The Las Vegas Method of Parallelization

Bogdan Zavalnij
Institute of Mathematics and Informatics
University of Pecs
Ifjusag utja 6, H-7634 Pecs, Hungary
bogdan@ttk.pte.hu

ABSTRACT: *While the methods of parallelizing Monte Carlo algorithms in engineering modeling very popular, these methods are of little use in discrete optimization problems. We propose that the variance of the Monte Carlo method, the Las Vegas method can be used for these problems. We would like to outline the basic concept and present the algorithm working on a specific problem of finding the maximum clique.*

Keywords: Parallelization, Montecarlo algorithms, Discrete optimization

Received: 2 May 2018, Revised 9 June 2018, Accepted 30 June 2019

1. Introduction

The Monte Carlo methods have been powerful tools in scientific and engineering modeling for the last half century [1]. Their usage become even more intense in the era of computers. The easiness of parallelization made these methods useful in supercomputer environments as well. But apart from the original idea the more recent versions of these methods, in which category the Las Vegas method is falling [2], found little usage in algorithms, and even less in parallel programs. The few exceptions are the primality tests and the quicksort algorithm, although some research was made earlier in this field [7][8].

The problem we concentrate on is the maximum clique problem [4], although the concept described in this paper applies to other problems in the field of discrete optimization as well. The maximum clique problem can be formulated in the following way. Given a finite simple graph $G=(V, E)$, where V represents the nodes and E represents the edges. We call Δ a clique of G if the set of vertices of Δ is subset of V; Δ is an induced subgraph of G; and Δ is an all connected graph, thus all its nodes connected to all the other nodes. We call Δ a maximum clique if no other clique of G is bigger than it. The maximum clique problem is to find the size of a maximum clique, and it is a well known NP-hard problem. A simplified variation of this problem is the k-clique problem, which is a problem of the NP-complete class. The question is, that if given a graph G, and a positive integer k, is there a clique of size k in the graph. To answer the question we either must present a k-clique of the graph, or either prove that there is none in the graph.

2. The Family of Monte Carlo Methods

The family of the Monte Carlo methods can be distinguished by the nature of their error. We can speak about two sided, one sided and zero sided error methods. In the case of two sided errors we approximate the solution step by step. In the analysis of the method we can measure the distance of the approximation from the real solution, and find that we can get closer and closer in each step. This method is mostly useful in engineering modeling of problems with real number solutions. This two sided method can be programmed in parallel environment with ease, as the steps usually independent.

In the case of one sided error, which takes place mostly in decision problems, in each step we either get a final solution, or get no answer. This is the idea behind many primality tests, where we can find the composite numbers, but get uncertain answer for primes. The algorithms make a few dozen steps, and the uncertainty of being wrong decreases to minimum. These algorithms usually are very fast and need no parallelization.

The last method, which is called the Las Vegas method, is the case of zero sided error. The famous quicksort algorithm falls into this category. With these algorithms we always get the right answer – as the quicksort sorts the sequence in the end –, but the running time of the algorithm can be described by a probability variable. In other words sometimes the algorithm is very fast, and sometimes it can be very slow. (Luckily the later case is very-very rare in the case of the quicksort.)

Formally, we call an algorithm a Las Vegas algorithm if for a given problem instance the algorithm terminates returning a solution, and this solution is guaranteed to be a correct solution; and for any given problem instance, the run-time of the algorithm applied to this problem is a random variable. [13]

From this description it is clear, that the Monte Carlo method, on one hand, can be easily used for engineering problems as we are looking for real number answers with certain correctness. On the other hand, in the case of discrete and combinatorial optimization problems we usually need exact answers, so the Las Vegas method can prove itself of more use.

3. Parallelization with the Aim of the Las Vegas Method

The variance in the running time of a Las Vegas algorithm led Truchet, Richoux and Codognet to implement an interesting way of parallelization the algorithms for some NP-complete discrete optimization problems [13]. The authors note that the algorithm implementation for those problems heavily depends on the “starting point” of the algorithm, as it starts from a random incorrect solution and constantly changes it to find a real solution. Depending on the starting incorrect solution the convergence of the algorithm may be very fast or slow as well. The idea behind the Las Vegas parallel algorithm was to start several instances of the sequential algorithm from different starting points and let them run independently. The first instance which finds the solution shuts down all the other instances and the parallel algorithm terminates. As the running time of the different instances vary, some will terminate faster, thus ending the procedure in shorter time. The article describes the connection of the variance of the running times and the possible speed-up when using k instances and found that for some problems a linear speedup could be achieved.

This approach can be useful in several ways. For example one can use different solvers for a given problem, and/or use different preconditioning techniques. Starting these solvers concurrently will lead the most suitable one to finish in the shortest time, thus leading us to a fast solution. (Note, that for different problems different versions of the solvers may be the fastest.)

While the previous approach is simple and extremely elegant as well, it lacks something. First, the different instances cannot help each other in finding the solution. Second, each of the instances trying to solve the whole problem and no division into subproblems appears in this proposal.

I propose a different approach, which includes these notions. If we divide an NP-hard problem into parts, then the arising subproblems falls into the same category as described: these are also NP-hard problems, and have great variance in solution time. But we have a problem of constructing the sequence of the divided subproblems. As a solution of one subproblem can be helpful in the solution for the other the sequence of these subproblems have great importance: we would like to solve the easier first to help the more complex ones later. Here we can use some heuristics, but more often we proceed in the order the subproblems are already given, which leads to an inefficient algorithm.

Instead we can use the proposed Las Vegas technique starting the instances of the solver for the arisen subproblems parallely. Thus we may overcome the question of the sequence construction. As we seen, some problems will run much faster and terminate with the desired answer. These answers the can be feed to the other instances and help them to solve their subproblem faster. This way each instance solving a partial problem instead of the whole, and can help other instances to solve their subproblems faster. This method resembles the BlackBoard technique known well in the field of Artificial Intelligence.

This approach can be used to parallelize several different discrete optimization algorithms. Namely, we can use it in any Branch-

and-Bound technique instead of the branching rule. As it happens at a branching we have the problem of choosing the sequence of the branches. The speed of the algorithm heavily depends on this sequence, as the result in one branch may help us in an other branch – as a new, better bound for example.

4. An Application

In order to demonstrate the described method we choose a more simple algorithm than a general Branch-and-Bound. Instead we used an algorithm from Sandor Szabo [10], which answers the *k*-clique problem by dividing the original problem into thousands of subproblems. These subproblems then can be processed parallely with a sequential program. Obviously this algorithm needs proper number of subproblems in order to achieve proper speed-ups, which this algorithms achieves well. The proposal starts with a quasi coloring with *k-1* colors, and then examines each disturbing edge, whether that edge can be an edge of a *k*clique. If yes, than we found a positive solution, if no, then the edge can be deleted from the graph. After all the disturbing edges are deleted, we get a proper *k-1* coloring, which forbids the *k*-clique, thus we solved the problem. I have implemented this proposal and measured the running times for several different problems [14].

The measurements compares three version of the algorithm. In the first there is no information given from one subproblem to another to help it in the solution. The program instances run totally independently. In the second I constructed a sequence where the helping information is the consequence of this sequential ordering, thus the help given in advance. This means that we can delete the edges in the sequence of the subproblems in advance, proposing that no *k*-clique can contain them. There is no actual communication between the program instances and they also run independently. These two versions are detailed in the paper of Sandor Szabo [10]. The third version is the Las Vegas method, where the program instances starts parallely, and when one is finished, this information is given to others thus speeding up their solution time as the subproblems can be reduced with the aim of this information. In our case if the algorithm for a given subproblem reports that there is no *k*-clique that contains that edge, then we delete this particular edge from all the subproblems including those that are already running. For this purpose we obviously need a sequential clique search program where an edge can be deleted during the runtime.

5. Results

I used three sets of graph problems. The first set is consists of random graph with given probability of the edges. The second set is taken from the DIMACS challenge website [5][6]. In the third set two extremely hard problem represented, one is from coding theory [3][12], the other is from combinatorial optimization [11]. For all problem we know the clique size, so I run the algorithm to prove that there is no clique bigger by one than the known clique number. This step is important, because finding the maximum clique depends only on luck, thus shows little about the goodness of an algorithm. While for proving that there is no clique which is bigger by one as the known one needs extended search through the whole search tree, and thus provides a good comparison for different algorithms and implementations.

The tables show the name of a problem instance, the size (N), the density (%), the maximum clique size (clique) and the running time for the sequential algorithm on the same computer (seq). I also noted the number of subproblems that arise in the algorithm (parts).

The tests were run on 4+1, 16, 64 and up to 512 processor cores (one core doing the distribution and not taking part in calculation itself), and I show the running times in seconds for those core numbers. The "noopt" results are from the first version with no help between the problems, the "opt" stands for the more optimal version with help from one instance to an other by the original sequence, and the "lv" represents the Las Vegas method of parallelization. If the running time exceeded the time limit the table continues no data (*). The produced results seems to prove the idea interesting. The running times of the third algorithm in most of the cases were close to the running times of the second algorithm with help to other subproblems, which is an interesting fact by itself. But even more interesting, that for some cases it surpassed the second algorithm. These were the most difficult cases, thus this method could perhaps be useful for the solution of the most difficult problems.

6. Acknowledgements

Author would like to thank the HPC Europe grant for the fruitful visit to Helsinki, to the Finish Computer Science Center which hosts the supercomputer Sisu on which the computations was performed.

N	200	300	500	500	500	1000	1000	1000
%	90	80	60	70	80	40	50	60
clique	40	29	17	22	32	12	15	20
parts	152	540	2478	2231	1664	10918	10955	9823
seq	**623**	**898**	**67**	**3453**	*	**136**	**447**	**15k**
5-nopt	376	466	431	*	*	1268	*	*
5-opt	109	231	420	1401	*	1156	*	*
5-lv	126	242	424	1444	*	1168	*	*
16-nopt	123	135	119	584	*	350	*	*
16-opt	33	64	116	387	*	319	*	*
16-lv	66	71	118	407	*	329	*	*
64-nopt	49	45	29	142	*	84	368	1236
64-opt	27	16	28	93	18k	76	345	1064
64-lv	39	23	29	99	21k	79	355	1100
512-nopt	38	23	4	25	14k	11	48	158
512-opt	27	15	4	14	6595	10	45	135
512-lv	39	23	4	19	4189	10	46	142

Table 1. Random graph problems

	brock 800_3	**latin_ square _10**	**keller5**	**MANN a45 _**	**p_hat 1500-1**	**p_hat 500-3**
N	800	900	776	1035	1500	500
%	65	76	75	99	25	75
clique	25	90	27	345	12	50
parts	4888	380	420	45	14918	657
seq	7302	4902	4531	3666	278	*
5-nopt	*	*	*	1340	894	*
5-opt	*	1423	*	719	814	*
5-lv	*	1504	*	1051	824	*
16-nopt	*	531	986	402	247	*
16-opt	*	403	672	205	225	*
16-lv	*	430	686	388	232	*
64-nopt	472	150	318	183	60	*
64-opt	413	105	173	140	54	11k
64-lv	425	128	228	174	56	7165
512-nopt	64	82	138	183	9	41k
512-opt	55	62	137	140	8	11k
512-lv	59	83	138	183	8	5300

Table 2. Problems from the DIMACS challenge

	monoton-7	**monoton-8**	**monoton-9**	**deletion-9**
N	343	512	729	512
%	79	82	84	93
clique	19	23	28	52
parts	313	590	932	375
seq	7	2347	*	*
5-nopt	76	*	-	-
5-opt	74	1282	-	-
5-lv	74	1292	-	-
16-nopt	23	959	-	-
16-opt	21	408	-	-
16-lv	22	385	-	-
64-nopt	8	475	150k	-
64-opt	6	409	150k	-
64-lv	6	195	44k	-
512-nopt	4	405	150k	*
512-opt	2	409	150k	*
512-lv	4	243	31k	255k

Table 3. Problems of monotonic matrices and deletion codes

Refernces

[1] Hammersley, J.M. and Handscomb, D.C. *Monte Carlo Methods.* London. 1975. (1964).

[2] Babai, L. Monte-Carlo algorithms in graph isomorphism testing. *Université de Montréal, D.M.S.* No.79–10. http://people.cs.uchicago.edu/~laci/lasvegas79.pdf

[3] Bogdanova, G.T., Brouwer, A.E., Kapralov, S.N. and Ostergaard, P.R.J. Error-Correcting Codes over an Alphabet of Four Elements. *Designs, Codes and Cryptography.* August 2001, Volume 23, Issue 3, pp 333–342.

[4] Bomze, I.M., Budinich, M., Pardalos, P.M. and Pelillo, M. (1999). The Maximum Clique Problem. In D.-Z. Du and P.M. Pardalos (Eds.) *Handbook of Combinatorial Optimization.* (pp. 1–74.) Kluwer Academic Publishers.

[5] DIMACS. ftp://dimacs.rutgers.edu/pub/challenge/graph/ (May 30, 2014)

[6] Hasselberg, J., Pardalos, P.M. and Vairaktarakis, G. Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization.* (1993), 463–482.

[7] Luby, M. and Ertel, W. Optimal Parallelization of Las Vegas Algorithms. In Enjalbert, P at all. (Eds.), *Lecture Notes in Computer Science.* (1994). (pp. 461–474.) Springer Berlin Heidelberg.

[8] Luby, M., Sinclair, A. and Zuckerman, D. Optimal Speedup of Las Vegas Algorithms. In: *Proceedings of* the 2nd Israel Symposium on Theory of Computing and *Systems.* Jerusalem, Israel, June 1993.

[9] Ostergaard, P.R.J. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics.* (2002), 197–207.

[10] Szabo, S. Parallel algorithms for finding cliques in a graph. *Journal of Physics: Conference Series Volume 268, Number 1.* 2011 J. Phys.: Conf. Ser. 268 012030 doi:10.1088/1742-6596/268/1/012030.

[11] Szabo, S. Monotonic matrices and clique search in graphs. *Annales Univ. Sci. Budapest., Sect. Comp.* (2013), 307–322.

[12] Sloan, N. http://neilsloane.com/doc/graphs.html (May 30, 2014)

[13] Truchet, C., et al. Prediction of Parallel Speed-ups for Las Vegas Algorithms. http://arxiv.org/abs/1212.4287 2012.

[14] Zavalnij, B. Three Versions of Clique Search Parallelization. Journal of Computer Science and *Information Technology.* June 2014, Vol. 2, No. 2, pp. 09–20.