounding gave a feasible solution. If not, we increment f by 1, and repeat until we get a feasible solution. This guarantees that f was the optimal integer value. The algorithm could have been improved by finding the optimal double value instead, which will give better solutions, but it seemed against the idea of simple rounding to do so.

Randomized Rounding: We set $c = 0.1$, so that the we get a union of $0.1 \log(n)$ randomly generated sets. This parameter was used, as it proved to require few repeats, and still giving low objective values. The algorithm was repeated until a feasible solution was found.

The results of the these algorithms can be seen in Table 1, where the simple rounding both has a minimum f for feasibility and its objective values. All times were measured from the beginning of rounding until a feasible solution was found. The time it took to solve the LP relaxation is not included to give a better comparison between the two rounding algorithms running time.

From Table 1, we may note that the random rounding algorithm (RR) in almost all cases has a better minimal objective value than that of simple rounding. This is because we allowed it to run 10 times, and it will therefore at some point reach a better solution than the simple rounding. Looking at averages, we see that simple rounding provides an on average better solution than RR, but at the cost of some computation time. In some instances, the randomized rounding finds a feasible solution 5 times faster than simple rounding (on average), in which case one might find a good objective value by repeating the RR algorithm a few number of times without the cost being higher than Simple Rounding.

2.3 Third Method

The implemented metaheuristic uses a hill-climbing (valley descent) approach to determine a local minimum. The approach initialises a valid set cover and greedily removes the most costly set while maintaining feasibility until no more sets can be removed. Two approaches of producing initial feasible solutions were tested:

1. In which one set covering each vertex was picked uniformly at random.
2. In which all sets were picked initially.

1) was made with the idea of adversarial situations in mind, in which input instances might foil the greedy approach used by the algorithm, however approach 2) produced much better results and was chosen instead. Our implementation thus lacks the ability to foil adversaries and 'escape' local minima, which could be remedied by the introduction of random choices. Enlarging the neighborhood to occasionally include infeasible solutions would also seem worthwhile, however both of these strategies call for more sophisticated heuristics, e.g. simulated annealing. The results of the heuristic can be seen in Table 1 - we were surprised how well the heuristic managed to find good covers despite its simplicity. In fact, it completely dominated both rounding methods both in terms of time and objective values, and would seem to be a better option to bound the optimal value than either of the roundings. The results for the third method can be reproduced by running `ass2op3.java`.

	CPLEX		Simple Rounding		Rand. Rounding		Heuristic	
	Value	Time	(f, Value)	Time	(avg/min) value	avg time	Value	Time
scpa3	232	484	(3, 313)	275	(397.6/353)	49	256	50
scpc3	243	1399	(4, 379)	414	(462.9/363)	101.1	273	66
scpnrf1	14	19271	(7, 25)	4219	(49.3/24)	3179	20	176
scpnrg5	N/A	N/A	(5, 321)	1320	(391.7/300)	482	196	97

Table 1: Reported costs and running time (ms) for the set cover implementations. The f in the simple rounding column denotes the choice of f resulting in the feasible solution. The randomized rounding results denote the means over 10 independent runs of the algorithm with $C = 0.1$.