The Match Fit Algorithm: A Testbed for the Computational Motivation of Attention

Joseph G. Billock[1], Demetri Psaltis[1], and Christof Koch[1]

California Institute of Technology
Pasadena, CA 91125, USA
billgr@sunoptics.caltech.edu

Abstract. We present an assessment of the performance of a new on-line bin packing algorithm, which can interpolate smoothly from the Next Fit to Best Fit algorithms, as well as encompassing a new class of heuristic which packs multiple blocks at once. The performance of this novel $O(n)$ on-line algorithm can be better than that of the Best Fit algorithm. The new algorithm runs about an order of magnitude slower than Next Fit, and about two orders of magnitude faster than Best Fit, on large sample problems. It can be tuned for optimality in performance by adjusting parameters which set its working memory usage, and exhibits a sharp threshold in this optimal parameter space as time constraint is varied. These optimality concerns provide a testbed for the investigation of the value of memory and attention-like properties to algorithms.

1 Introduction

The computational abilities of humans and computers are in many ways complementary. Computers are good at routine, serialized tasks where high degrees of precision are required. Humans are good at dealing with novel situations in cases where highly precise operation is less important. One contributing factor to this is the computational architecture used by computers and humans. Computer algorithms usually approach problems with the goal of seeking exact solutions, or at least solutions which are optimal in some sense. To do this, they use as much information about the problem domain as possible. Human computation, we argue, is bound by a different set of constraints. The computational architecture of the brain is set up in a way which sharply limits the amount of information reaching its planning areas. This bottleneck (which mandates the contents of "awareness") makes humans very good at generalizing, dealing with novel situations, and responding quickly, but less good at finding exact solutions.

We would like to explore the area where the demands placed on computer algorithms are more similar to those which the brain handles–situations where highly complex problems defy exact solutions, and where external time pressure forces rapid response– and investigate how computer algorithms deal with these constraints.

The bin packing problem is a promising testbed. It is a known NP hard problem ([2], [3]), and is very general, with applications to cutting stock, machine and

© Springer-Verlag Berlin Heidelberg 2001

job scheduling, parallel processing scheduling, FPGA layout, loading problems, and more ([4], [1]). In its most basic form, the problem is phrased thus:

Definition 1. *Given a set S of real numbers in $(0,1]$, we wish to find the smallest possible number k such that there are k subsets of S, s_i, $i = 1..k$, with $s_i \cap s_j = \emptyset$ for $i \neq j$, $\cup s_i = S$, and $\forall s_i \sum\{s_i\} \leq 1$.*

One can think of this problem as an assortment of blocks of varying sizes being fit into bins of unit size. The goal is to fit the blocks into as few bins as possible without overfilling them. The elegance of the bin-packing problem has attracted much attention. Since finding exact solutions for NP problems is computationally intractable, researchers have generally attempted to find heuristics which perform well, and to analyze this performance. A large number of heuristic approaches have been suggested. These can be classified into on-line and meta-heuristic approaches. The on-line approaches (such as the Best Fit algorithm) are in general much, much faster than the meta-heuristic approaches (such as genetic algorithms or simulated annealing). In this paper, we will present and characterize a new online heuristic for the bin packing problem.

The algorithm we describe is motivated by some considerations on the Best Fit algorithm. Given that the worst case performance ratio of on-line algorithms is quite low, from 1.7 for Best Fit to 2 for Next Fit ([5], [6]), there is not much room for algorithms which perform dramatically better than these. When we consider a time-constrained problem, however, where algorithms are under time pressure to produce the best possible packing, then we can consider algorithms which have comparable or slightly improved performance than Best Fit (BF), but which perform at speeds nearer to that of Next Fit (NF). The inspiration for our algorithm is a characteristic of the way in which Best Fit treats bins in its interim solutions. Bins below a certain level can be thought of as "in progress," that is, actively being used by the algorithm to pack new blocks. Above a certain bin level, however, the flow of incoming bins is enough to make the algorithm unlikely to work any more with a particular bin at that level. Furthermore, the number of "in progress" bins remains quite constant even for very large problems. We have designed an algorithm which operates in linear time (as Next Fit), but which uses approximately those bins and blocks which Best Fit would use, and whose performance is thus very close to that of Best Fit.

2 Match Fit Algorithm

The operation of the algorithm (see Fig. 1) maintains in memory those bins which are actively involved in the solution of the problem. It does this by limiting the size of its "short-term memory" according to parameter specification, and taking full bins out of this short-term memory to keep within that bound. We have introduced a memory for blocks, as well. The algorithm matches blocks and bins from its memory, which is usually quite small compared to the problem as a whole. The criteria used for matching produces bins which are nearly full (and will not be re-examined). Thus, the algorithm takes advantage of the relatively

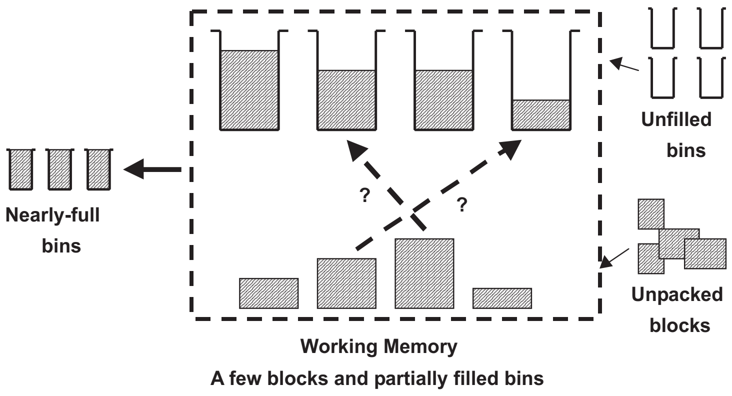

Fig. 1. The Match Fit algorithm illustrated. The algorithm maintains a memory of bins and blocks. On each cycle it attempts to find matches between bin/block pairs which are within a small threshold of making a full bin. When it finds such a match, it removes the nearly-full bin from its memory and refreshes its memory from the queue of waiting blocks and/or new, empty bins. If no such match is found, blocks are put into bins in which they don't match exactly, but the bin is not removed from memory.

small number of bins in the ”in progress” category for excellent time-performance even for large problems (the algorithm is $O(n)$) with a relatively small ($O(1)$) memory. After each iteration, the memory is replenished from any blocks left unpacked. The bin memory is replenished with empty bins. If no suitable matches in memory can be found, the algorithm forces placement of blocks into bins anyway, and then replenishes memory.

We have run Match Fit on the Falkenauer [7] test sets and compared its performance to Best Fit (Table 1).

Table 1. Falkenauer data test comparison results

	Best Fit	**Match Fit** 3 Bins, 3 Blocks	**Match Fit** 10 Blocks, 6 Bins	**Match Fit** unlimited memory
Performance Ratio (mean)	0.96	0.90	0.96	> 0.99

Statistically, Match Fit (MF) can outperform the Best Fit algorithm. For working memory sizes of only 10 blocks and 6 bins, or of 6 blocks and 8 bins, the average performance of Match Fit was 0.96, which is what Best Fit performs on the test set. For this test, the 1000 block probems were used, which is composed of problems with a uniform normalized block distribution on integers in [20,100] with bin size 150. The working memory size, then, is about 2% of the problem size for performance at parity with Best Fit. For very large working memory sizes (comparable to the size of the problem), Match Fit very often yeilds optimal solutions (which are known for these test problems), with an average performance of 0.995.

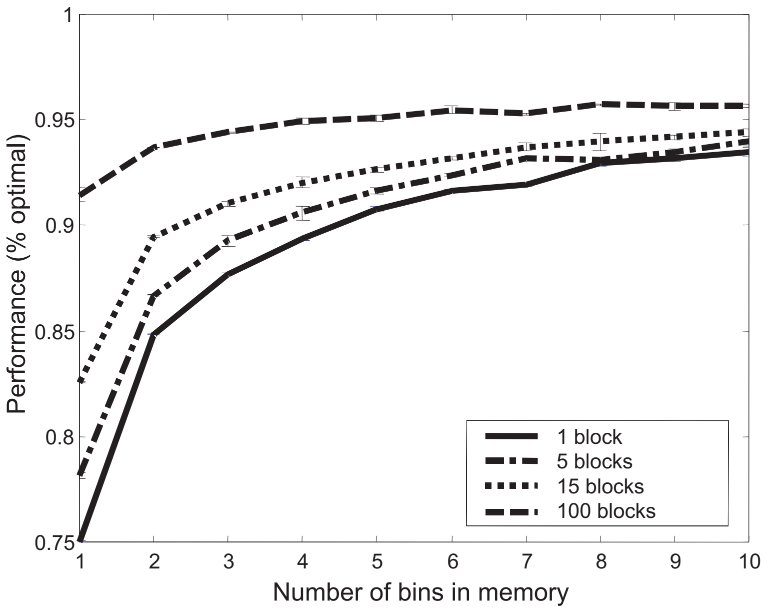

Fig. 2. Performance of MF algorithm on large test problem. 10,000 blocks; uniform distribution $(0, 1]$ on size. The bins are of size unity. The performance increases as the working memory of the algorithm increases slowly towards the performance of Best Fit (which is 98%).

In Fig. 2, the algorithms performance on a very large problem (10,000 blocks; uniformly distributed in size) is shown. The early saturation of performance suggests that Match Fit will do well under time pressure. The reason is that since the MF algorithm takes a shorter time to operate, and still can produce competitive performance with BF, then when there is not much time to operate, the MF algorithm will be able to keep up and perform better than BF. The saturation is comparable when either bin or block memory is increased. Of the two, increasing block memory offers slightly better marginal performance. This suggests that bin packing algorithms which operate in very resource-limited environments would do well to expand the number of blocks they consider simultaneously alongside, or even before, they expand the number of partially-filled bins they consider.

3 Time-Pressured Performance Characteristics

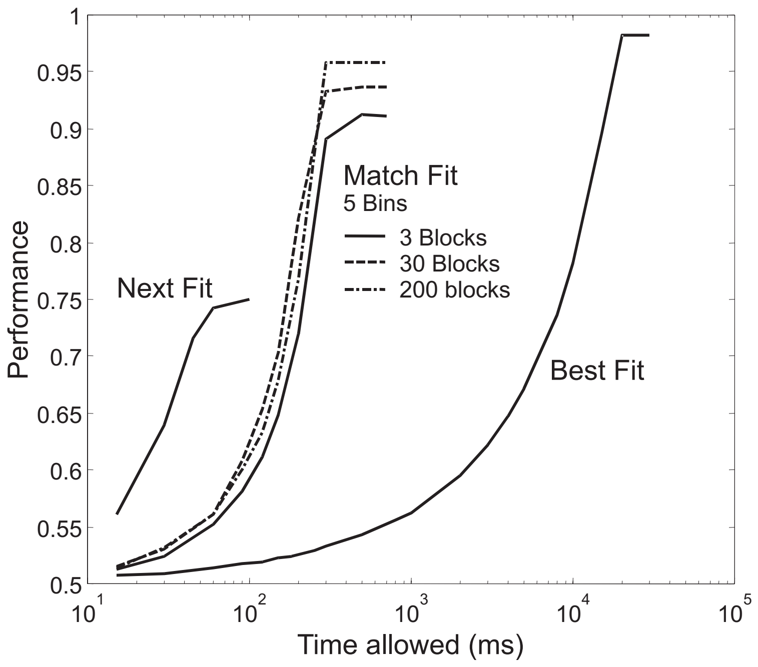

Fig. 3. Comparison of time-pressured performance of three bin-packing algorithms. The performance of Match-Fit is intermediate to Next Fit and Best Fit. The time allowed for the problem solution is shown in milliseconds, but will scale if a different processor is used for the problem, while maintaining the general shape of the curves. The problem is packing 10,000 blocks uniformly distributed in size over $(0, 1]$ into unity-sized bins.

Our primary focus in this investigation is the performance under time pressure of these various algorithms. By time pressure we mean the performance of the algorithm in a situation where the best solution is demanded from the algorithm after a particular length of time. The time constraint is enforced by an external controller, which allows the algorithms to run on a problem for a fixed amount of time, and then allocates any remaining blocks unpacked by the algorithm at one block per bin. This is equivalent to blocks passing a real-life packing machine operated by one of these algorithms. If the algorithm could not consider a particular block as it passed (that is, if the blocks passed too quickly) then that block would pass outside the working area of the machine and be placed into its own bin. Fig. 3 shows a performance comparison between the BF, NF, and a few configurations of the MF algorithm (with 5 bins and a varying numbers of blocks available to its working memory). The problems being solved by the algorithms are the packing of the 10,000-block problem (uniform (0,1] distribution on block size) into bins of size unity. As can be seen, the performance of the MF algorithm is intermediate to BF and NF performance.

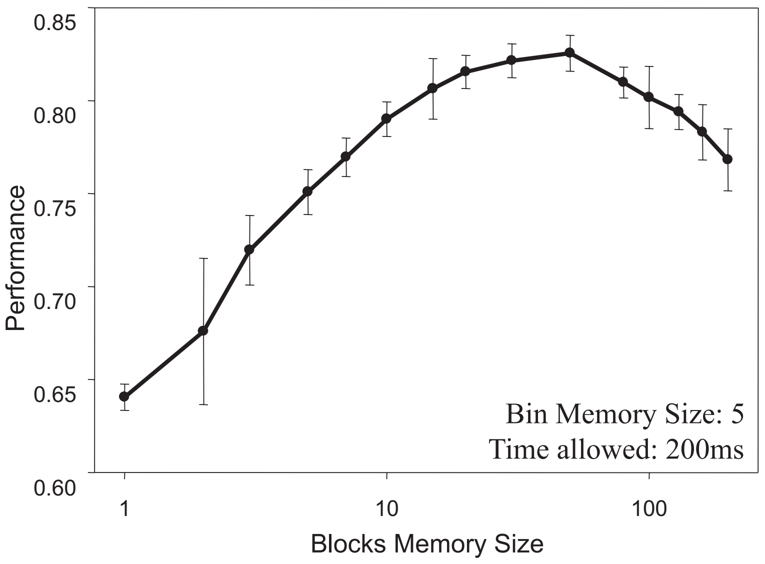

Fig. 4. Optimality in memory size for time-pressured MF algorithm. Under time pressure, there is a performance optimum for the MF algorithm. Too many items in memory slows the algorithm down too much, whereas with too few items, it does not perform as well. The problem is packing 10,000 blocks uniformly distributed in size over $(0, 1]$ into unity-sized bins.

An examination of the performance characteristics for the MF algorithm indicates that when the algorithm is in its most interesting performance region in terms of its time-pressured performance–that is, performing well, but not yet at its optimum where it would be best to choose the most possible memory–there is an optimum in the amount of working memory the algorithm uses. This is shown more explicitly in Fig. 4. The optimal for this case (with 5 bins) is about 25-30 blocks in memory. When fewer blocks are used, the performance is less because the algorithm doesn't have as good a chance of finding good packings for blocks. When the memory uses more blocks, the performance also decreases, because although good packings are being found, it takes the algorithm longer to find them and runs out of time. In the case of the BF algorithm, which is equivalent to a limiting case of the MF algorithm where an almost unlimited bin memory is allowed (but using a single block in memory at a time), the packing is superior, but the time taken is roughly two orders of magnitude more.

There is another interesting effect in how the optimal strategy changes as the time pressure eases. Fig. 5 illustrates that there is a quite abrupt change in the optimal approach to solving the problem as the time pressure is slowly varied. In this figure, the memory parameters of the algorithm were varied widely (from 1 bin and 1 block to 200 blocks in memory and 20 bins in memory). For each value of time pressure, the various Match Fit algorithms using their memory parameters were run, and the best performer was examined. The top plot of Fig. 5 shows the best performance of any of the Match Fit algorithm configurations in solving the problem. The bottom plot shows the working memory size (the addition of blocks and bins in memory) at this optimal point. We observe a sharp threshold in the parameter space of the optimal configuration. Below this threshold, the optimal approach to solving the problem is for the algorithm to use a small working memory to best advantage. This remains true as the time pressure eases off and the algorithm is able to perform better and better. When the performance becomes close to its asymptotic limit, however, there is a transition. For time pressures less than this transitional value, the algorithm is better off to use basically as much memory as is available to it (the saturation in working memory size shows reflects the maximum sizes used in the simulations). Before the threshold, the performance curves exhibit the clear optimum we anticipate for a system solving a demanding problem in real-time: there is an optimum in the amount of resources it should dedicate to the task. As the time pressure eases off, this optimum becomes less pronounced, and the approaches which use more resources start to become attractive.

4 Conclusions

The Match Fit algorithm provides an interpolation between two of the most interesting on-line approaches to solving the bin packing problem. The bin packing problem has the charactistic, shared with many difficult problems, that only part of the problem is important at any one time, and neglecting much of the problem state in the functioning of the algorithm need not lead to great performance deficits. We then devised an algorithm which would take advantage of this idea by utilizing a user-defined amount of computational resources in its solution of the problem.

Since we are able to vary the size of the memory available to the MF algorithm smoothly, we can observe the impacts of strategies which are resource-intensive and those which do not require so many resources. When not under time pressure, we observe asymptotic performance behavior with respect to the amount of resources used. When the algorithm is under time pressure, however, there is an optimum in the computational resources used by the algorithm. This corresponds to the difference between the usual computer and human computational strengths. When there is less time pressure, and when exact solutions are wanted, the approach which uses the most information about the problem is favored. Under conditions where time pressures are important, it is best to

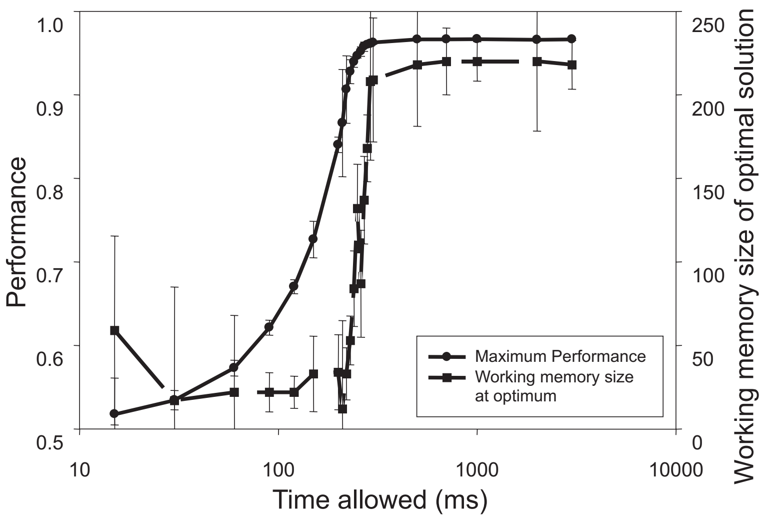

Fig. 5. The performance of MF as a funtion of time pressure and the corresponding sudden change in strategy needed to achieve optimal performance. For time pressures such that the algorithm cannot perform at its asymptotic level, the optimal strategy is to use a relatively small working memory. The transition between this regime-that where it is optimal to use a very small working memory and that where it is optimal to use a very large working memory-is extremely sharp. The error bars in the top plot show the standard deviation in performance of the memory configuration with the highest mean performance over 10 runs of the simulation. The error bars in the bottom plot indicate the standard deviations for those values of working memory size for which the performance at a given time pressure was ever the best in any simulation run, and so are quite pessimistic.

severely restrict the amount of information considered by the planning parts of the algorithm.

Under increasing time pressure, we observe that there is a very sharp threshold between the optimal performance of the two approaches. When time pressure is severe-that is, there is not enough time to quite get to near-asymptotic performance, it is advantageous to select a strategy which uses very few computational resources. When the time pressure is not so severe, it very quickly becomes advantageous to use very large (relative to the previous case) amounts of computational resources. The reason for this is that under intense time pressure, the system switches from performance being asymptotic in the amount of resources used to having an optimum. Instead of providing a performance boost, having more resources available simply "distracts" the algorithm and slows it down as it has to take time to take the extra information into account.

These two lessons–an optimum in the performance vs. resource utilization curve when the algorithm operates in time constrained environment, and the sudden transition from a limited-memory optimum to a large-memory optimum–we believe are extensible to a wide variety of computationally interesting problems. The bin packing problem shares with many other interesting problems the characteristic that it is very hard to solve exactly, but relatively easy to get close. When there is this smooth measure on performance (instead of the more binary case of, say, the k-SAT [8] problem), we expect to observe these two phenomena in real-time algorithms.

References

1. Izumi, T., Yokomaru, T., Takahashi, A., Kajitani, Y.: Computational complexity analysis of Set-Bin-Packing problem. IEICE Transactions on Fundamentals Of Electronics Communications and Computer Sciences: **5** (1998) 842–849
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA. (1979)
3. Karp, R.M.: Reducibility Among Combinatorial Problems. In Complexity of Computer Computations, R.E. Miller and J.W. Thatcher eds. Plenum Press, NY. 1972 85–104
4. Johnson, D.S.: Fast Algorithms for Bin-Packing. Journal of Computer Systems Science **8** (1974) 272–314
5. Johnson, D.S., Demers, A., Ullman, J.D., Garey, M.R., Graham,R.L.: Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. SIAM Journal of Computing **3** (1974) 299–326
6. Mao, W.: Tight Worst-case Performance Bounds for Next-k-Fit Bin Packing. SIAM Journal on Computing **22(1)** (1993) 46–56
7. Falkenauer, E.: A Hybrid Grouping Genetic Algorithm for Bin Packing. Working paper CRIF Industrial Management and Automation, CP 106 - P4, 50 av. F.D.Roosevelt, B-1050 Brussels, Belgium. (1996)
8. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity from characteristic 'phase transitions'. Nature **400(6740)** (1999) 133–137