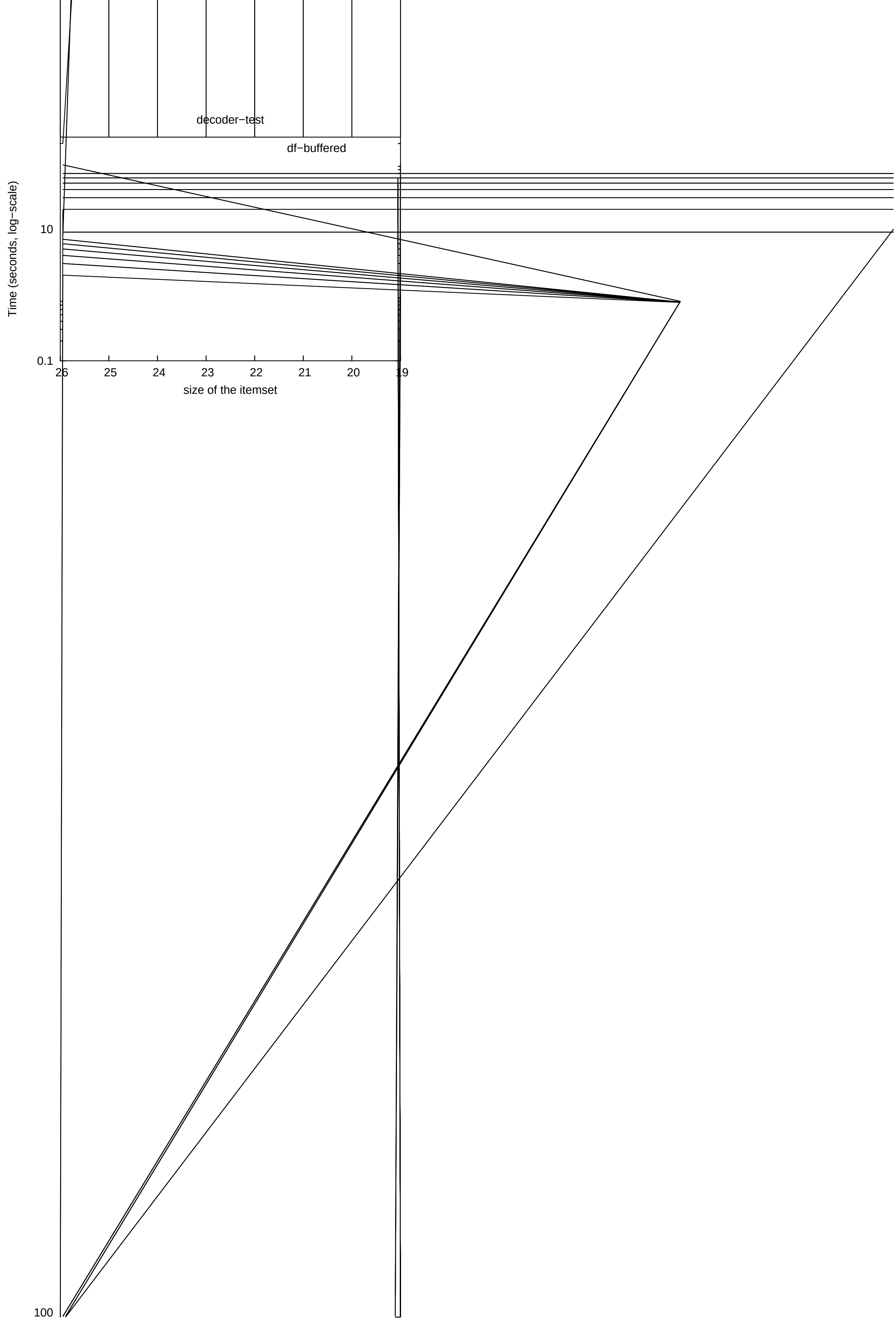
cade, over 100 papers were published
set Mining. Some of these brought
hile others tuned them with heuristics
imizations. There is one common fea-
an implementation was benchmarked
ter/better (in memory usage or disk
other (publicly available) implementa-
alled for the "Frequent Itemset Mining Implementa-


1
0.1
0.01

```
class AlgBase{
  void f(args);
};
inline void AlgBase::f(args){ ... }

class SpecAlg {
  void f(args);
};
inline void SpecAlg::f(args){ ... }

template <class T>
class Alg : public T{
  void g()
}
void
```



decoder−test
df−buffered
Time (seconds, log−scale)
10
0.1
100
26
25
24
23
22
21
20
19
size of the itemset

executed on mispredicted branches and were rolled back.

- The comparison of `eclat-cover` and `eclat-diffset` gives some really interesting results. Traditionally it is believed that di sets are well suited to dense datasets, and covers to sparse datasets, because the respective representation gives shorter lists to merge. However, in the displayed case, `eclat-cover` and `eclat-diffset` require roughly the same amount of memory accesses (30% di erence at most, depending on which metric we look at), while in the run time we see over 2-fold increase. The amount of memory accesses hint that

lows: few ten KB for L1 cache (16–32 KB in Intel Pentium 4, 64–128 KB in AMD processors), while 512 KB–2 MB for L2 cache. Non-mainstream (value market) processors may have considerably smaller caches.

When a data that is loaded was recently used, there is a high chance of finding it in the cache memory. However, when the program has to process a large amount of data (sequentially), then almost all data accesses will be cache misses, thus the execution engine will wait for the memory, then process the data segment it got, then issue the next memory read request, wait for the memory, etc. The memory interface and the execution units will be alternately idle. To overcome this, the prefetch