FIMI'03: Workshop on Frequent Itemset Mining Implementations

FIMI Repository: http://fimi.cs.helsinki.fi/
FIMI Repository Mirror: http://www.cs.rpi.edu/~zaki/FIMI03/

Bart Goethals
HIIT Basic Research Unit
Department of Computer Science
University of Helsinki, Finland
bart.goethals@cs.helsinki.fi

Mohammed J. Zaki
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York, USA
zaki@cs.rpi.edu

1 Why Organize the FIMI Workshop?

Since the introduction of association rule mining in 1993 by Agrawal Imielinski and Swami [3], the frequent itemset mining (FIM) tasks have received a great deal of attention. Within the last decade, a phenomenal number of algorithms have been developed for mining all [3–5, 10, 18, 19, 21, 23, 26, 28, 31, 33], closed [6, 12, 22, 24, 25, 27, 29, 30, 32] and maximal frequent itemsets [1, 2, 7, 11, 15–17, 20, 35]. Every new paper claims to run faster than previously existing algorithms, based on their experimental testing, which is oftentimes quite limited in scope, since many of the original algorithms are not available due to intellectual property and copyright issues. Zheng, Kohavi and Mason [34] observed that the performance of several of these algorithms is not always as claimed by its authors, when tested on some different datasets. Also, from personal experience, we noticed that even different implementations of the same algorithm could behave quite differently for various datasets and parameters.

Given this proliferation of FIM algorithms, and sometimes contradictory claims, there is a pressing need to benchmark, characterize and understand the algorithmic performance space. We would like to understand why and under what conditions one algorithm outperforms another. This means testing the methods for a wide variety of parameters, and on different datasets spanning dense and sparse, real and synthetic, small and large, and so on.

Given the experimental, algorithmic nature of FIM (and most of data mining in general), it is crucial that other researchers be able to independently verify the claims made in a new paper. Unfortunately, the FIM community (with few exceptions) has a very poor track record in this regard. Many new algorithms are not available even as an executable, let alone the source code. How many times have we heard "this is proprietary software, and not available." This is not the way other sciences work. Independent verifiability is the hallmark of sciences like physics, chemistry, biology, and so on. One may argue, that the nature of research is different, they have detailed experimental procedure that can be replicated, while we have algorithms, and there is more than one way to code an algorithm. However, a good example to emulate is the bioinformatics community. They have espoused the open-source paradigm with more alacrity than we have. It is quite common for journals and conferences in bioinformatics to *require* that software be available. For example, here is a direct quote from the journal *Bioinformatics* (http://bioinformatics.oupjournals.org/):

> Authors please note that software should be available for a full 2 YEARS after publication of the manuscript.

We organized the FIMI workshop to address the three main deficiencies in the FIM community:

- Lack of publicly available implementations of FIM algorithms

- Lack of publicly available "real" datasets
- Lack of any serious performance benchmarking of algorithms

1.1 FIMI Repository

The goals of this workshop are to find out the main implementation aspects of the FIM problem for all, closed and maximal pattern mining tasks, and evaluating the behavior of the proposed algorithms with respect to different types of datasets and parameters. One of the most important aspects is that only open source code submissions are allowed and that all submissions will become freely available (for research purposes only) on the online FIMI repository along with several new datasets for benchmarking purposes. See the URL: `http://fimi.cs.helsinki.fi/`.

1.2 Some Recommendations

We strongly urge all new papers on FIM to provide access to source code or at least an executable immediately after publication. We request that researchers to contribute to the FIMI repository both in terms of algorithms and datasets. We also urge the data mining community to adopt the open-source strategy, which will serve to accelerate the advances in the field. Finally, we would like to alert reviewers that the FIMI repository now exists, and it contains state-of-the-art FIM algorithms, so there is no excuse for a new paper to not do an extensive comparison with methods in the FIMI repository. Such papers should, in our opinion, be rejected outright!

2 The Workshop

This is a truly unique workshop. It consisted of code submission as well as a paper submission describing the algorithm and a detailed performance study by the authors on publicly provided datasets, along with a detailed explanation on when and why their algorithm performs better than existing implementations. The submissions were tested independently by the co-chairs, and the papers were reviewed by members of the program committee. The algorithms were judged for three main tasks: all frequent itemsets mining, closed frequent itemset mining, and maximal frequent itemset mining.

The workshop proceedings contains 15 papers describing 18 different algorithms that solve the frequent itemset mining problems. The source code of the implementations of these algorithms is publicly available on the FIMI repository site.

The conditions for "acceptance" of a submission were as follows: i) a correct implementation for the given task, ii) an efficient implementation compared with other submissions in the same category or a submission that provides new insight into the FIM problem. The idea is to highlight both successful and unsuccessful but interesting ideas. One outcome of the workshop will be to outline the focus for research on new problems in the field.

In order to allow a fair comparison of these algorithms, we performed an extensive set of experiments on several real-life datasets, and a few synthetic ones. Among these are three new datasets, i.e. a supermarket basket dataset donated by Tom Brijs [9], a dataset containing click-stream data of a Hungarian on-line news portal donated by Ferenc Bodon [8], and a dataset containing Belgian traffic accident descriptions donated by Karolien Geurts [13].

2.1 Acknowledgments

We would like to thank the following program committee members for their useful suggestions and reviews:

- Roberto Bayardo, IBM Almaden Research Center, USA
- Johannes Gehrke, Cornell University, USA
- Jiawei Han, University of Illinois at Urbana-Champaign, USA
- Hannu Toivonen, University of Helsinki, Finland

We also thank Taneli Mielikäinen and Toon Calders for their help in reviewing the submissions.

We extend our thanks to all the participants who made submissions to the workshop, since their willingness to participate and contribute source-code in the public domain was essential in the creation of the FIMI Repository. For the same reason, thanks are due to Ferenc Bodon, Tom Brijs, and Karolien Geurts, who contributed new datasets, and to Zheng, Kohavi and Mason for the KDD Cup 2001 datasets.

3 The FIMI Tasks Defined

Let's assume we are given a set of items $\mathcal{I}$. An *itemset* $I \subseteq \mathcal{I}$ is some subset of items. A *transaction* is a couple $T = (tid, I)$ where tid is the transaction identifier and I is an itemset. A transaction $T = (tid, I)$ is said to *support* an itemset X, if $X \subseteq I$. A *transaction database* $\mathcal{D}$ is a set of transactions such that each transaction has a unique identifier. The *cover*

of an itemset X in $\mathcal{D}$ consists of the set of transaction identifiers of transactions in $\mathcal{D}$ that support X: $cover(X, \mathcal{D}) := \{tid \mid (tid, I) \in \mathcal{D}, X \subseteq I\}$. The *support* of an itemset X in $\mathcal{D}$ is the number of transactions in the cover of X in $\mathcal{D}$:

$$support(X, \mathcal{D}) := |cover(X, \mathcal{D})|.$$

An itemset is called *frequent* in $\mathcal{D}$ if its support in $\mathcal{D}$ exceeds a given minimal support threshold σ. $\mathcal{D}$ and σ are omitted when they are clear from the context. The goal is now to find all frequent itemsets, given a database and a minimal support threshold.

The search space of this problem, all subsets of $\mathcal{I}$, is clearly huge, and a frequent itemset of size k implies the presence of 2^k-2 other frequent itemsets as well, i.e., its nonempty subsets. In other words, if frequent itemsets are long, it simply becomes infeasible to mine the set of all frequent itemsets. In order to tackle this problem, several solutions have been proposed that only generate a representing subset of all frequent itemsets. Among these, the collections of all *closed* or *maximal* itemsets are the most popular.

A frequent itemset I is called *closed* if it has no frequent superset with the same support, i.e., if

$$I = \bigcap_{(tid, J) \in cover(I)} J$$

An frequent itemset is called *maximal* if it has no superset that is frequent.

Obviously, the collection of maximal frequent itemsets is a subset of the collection of closed frequent itemsets which is a subset of the collection of all frequent itemsets. Although all maximal itemsets characterize all frequent itemsets, the supports of all their subsets is not available, while this might be necessary for some applications such as association rules. On the other hand, the closed frequent itemsets form a lossless representation of all frequent itemsets since the support of those itemsets that are not closed is uniquely determined by the closed frequent itemsets. See [14] for a recent survey of the FIM algorithms.

4 Experimental Evaluation

We conducted an extensive set of experiments for different datasets, for all of the algorithms in the three categories (all, closed and maximal). Figure 1 shows the data characteristics.

Our target platform was a Pentium4, 3.2 GHz Processor, with 1GB of memory, using a WesternDigital IDE 7200rpms, 200GB, local disk. The operating system was Redhat Linux 2.4.22 and we used gcc 3.2.2 for the compilation. Other platforms were also tried, such as an older dual 400Mhz Pentium III processors with 256MB memory, but a faster SCSI 10,000rpms disk. Independent tests were also run on quad 500Mhz Pentium III processors, with 1GB memory. There were some minor differences, which have been reported on the workshop website. Here we refer to the target platform (3.2Ghz/1GB/7200rpms).

All times reported are *real* times, including system and user times, as obtained from the unix `time` command. All algorithms were run with the output flag turned on, which means that mined results were written to a file. We made this decision, since in the real world one wants to see the output, and the total wall clock time is the end-to-end delay that one will see. There was one unfortunate consequence of this, namely, we were not able to run algorithms for mining all frequent itemsets below a certain threshold, since the output file exceeded the 2GB file size limit on a 32bit platform. For each algorithm we also recorded its memory consumption using the `memusage` command. Results on memory usage are available on the FIMI website.

For the experiments, each algorithm was allocated a maximum of 10 minutes to finish execution, after which point it was killed. We had to do this to finish the evaluation in a reasonable amount of time. We had a total of 18 algorithms in the *all* category, 6 in the *closed* category, and 8 in the *maximal* category, for a grand total of 32 algorithms. Please note the algorithms eclat_zaki, eclat_goethals, charm and genmax, were not technically submitted to the workshop, however we included them in the comparison since their source code is publicly available. We used 14 datasets, with an average of 7 values of minimum support. With a 10 minute time limit per algorithm, the total time to finish *one round* of evaluation took 31360 minutes of running time, which translates to an upper-bound of 21 days! Since not all algorithms take a full 10 minute, the actual time for one round was roughly 7-10 days.

We should also mention that some algorithms had problems on certain datasets. For instance for mining all frequent itemsets, armor is not able to handle dense datasets very well (for low values of minimum support it crashed for chess, mushroom, pumsb); pie gives a segmentation fault for bms2, chess, retail and the synthetic datasets; cofi gets killed for bms1 and kosarak; and dftime/dfmem crash for accidents, bms1 and retail. For closed itemset mining, fpclose segmentfaults for bms1, bms2, bmspos and retail; borgelt_eclat also has problems with retail. Finally, for maximal set mining, apriori_borgelt crashes for bms1 for low value of support and so does eclat_borgelt for pumsb.

Database	#Items	Avg. Length	#Transactions
accidents	468	33.8	340,183
bms1	497	2.5	59,602
bms2	3341	5.6	77,512
bmspos	1658	7.5	515,597
chess	75	37	3,196
connect	129	43	67,557
kosarak	41,270	8.1	990,002
mushroom	119	23	8,124
pumsb*	2088	50.5	49,046
pumsb	2113	74	49,046
retail	16,469	10.3	88,162
T10I5N1KP5KC0.25D200K	956	10.3	200,000
T20I10N1KP5KC0.25D200K	979	20.1	200,000
T30I15N1KP5KC0.25D200K	987	29.7	200,000

Figure 1. Database Characteristics

4.1 Mining All Frequent Itemsets

Figures 5 and 6 show the timings for the algorithms for mining all frequent itemsets. Figure 2 shows the best and second-best algorithms for high and low values of support for each dataset.

There are several interesting trends that we observe:

1. In some cases, we observe a high initial running time of the highest value of support, and the time drops for the next value of minimum support. This is due to file caching. Each algorithm was run with multiple minimum support values before switching to another algorithm. Therefore the first time the database is accessed we observe higher times, but on subsequent runs the data is cached and the I/O time drops.

2. In some cases, we observe that there is a cross-over in the running times as one goes from high to low values of support. An algorithm may be the best for high values of support, but the same algorithm may not be the best for low values.

3. There is *no one best* algorithm either for high values or low values of support, but some algorithms are the best or runner-up more often than others.

Looking at Figure 2, we can conclude that for high values the best algorithms are either kdci or patricia, across all databases we tested. For low values, the picture is not as clear; the algorithms likely to perform well are patricia, fpgrowth* or lcm. For the runner-up in the low support category, we once again see patricia and kdci showing up.

4.2 Mining Closed Frequent Itemsets

Figures 7 and 8 show the timings for the algorithms for mining closed frequent itemsets. Figure 3 shows the best and second-best algorithms for high and low values of support for each dataset. For high support values, fpclose is best for 7 out of the 14 datasets, and lcm, afopt, and charm also perform well on some datasets. For low values of support the competition is between fpclose and lcm for the top spot. For the runner-up spot there is a mixed bag of algorithms: fpclose, afopt, lcm and charm. If one were to pick an overall best algorithm, it would arguably be fpclose, since it either performs the best or shows up in the runner-up spot, more times than any other algorithm. An interesting observation is that for the cases where fpclose doesn't appear in the table it gives a segmentation fault (for bms1, bms2, bmspos and retail).

4.3 Mining Maximal Frequent Itemsets

Figures 9 and 10 show the timings for the algorithms for mining maximal frequent itemsets. Figure 4 shows the best and second-best algorithms for high and low values of support for each dataset. For high values of support fpmax* is the dominant winner or runner-up. Genmax, mafia and afopt also are worth mentioning. For the low support category fpmax* again makes a strong show as the best in 7 out of 14 databases, and when it is not best, it appears as the runner-up 6 times. Thus fpmax* is the method of choice for maximal pattern mining.

4.4 Conclusions

We presented only some of the results in this report. We refer the reader to the FIMI repository for a more detailed experimental study. The study done by us was also somewhat limited, since we performed only timing and memory usage experiments for given datasets. Ideally, we would have liked to do a more detailed study of the scale-up of the algorithms, and for a variety of different parameters; our preliminary studies show that *none* of the algorithms is able to gracefully scale-up to very large datasets, with millions of transactions. One reason may be that most methods are optimized for in-memory datasets, which points to the area of *out-of-core* FIM algorithms an avenue for future research.

In the experiments reported above, there were no clear winners, but some methods did show up as the best or second best algorithms for both high and low values of support. Both patricia and kdci represent the state-of-the-art in all frequent itemset mining, whereas fpclose takes this spot for closed itemset mining, and finally fpmax* appears to be one of the best for maximal itemset mining. An interesting observation is that for the synthetic datasets, apriori_borgelt seems to perform quite well for all, closed and maximal itemset mining.

We refer the reader to the actual papers in these proceedings to find out the details on each of the algorithms in this study. The results presented here should be taken in the spirit of *experiments-in-progress*, since we do plan to diversify our testing to include more parameters. We are confident that the workshop will generate a very healthy and critical discussion on the state-of-affairs in frequent itemset mining implementations.

To conclude, we hope that the FIMI workshop will serve as a model for the data mining community to hold more such open-source benchmarking tests, and we hope that the FIMI repository will continue to grow with the addition of new algorithms and datasets, and once again to serve as a model for the rest of the data mining world.

References

[1] C. Aggarwal. Towards long pattern generation in dense databases. *SIGKDD Explorations*, 3(1):20–26, 2001.

[2] R. Agrawal, C. Aggarwal, and V. Prasad. Depth First Generation of Long Patterns. In *7th Int'l Conference on Knowledge Discovery and Data Mining*, Aug. 2000.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216. ACM Press, 1993.

[4] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. Fayyad and et al, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA, 1996.

[5] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th VLDB Conference*, Sept. 1994.

[6] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2), Dec. 2000.

[7] R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD Conf. Management of Data*, June 1998.

[8] F. Bodon. A fast apriori implementation. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.

[9] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Knowledge Discovery and Data Mining*, pages 254–260, 1999.

[10] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *ACM SIGMOD Conf. Management of Data*, May 1997.

[11] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: a maximal frequent itemset algorithm for transactional databases. In *Intl. Conf. on Data Engineering*, Apr. 2001.

[12] D. Cristofor, L. Cristofor, and D. Simovici. Galois connection and data mining. *Journal of Universal Computer Science*, 6(1):60–73, 2000.

[13] K. Geurts, G. Wets, T. Brijs, and K. Vanhoof. Profiling high frequency accident locations using association rules. In *Proceedings of the 82nd Annual Transportation Research Board*, page 18, 2003.

[14] B. Goethals. *Efficient Frequent Pattern Mining*. PhD thesis, transnational University of Limburg, Belgium, 2002.

[15] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *1st IEEE Int'l Conf. on Data Mining*, Nov. 2001.

[16] G. Grahne and J. Zhu. High performance mining of maximal frequent itemsets. In *6th International Workshop on High Performance Data Mining*, May 2003.

[17] D. Gunopulos, H. Mannila, and S. Saluja. Discovering all the most specific sentences by randomized algorithms. In *Intl. Conf. on Database Theory*, Jan. 1997.

[18] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Conf. Management of Data*, May 2000.

[19] M. Houtsma and A. Swami. Set-oriented mining of association rules in relational databases. In *11th Intl. Conf. Data Engineering*, 1995.

[20] D.-I. Lin and Z. M. Kedem. Pincer-search: A new algorithm for discovering the maximum frequent set. In *6th Intl. Conf. Extending Database Technology*, Mar. 1998.

[21] J.-L. Lin and M. H. Dunham. Mining association rules: Anti-skew algorithms. In *14th Intl. Conf. on Data Engineering*, Feb. 1998.

[22] F. Pan, G. Cong, A. Tung, J. Yang, and M. Zaki. CARPENTER: Finding closed patterns in long biological datasets. In *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, Aug. 2003.

[23] J. S. Park, M. Chen, and P. S. Yu. An effective hash based algorithm for mining association rules. In *ACM SIGMOD Intl. Conf. Management of Data*, May 1995.

[24] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *7th Intl. Conf. on Database Theory*, Jan. 1999.

[25] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In *SIGMOD Int'l Workshop on Data Mining and Knowledge Discovery*, May 2000.

[26] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *21st VLDB Conf.*, 1995.

[27] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *ACM SIGMOD Intl. Conf. Management of Data*, May 2000.

[28] H. Toivonen. Sampling large databases for association rules. In *22nd VLDB Conf.*, 1996.

[29] J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, Aug. 2003.

[30] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372-390, May-June 2000.

[31] M. J. Zaki and K. Gouda. Fast vertical mining using Diffsets. In *9th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, Aug. 2003.

[32] M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *2nd SIAM International Conference on Data Mining*, Apr. 2002.

[33] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *3rd Intl. Conf. on Knowledge Discovery and Data Mining*, Aug. 1997.

[34] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 401–406. ACM Press, 2001.

[35] Q. Zou, W. Chu, and B. Lu. Smartminer: a depth first algorithm guided by tail information for mining maximal frequent itemsets. In *2nd IEEE Int'l Conf. on Data Mining*, Nov. 2002.

	High Support		Low Support	
Database	1st	2nd	1st	2nd
accidents	kdci	eclat_zaki	fpgrowth*	patricia
bms1	patricia	lcm		
bms2	patricia	lcm	lcm	patricia
bmspos	kdci	patricia	fpgrowth*	patricia
chess	patricia	kdci	lcm	patricia
connect	kdci	aim	lcm	kdci
kosarak	kdci	patricia	patricia	afopt
mushroom	kdci	lcm	lcm	kdci
pumsb	patricia	fpgrowth*	mafia	lcm
pumsb*	kdci	aim/patricia	patricia	kdci
retail	patricia	afopt	lcm	patricia/afopt
T10I5N1KP5KC0.25D200K	patricia	fpgrowth*	fpgrowth*	patricia
T20I10N1KP5KC0.25D200K	kdci	apriori_borgelt	fpgrowth*	lcm
T30I15N1KP5KC0.25D200K	kdci	eclat_zaki/apriori_borgelt	apriori_borgelt	fpgrowth*

Figure 2. All FIM: Best (1st) and Runner-up (2nd) for High and Low Supports

	High Support		Low Support	
Database	1st	2nd	1st	2nd
accidents	charm	fpclose	fpclose	afopt
bms1	lcm	fpclose	lcm	fpclose
bms2	lcm	apriori_borgelt	lcm	charm
bmspos	apriori_borgelt	charm/afopt	lcm	charm
chess	lcm	fpclose	lcm	fpclose
connect	fpclose	afopt	lcm	fpclose
kosarak	fpclose	charm	fpclose	afopt
mushroom	fpclose	afopt	fpclose	lcm
pumsb	fpclose/charm	afopt	lcm	fpclose
pumsb*	fpclose	afopt/charm	fpclose	afopt
retail	afopt	lcm	lcm	apriori_borgelt
T10I5N1KP5KC0.25D200K	fpclose	afopt	fpclose	lcm
T20I10N1KP5KC0.25D200K	apriori_borgelt	charm	fpclose	lcm
T30I15N1KP5KC0.25D200K	fpclose	apriori_borgelt	apriori_borgelt	fpclose

Figure 3. Closed FIM: Best (1st) and Runner-up (2nd) for High and Low Supports

	High Support		Low Support	
Database	1st	2nd	1st	2nd
accidents	genmax	fpmax*	fpmax*	mafia/genmax
bms1	fpmax*	lcm	lcm	fpmax*
bms2	afopt	fpmax*	afopt	fpmax*
bmspos	fpmax*	genmax	fpmax*	afopt
chess	fpmax*	afopt	mafia	fpmax*
connect	fpmax*	afopt	fpmax*	afopt
kosarak	fpmax*	genmax	afopt	fpmax*
mushroom	fpmax*	mafia	fpmax*	mafia
pumsb	genmax	fpmax*	fpmax*	afopt
pumsb*	fpmax*	mafia	mafia	fpmax*
retail	afopt	lcm	afopt	lcm
T10I5N1KP5KC0.25D200K	fpmax*	afopt	fpmax*	afopt
T20I10N1KP5KC0.25D200K	apriori_borgelt	genmax	fpmax*	afopt
T30I15N1KP5KC0.25D200K	genmax	fpmax*	apriori_borgelt	fpmax*

Figure 4. Maximal FIM: Best (1st) and Runner-up (2nd) for High and Low Supports

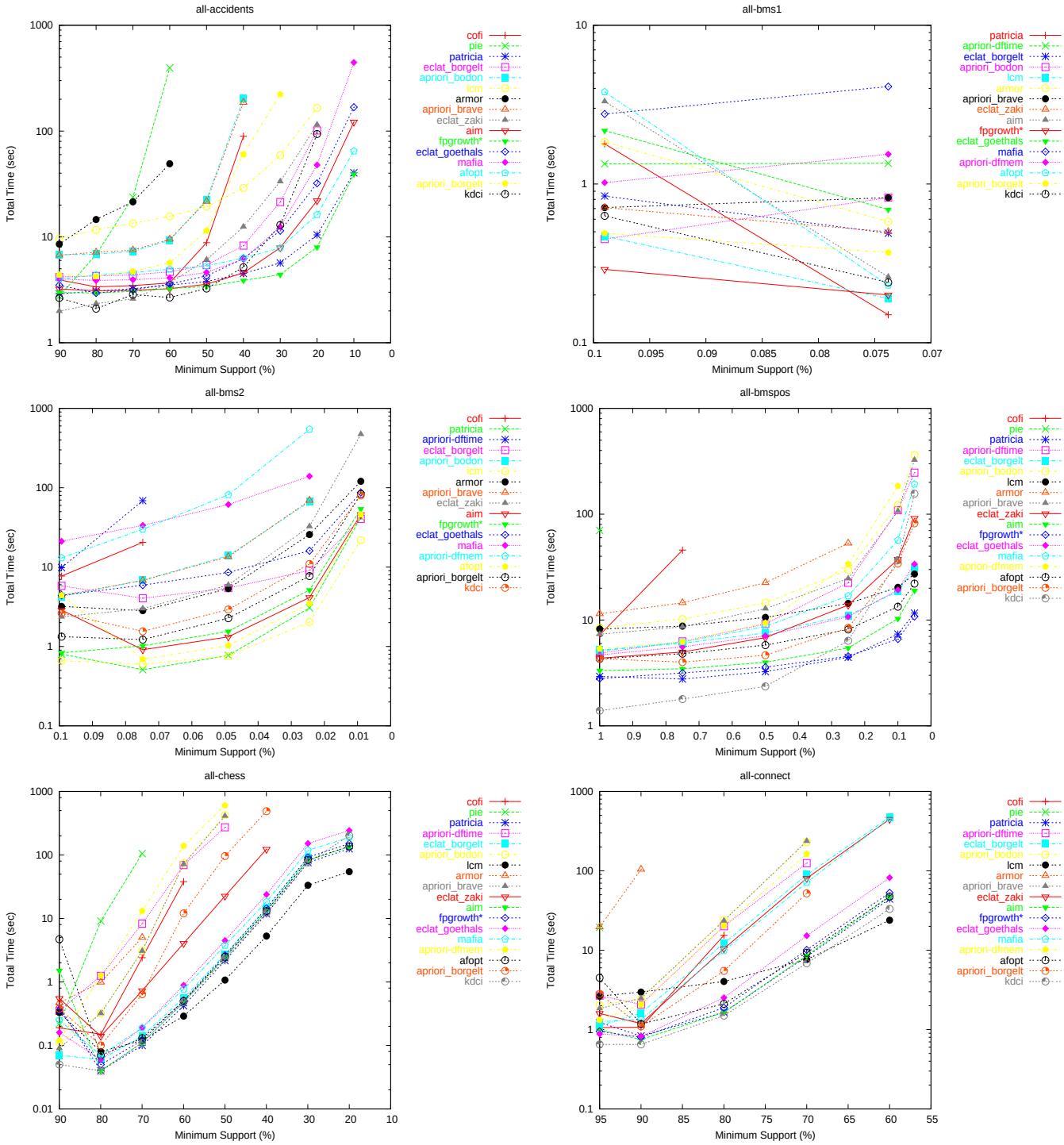

Figure 5. Comparative Performance: All

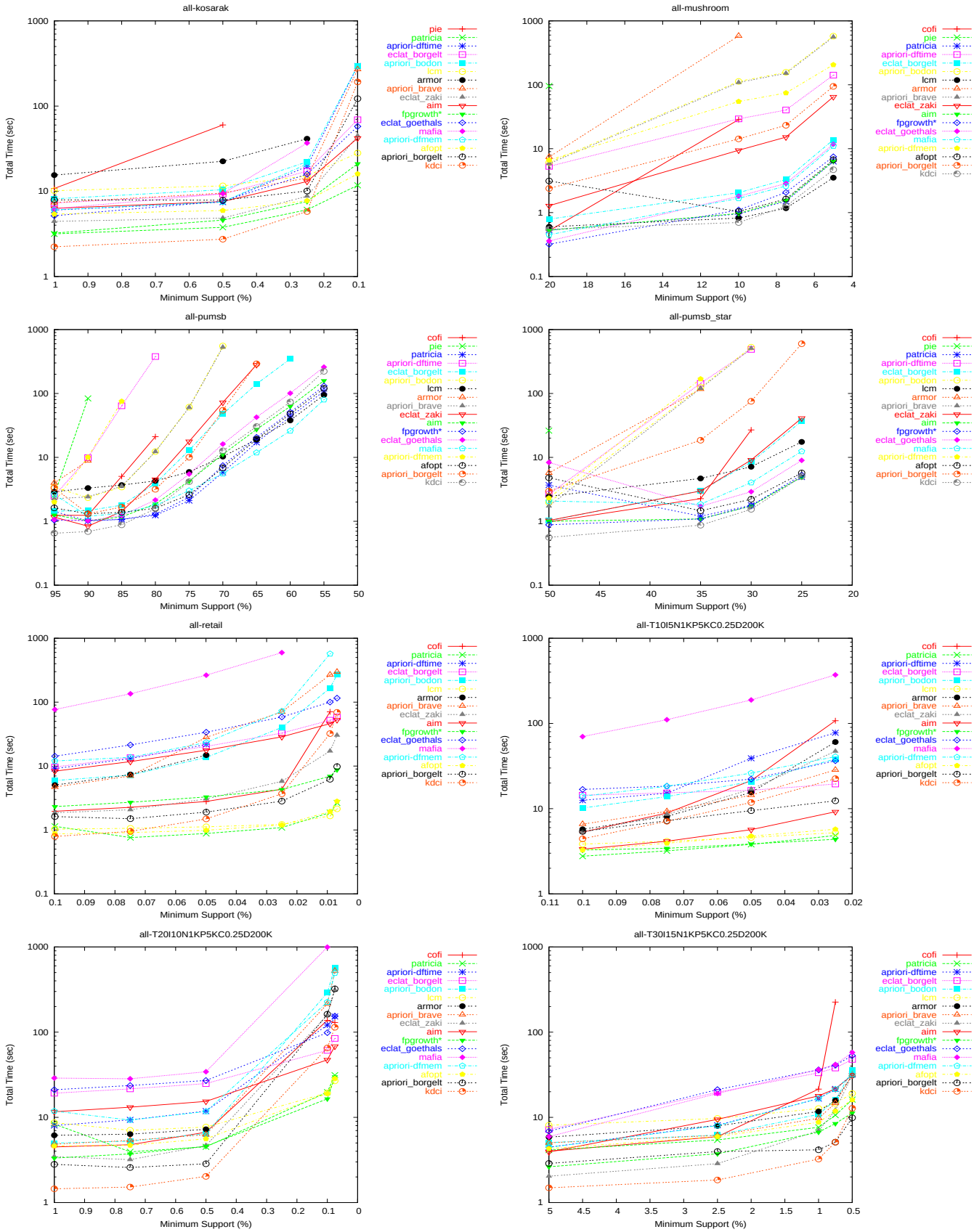

Figure 6. Comparative Performance: All

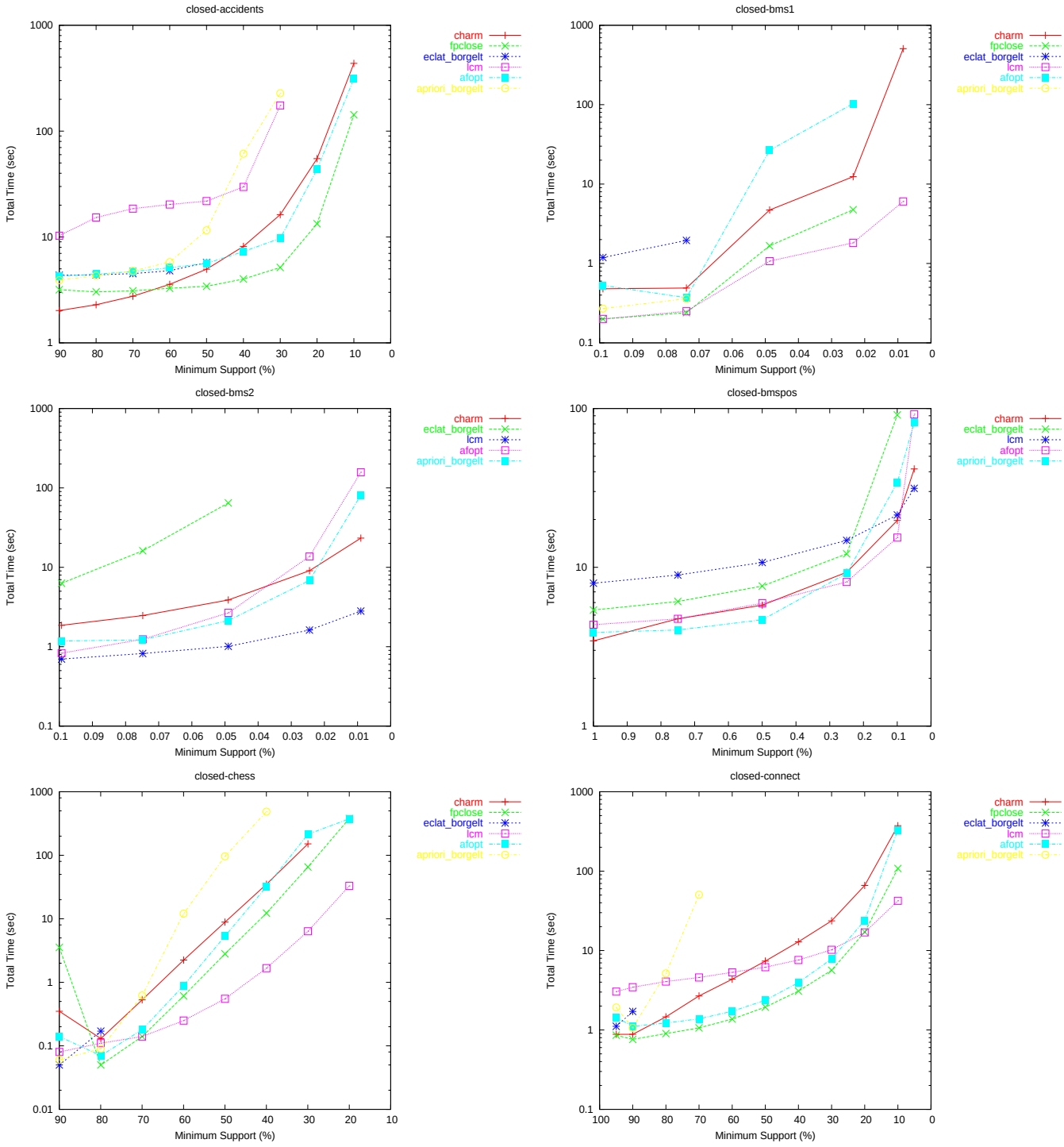

Figure 7. Comparative Performance: Closed

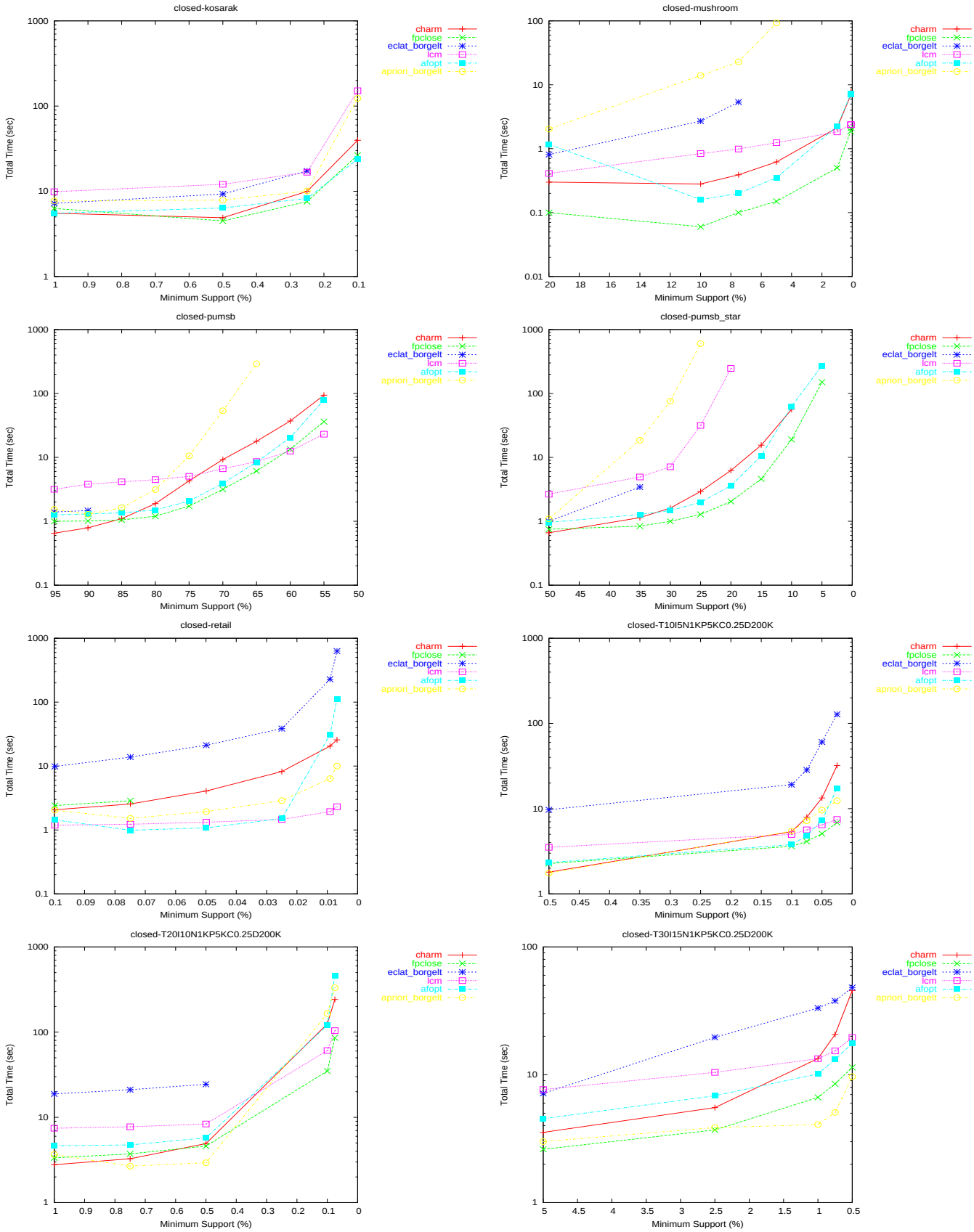

Figure 8. Comparative Performance: Closed

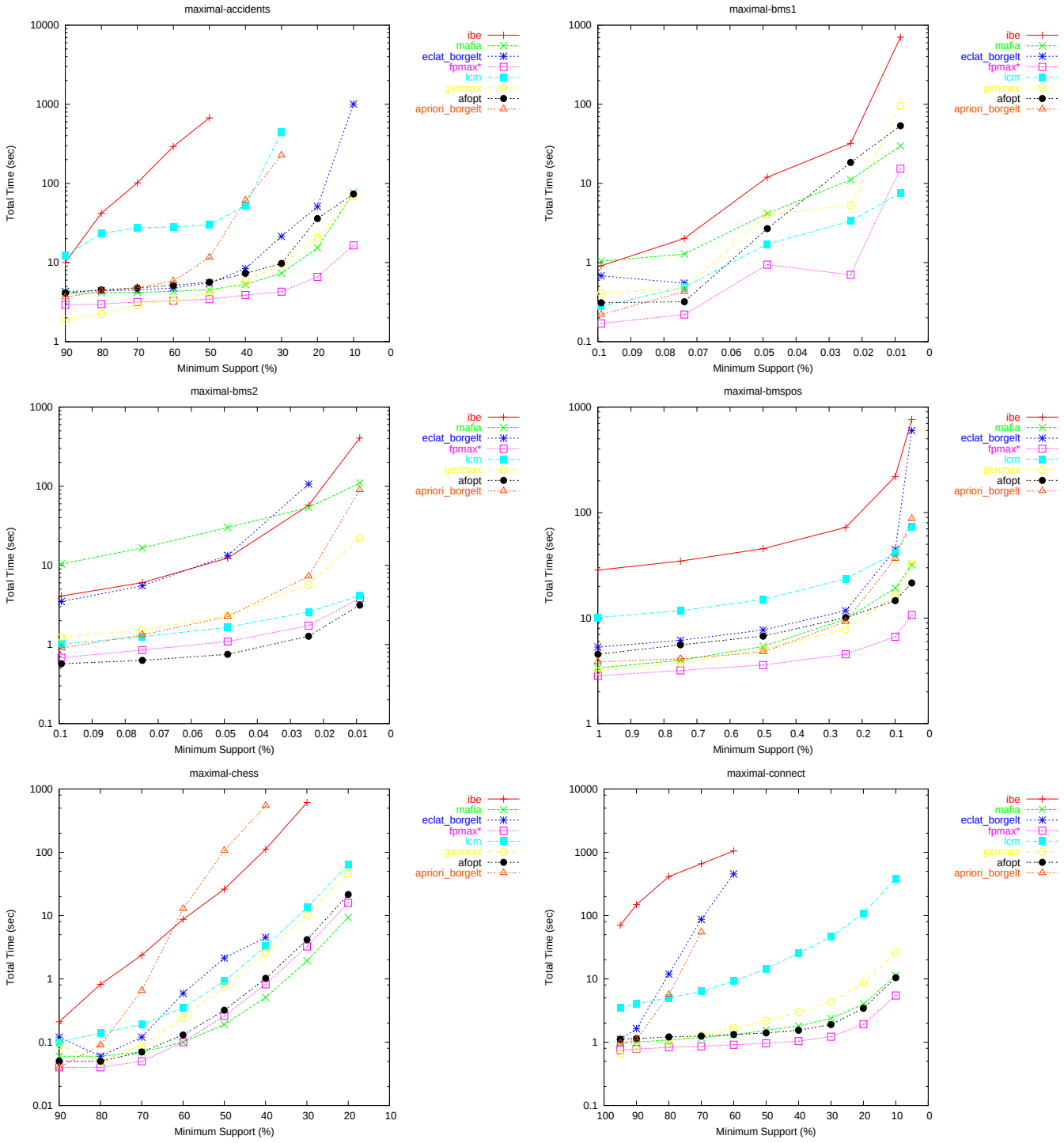

Figure 9. Comparative Performance: Maximal

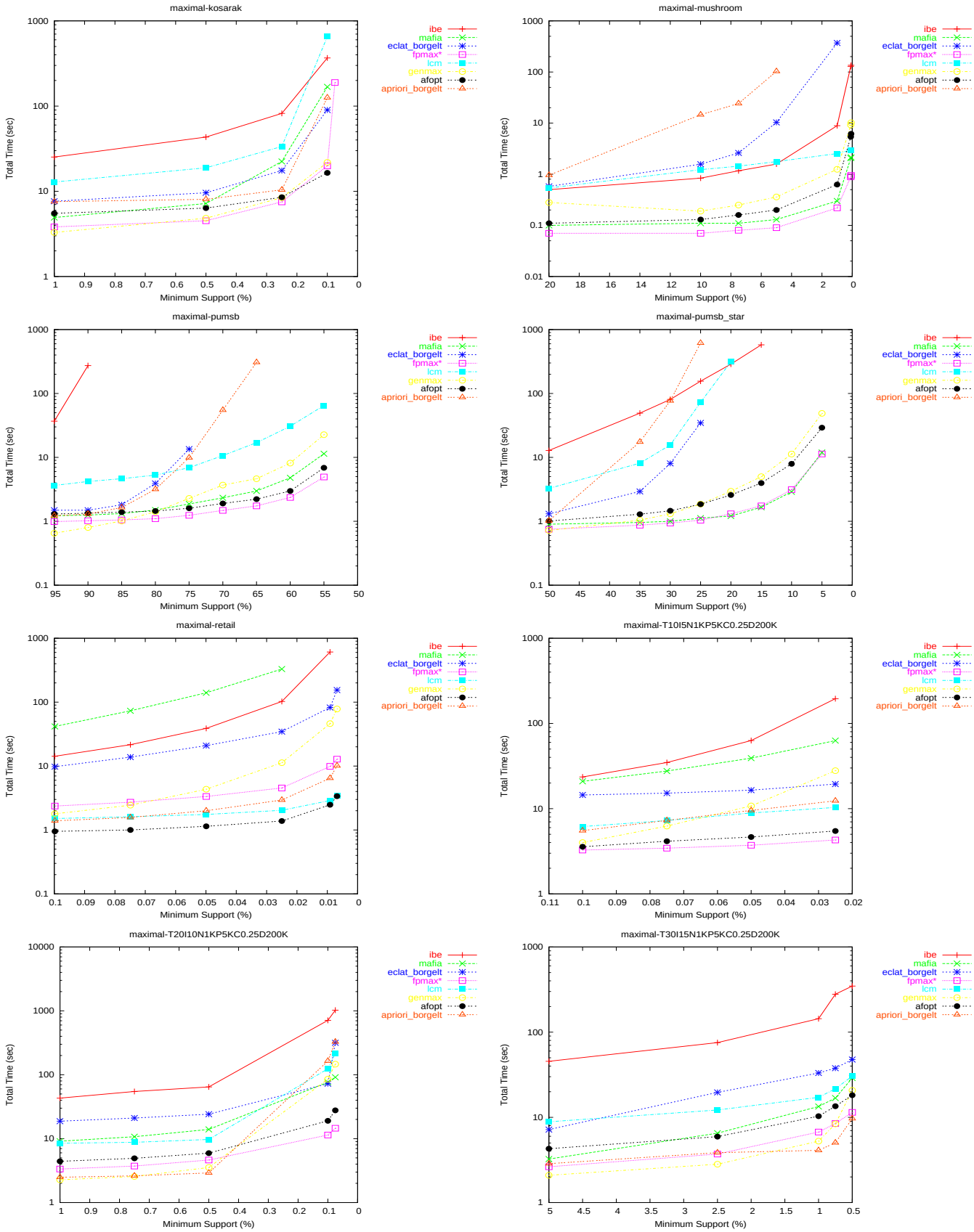

Figure 10. Comparative Performance: Maximal