Computing Approximate b-Matchings in Parallel

Arif Khan[†], Mahantesh Halappanavar[‡], Fredrik Manne[*], Alex Pothen[†]
[†]Purdue University; (khan58, apothen)@purdue.edu
[‡]Pacific Northwest National Laboratory; Mahantesh.Halappanavar@pnnl.gov
[*]University of Bergen; Fredrik.Manne@ii.uib.no

We consider sequential and shared-memory parallel (on multicore computers) algorithms that implement a half-approximation algorithm for weighted b-matching on arbitrary graphs. Consider an undirected graph $G(V, E, w)$ with vertex set V, edge set E, and weight function $w(e) \geq 0$ for each $e \in E$, and a function $f : V \to \mathbb{Z}_+$ assigning non-negative integers to the vertices. (We assume without loss of generality that $f(v)$ is less than or equal to the degree of the vertex v.) Then a b-matching on G is a subset of edges $M \subseteq E$ such that every vertex $v \in V$ has at most $f(v)$ edges in M incident on it. The values $f(v)$ for each vertex v could be be different or the same (in the latter case $f(v) = b$ for some positive integer b, and hence the name b-matching). The usual notion of matching has $f(v) = 1$ for all v, and we will call it a 1-matching. If all vertices in M are required to have degree exactly $f(v)$, we call it a perfect b-matching. A *maximum cardinality* b-matching M has the cardinality $|M|$ as large as possible. A *maximum weight* b-matching M has the sum of weights $\sum_{e \in M} w(e)$ as large as possible. In this abstract we focus on the *maximum weight* b-matching with $f(v) >= 2$ for all v.

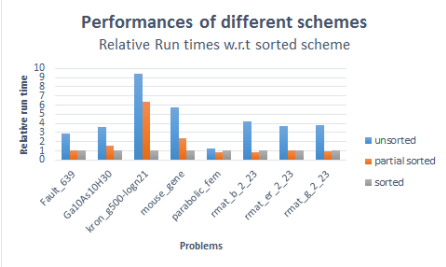
The applications of 1-matching problem include Google's Ad words problem, image recognition in computer vision, network alignment, sparse matrix computations, etc. Jabera et al. [4] have shown that b-matching is useful in various machine learning problems such as classification, spectral clustering, semi-supervised learning and graph embedding.

Let m denote the number of edges and n the number of vertices in G. The fastest exact algorithm for this problem, by Gabow and Tarjan [1], requires $O(n^{1/2}m)$ time. Fremuth-Paeger et al. [6] and Jabera et al. [4] describe exact algorithms that use min-cost flow and belief propagation techniques, respectively. Both of these algorithms have running time $O(nm)$, but the belief propagation technique is currently the fastest practical algorithm. However, these running times are prohibitive in case of even moderate-sized graphs, and hence we design linear-time approximation algorithms. The approximate edge weighted 1-matching problem has also been studied. Mestre [3] showed 1/2- and $(2/3 - \epsilon)$-approximation algorithms for b-matching by extending path-growing approximation algorithms for the 1-matching problem. Morales et al. [2] and Georgiadis et al. [5] developed $1/2-$approximation algorithms for b-matching based on the concept of *locally dominant edges*. Although the latter describes a distributed algorithm and uses different notation, their algorithm is similar to ours.

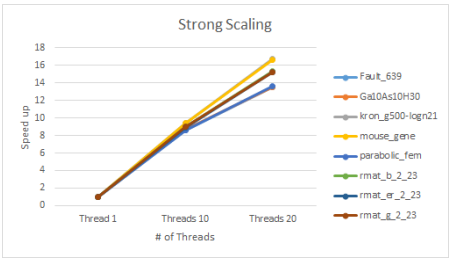
Here we propose a new $1/2-$approximation *maximum weight* b-matching algorithm, called *b-Suitor*, which we show to be practically faster than earlier algorithms. The *b-Suitor* algorithm is an extension of the *Suitor* algorithm proposed by Manne and Halappanavar [7] for solving the maximum weight 1-matching problem. All of these are serial algorithms. We also study these algorithms in the parallel (shared memory) context.

In the *b-Suitor* algorithm each vertex v proposes to a set of $f(v)$ vertices (v is a *suitor* of these vertices). Hence each vertex maintains two lists of vertices: $S(v)$ is the list of current Suitors of v, i.e., vertices that propose to match to v, and $T(v)$ is a list of vertices that v has proposed to. The process responsible for matching a vertex v includes in $T(v)$ its heaviest available $f(v)$ neighbors, where a neighbor w is unavailable if it has $f(w)$ heavier Suitors than v. This is a speculative algorithm: If v finds that it is heavier than the $f(w)$-th Suitor of w, call it y, then it unmatches y and includes itself as a Suitor of w. This process now tries to find a neighbor not included in the list $T(y)$ for y to propose to.

We can describe three variants of this algorithm, based on whether the adjacency lists of the vertices are sorted in non-increasing order of weight or not. We can choose to have unsorted



(a) Relative performance of variant algorithms



(b) Strong scaling

Figure 1: The performance of three variant Suitor algorithms, and strong scaling of the sorted Suitor algorithm for computing a 5-matching on a multicore computer.

adjacency lists, or fully sorted lists; the third option is to partially sort so that small multiples of $f(v)$ highest weighted vertices in each adjacency list are sorted. The time complexity of the unsorted algorithm is $O(m\Delta B)$, where $B = \max\{f(v) : v \in V\}$, and Δ is the maximum vertex degree. For the sorted algorithm it is $O(n + m \log \Delta)$,

We present preliminary results on computing a 5-matching from a shared memory parallel implementation of the b-Suitor algorithm. The machine is an Intel Xeon multiprocessor, with ten cores per socket, and the computer consists of two sockets. The processor speed is 3.0 GHz, and the system has 128 GB memory. Figure 1 shows the performance of eight problems, five of which are from the the University of Florida collection and three which are synthetic `RMAT` graphs. We report the first set of results for the parallel b-Suitor algorithm with $b = 5$. The results in Figure 1(a) show that sorting leads to faster algorithms, by factors upto nine, for graphs with several hundred million edges. Complete sorting seems to be better than partial sorting for most of these problems. Figure 1(b) shows strong scaling results on the twenty Intel Xeon cores, and all eight problems show good speed-ups. We will discuss the factors that influence the performance and scalability of these problems at the Workshop. We are also working with colleagues at Intel Corporation on Xeon Phi implementations.

References

[1] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM Journal of Computing*, vol. 5, no. 18, pp. 1013–1036, 1989.

[2] G. D. F. Morales, A. Gionis, and M. Sozio, "Social content matching in Mapreduce," in *"37th Int. Conf. on Very Large Data Bases (VLDB 2011)"*, vol. 4, no. 7, 2011, pp. 460–469.

[3] J. Mestre, "Greedy in approximation algorithms," in *Algorithms - ESA 2006*, Lecture Notes in Computer Science, vol. 4168. Springer, 2006, pp. 528–539.

[4] B. C. Huang and T. Jebara, "Fast b-matching via sufficient selection belief propagation," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011*, ser. JMLR Proceedings, vol. 15. 2011, pp. 361–369.

[5] G. Georgiadis and M. Papatriantafilou, "Overlays with preferences: Distributed, adaptive approximation algorithms for matching with preference lists," *Algorithms*, vol. 6, no. 4, pp. 824–856, 2013.

[6] Fremuth-Paeger, Christian, and Dieter Jungnickel. "Balanced network flows. I. A unifying framework for design and analysis of matching algorithms." Networks 33.1 (1999): 1-28.

[7] F. Manne and M. Halappanavar. "New effective multithreaded matching algorithms", Proceedings of IPDPS 2014, to appear.