Values of the merging function and algorithm design as a game

Warren D. Smith*
Kevin J. Lang
{wds,kevin}@research.NJ.NEC.COM

June 29, 1994

Abstract — **Finding algorithms that are optimal for worst case input is equivalent to solving a 2-player game of perfect information. The *merging game* arises when finding the minimum number $M(m,n)$ of comparisons needed to merge two sorted lists of cardinalities m and n. We devised a computer program capable of determining any value of $M(m,n) \leq 20$. It was based on standard game solving techniques (alpha-beta search, transposition table) plus bounding, "fuzzy trans table," and "partitioning" ideas specific to the merging game. Thanks to a new idea called "iterative widening," our game solver can also determine strong upper bounds (but not lower bounds) on $M(m,n)$ in a much larger region. The widening/shrinking idea may have other applications in computer game playing.**

We present a large table of bounds on $M(m,n)$ which were obtained by combining our new results with previous bounds on $M(m,n)$ (which we survey).

Keywords — Computer game-playing, optimal algorithms, minimum comparison merging, iterative widening, algorithm design as a game.

1 Introduction

1.1 Low rent plan

Those uninterested in merging and algorithm design, but who are interested in an idea for speeding up computer chess, can skip directly to §5.2.

1.2 Games and algorithm design

The problem of finding the best algorithm in the sense of having minimal cost for worst case input is a 2-player minimax game. The "Min" player is choosing the algorithm to minimize the cost and the "Max" player is choosing the input to maximize the cost. In the case of "on-line" algorithms, in which the algorithm has to read a character of the input, then make some computation, then read the next character, and so on, the game is rather similar to games such as chess, since there are alternating moves by the Min player (choosing the algorithm's next "move") and the max player (choosing the next batch of input bits to feed to the algorithm) and the final cost only becomes apparent at the end of the game, which consists of a large number of simple moves. The game may be solved via "alpha beta search" [10], which is a considerably more efficient method for solving such games (due to "cutoffs") than naive methods.

Perhaps the canonical algorithm problem is sorting. If the "cost" of a sorting algorithm is the number of pairwise {=,<,>} comparisons between elements it makes (all other computation being regarded as having zero "cost"), and it is *only* allowed to use comparisons, then sorting may be viewed as an "on-line" algorithm, where the "input characters" are the outcomes of the comparisons called for by the algorithm.

In this paper we will concern ourselves with the related problem of merging. We wrote a game-solving program that finds min-comparison list merging algorithms via alpha beta search. But before we specialize the discussion to merging, a little more on games and algorithms.

The problem of finding the best *randomized* algorithm to minimize expected cost for worst case input is a matrix game in the sense of Von Neumann. Von Neumann's duality theorem ([3] §13-2 page 287) for matrix games

*NEC, 4 Independence Way, Princeton, NJ 08540

tells us that instead of seeking the worst case input for the best randomized algorithm, we can equivalently seek the deterministic algorithm with the least expected cost for the worst (randomized) distribution of inputs. This was observed by Yao [17]. Again the games are much more interesting for on-line algorithms. These games are much harder to solve than our kind of games [11].

The problem of finding the least expected cost algorithm where input is selected from a fixed, known, probability distribution, and the problem of finding the worst case of a specified algorithm, are 1-player games. One-player games are much harder to solve than 2-player games because one cannot take advantage of "alpha-beta cutoffs."

Finally, we now describe a general formalism for finding the most "competitive" on-line algorithms. In this scenario, an on-line algorithm A is regarded as being in competition with an algorithm B which gets to see the entire sequence of inputs to A, ahead of time, and is not constrained (as A is) to read chunks of input and do partial computations alternately. The "competitive ratio" of A and B for input I is the cost of $A(I)$ divided by the cost of $B(I)$. A is said to have optimal competitive properties if $\max_I \max_B \mathrm{cost}(A(I))/\mathrm{cost}(B(I))$ is minimal over all choices of A.

Now to make the competitive scenario more explicit, we have some model of our computer which operates in a space S of possible states $s \in S$. The cost of a transition between two states $s \in S$ and $t \in S$ is $C(s,t) \geq 0$ where C is some given transition-cost matrix. Each input character is selected from an alphabet Z. The on-line constraint states that, after A (or B) reads an input chunk $z \in Z$, it is required to be in a state $s \in S_z \subset S$, where the subsets S_z of S are given and depend only on z. The total cost of the algorithm on input I is the sum of the state transition costs it must make, but algorithm B , which knows the whole input sequence in advance, may be able to choose a less costly sequence of states and transitions than algorithm A. Also, the initial state may be either forced to be the same for both A and B, or (in a different version of the problem) B gets to choose the state it wishes to start from.

It is possible to find, for any given input sequence I, the least possible cost for algorithm B on that input, by "dynamic programming" in time $O((\max_z |S_z|)^2 \cdot |I|)$ and bookkeeping space $O(|S|)$. One simply constructs a table specifying the least cost to end up in state s after reading the ith input character, and then the table for $i+1$ may be constructed from the table for i in time $O((\max_z |S_z|)^2)$. With a procedure for finding $\mathrm{cost}(B(I))$ as subroutine, one then may find the best possible competitive ratio for algorithm A on inputs of a specified length ℓ by solving a minimax game. The Min player chooses the next transition to make (from the set of transitions compatible with the previous chunk of input), the Max player chooses the next input chunk. At the end of the game, the "leaf evaluation" of the competitive ratio is made with the aid of the subroutine we just mentioned. This game has length ℓ moves and ℓ replies, and has branching factor $\leq \max_z |S_z|$ at Min moves and $|Z|$ at Max moves. Optimally competitive *memoryless* on line algorithms A could also be found using the same sort of game solving approach.

1.3 The merging game

Consider merging two sorted lists $A_1 < A_2 < .. < A_m$ and $B_1 < B_2 < .. < B_n$, $m \leq n$, to create a sorted list with $m+n$ elements. We wish to devise a deterministic algorithm which will accomplish this after making the minimum possible number $M(m,n)$ of comparisons between elements of A and B.

The "merging game" is as follows. There are two players, "Max" and "Min" who move alternately. Min starts the game by choosing a comparison "a:b." Max replies by choosing the result of this comparison, that is, $A_a < B_b$, or $A_a > B_b$. The game is over when the results of all the comparisons made so far suffice to determine a unique sorted order for the whole $(m+n)$-element set. The score of the game is the number of comparisons that were used, and Max is trying to maximize, and Min to minimize, this score.

Of course, Min, being a sensible fellow, never compares two elements of A or two elements of B, since the result of any such comparison is already known. Min only compares elements A_a with elements of B_b in cases where the result of the comparison is *not* already known to (or deducible by) him. As a consequence Max need not worry about making her answers consistent, since any answers she makes will automatically be consistent.

The game-theoretic value of the merging game is exactly $M(m,n)$. Incidentally, for convenience later we define $M(x,y) = 0$ if $x \leq 0$ or $y \leq 0$.

As a consequence, standard game-solving methods (alphabeta search with transposition table) can be used to determine values of $M(m,n)$. "Alpha beta cutoffs" can cause dramatic cost savings compared to other, more naive, computer search methods written by people who did not view the problem as a game. More on this in §3.

The same idea is applicable to other optimum-algorithms problems besides merging. The most obvious one to try next is minimum-comparison sorting, see §5.3.

1.4 Asymmetry and iterative widening

When using the game solver, one quickly notices that it takes considerably longer to establish a new lower bound on $M(m,n)$ than to establish a new upper bound. This is because, to establish a new lower bound, it must search every possible first move by Min and disprove them all. But finding a new upper bound usually may be accomplished after searching only a few of Min's possible first moves.

The discrepancy is particularly large because the merging game is very asymmetric: at any given turn Min usually has a large number of moves (of order mn) to choose from, whereas Max has at most 2 choices. Unfortunately the alphabeta algorithm does not work as well for solving such unbalanced games, as it works for solving more balanced games. All this suggests a new idea which we call "iterative widening." In the "width-W restricted merging game," Min is only allowed to choose from the first (according to some good ordering) W of his candidate moves at each turn. The value of the width-W restricted merging game is of course an upper bound on the value of the unrestricted merging game. This upper bound gets tighter as W is increased. In our experience so far, W has never needed to get larger than 5 before the restricted[1] and unrestricted games had the same value. Of course, it is *much* easier to solve the restricted game for small W than it is to solve the unrestricted game. We've seen cost saving factors of $\approx$ 5000. This is due both to the dramatic reduction in the branching factor at Min nodes, and also to the better balance.

This leads to our "iterative widening" idea, in which one successively solves the restricted-width games for $W = 1, 2, 3, \ldots$. In the event that (due to limited resources) one stops the process before W has reached full width, one has still managed to get a valid upper bound on $M(m, n)$.

This idea may also be applicable in other game-playing contexts and in other ways. See §5.2.

1.5 Other methods

Since solving games is laborious, one prefers to avoid it whenever possible. In many cases upper bounds (from known merging algorithms, and known ways to combine smaller merging algorithms) and lower bounds (from known counterstrategies and known ways to combine smaller counterstrategies) are available. In the event that these lower and upper bounds meet, of course no further work is needed. If they do not meet, then they still may be used to reduce one's game-solving effort, because one may start the game solver with a small "alphabeta window." In the other direction, results from the game solver for some $M(m, n)$ can be plugged into the known bounds-combining methods to yield better upper and/or lower bounds for other (m, n).

As the result of the symbiosis of all these ideas, we present, in the next section, a large table of upper and lower bounds on $M(m, n)$.

2 Table of bounds on the merging function

Let $T_m(k)$ denote the maximal n such that a m-element chain can be merged with a n-element chain in at most k comparisons. It is sometimes more convenient to express results in terms of this inverse function. As a consequence of the present paper, $T_m(k)$ is now known for $k \leq 20$ (and all m) with the sole exception of $T_6(20) = 21$ or 22 (it is almost certainly 21).

(INSERT TABLES OF $T_m(k)$ AND $M(m, n)$ HERE. SEE SEPARATE SHEETS.)

Key:

`a` Adjacency, e.g. (EQ 19).
`b` Binary insertion, (EQ 4).
`c` Christen's method 0, (EQ 7).
`d` Recurrence relation (EQ 14).
`e` Even/Odd, (EQ 17).
`g` Result from the Game Solver, see §3.12.
`h` Hwang, Theorem 2.
`i` Information-theoretic lower bound (EQ 1).
`j` Christen, (EQ 11).
`k` Knuth, Theorem 7.
`m` Mönting, Theorem 3.
`s` Splitting, (EQ 20).
`w` Graham and Hwang/Lin, Theorem 1.
`z` Straight merge, (EQ 5).

Capitalized letters have the same meaning as lowercase letters, except that they depend on feeding some new result of the Game Solver into the recurrence; for `K` see the discussion after theorem 7 in §4.3.

3 Solving the merging game

3.1 Board positions and how to make moves

The sum total of everything that Min has learned so far (as a consequence of all the comparisons he has made up to that point) can be expressed in two m-element arrays $L_{1..m}$ and $U_{1..m}$. Specifically, L_a is the greatest number, and U_a is the least number, such that Min knows that $B_{L_a} < A_a < B_{U_a}$, for each $a \in \{1..m\}$. If no lower bound on some A_a is known then $L_a = 0$, and if no upper bound on some A_a is known then $U_a = n + 1$. Otherwise $1 \leq L_a < U_a \leq n$. Of course $L_a \leq L_{a+1}$ and $U_a \leq U_{a+1}$.

These two arrays plus the "current score" (number of comparisons Min has made so far) describe (to use chess playing terminology) the "current board position" at a Min node. Min then selects a "move" $a{:}b$, where $L_a < b < U_a$ and, depending on Max's response, the value of L_a or U_a is altered to become b and the current score is incremented. (Of course, this alteration may improve some other values of L_j or U_j.) This shows us how to make moves in the merging game. The game is over when no moves are available for Min, that is, when $L_a + 1 = U_a$ for all $a \in \{1..m\}$.

3.2 The game tree and alpha beta search

The "game tree" is the tree whose nodes are the board positions, whose directed (from parent to child) edges are the moves, and whose root node is the initial position (in which the current score is zero and $L_a \equiv 0$, $U_a \equiv n + 1$). The game theoretic value of a leaf node in the game tree is its current score, and the game theoretic values of the interior nodes of the game tree are defined according to:

- The value of a Min node is the min of the values of its children.

[1] Well, partially restricted. See §3.10.

- The value of a Max node is the max of the values of its children.

"Solving a game" means finding the value of the root node, i.e. the value of the game.

The naive method for solving a game would be to generate the entire game tree in root-downwards pass, and then find the values of all the nodes in a leaves-upwards pass. The alpha-beta method [10] is functionally equivalent to the naive method (that is, it will always produce the same result for the root's value) but is usually much faster, and also is more flexible. The alpha-beta algorithm, in its barest form, is defined recursively as follows[2].

```
MaxSolve(Alpha, Beta)
{
   t = -infinity;
   Moves[] = FindMaxMoves();
   for(mv in Moves[]){
      MakeMove(mv);
      s = MinSolve(max(Alpha,t), Beta);
      UnMakeMove(mv);
      if(s>t){ t=s; if(t>=Beta) goto DONE; }
   }
DONE:
   return(t);
}

MinSolve(Alpha, Beta)
{
   Moves[] = FindMinMoves();
   if( Moves[] is empty ) return(Current Score);
   else t = infinity;
   for(mv in Moves[]){
      MakeMove(mv);
      s = MaxSolve(Alpha, min(Beta,t));
      UnMakeMove(mv);
      if(s<t){ t=s; if(t<=Alpha) goto DONE; }
   }
DONE:
   return(t);
}
```

An initial call to `MinSolve`($-\infty$, ∞), assuming that the current board position is the starting position, will solve the game and return its value. More generally, if the true value t of the game obeys $t \geq \alpha$, then an initial call to `MinSolve`(α, β), will return a lower bound on t. If the true value t of the game obeys $t \leq \beta$, then an initial call to `MinSolve`(α, β), will return an upper bound on t. In particular, if $\alpha \leq t \leq \beta$, then t is returned. These facts allow the alphabeta algorithm with reduced windows to obtain bounds on the game theoretic value without actually finding it.

The `goto` statements in the algorithm are called "cutoffs" and represent speedups over the naive algorithm. If the `FindMoves` routine generates moves in a nearly best-first order, then these speedups can be significant. Specifically, suppose the branching factor at Min nodes were always U and the branching factor at Max nodes were always V, and the leafs of the tree were always $2D$ deep. Then the naive game solving procedure (or, for that matter, alphabeta with pessimal move ordering) would examine $U^D V^D$ leafs. Alphabeta with best-first move order and a doubly infinite initial window would examine only $V^D + U^D - U^{D-1}$ leafs. (A multiplication has been replaced by an addition!) With a finite initial window, alphabeta would examine even fewer leafs. This savings is most significant when $U \approx V$ ("balanced" games) when the work is roughly square-rooted, but is still exponentially large even for the highly unbalanced merging game where $U \gg V = 2$.

The move ordering heuristic that we will describe is empirically fairly good, so that the best case formula is fairly close to the truth.

3.3 Move ordering and upper and lower bounds

A certain number $P(\theta)$ of the $(m+n)!$ permutations of the $(m+n)$ elements are compatible with any given board position θ. For example, the number of permutations compatible with the *initial* board position θ_0 is $P(\theta_0) = \binom{m+n}{m}$. Now the number of additional comparisons that Min must make to complete the merging[3], starting from θ, is at least $\lceil \lg P(\theta) \rceil$. In the event that this lower bound (plus the current score) is greater than or equal to β, we may perform a beta cutoff without searching any descendants of node θ.

In the case when $\theta = \theta_0$, the lower bound degenerates to the well known "information theoretic lower bound on merging"

$$M(m,n) \geq \lceil \lg \binom{m+n}{m} \rceil. \tag{1}$$

Now although in general counting the number of permutations compatible with a partial order is #P-hard, there is a quadratic-time algorithm for counting the permutations compatible with the partial orders that arise inside merging algorithms. Specifically:

```
CountPermutations(L[], U[], m, n)
{
   Start with q[1..m][0..n] = 0;

   for(i=L[m]; i<U[m]; i++) q[m][i] = 1;

   for(j=m-1; j>=0; j--){
      s = 0;
      for(i=U[j+1]-1; i>=L[max(j,1)]; i--){
         s += q[j+1][i];
         if(i<U[j] and 0<j) q[j][i] = s;
      }
   }
```

[2]All algorithms in this paper are expressed in a pseudocode resembling the C language.

[3]Throughout this paper lg means $\log_2$.

```
   return(s);
}
```

In this algorithm, `q[j][i]` is the number of placements of $A_{j+1}..A_m$ among the B's compatible with the L's and U's, and the extra requirement that $B_i < A_j < B_{i+1}$. The algorithm computes the `q[j][i]` recursively and returns $\sum_i \mathtt{q}[1][\mathtt{i}]$. The reader may enjoy checking that in the initial position this recurrence reduces to the Pascal triangle recurrence for the binomial coefficients so that $\binom{m+n}{n}$ is returned. Of course the storage requirement can be reduced from order mn to order n since only two rows of the `q` table are ever in use at any time.

`CountPermutations` was used in the game solver both to compute lower bounds, and also to order Min and Max's moves. Incidentally, if $P(\theta) = 2$, then exactly 1 more comparison is needed, and will suffice, to complete the merging, a fact which allows us to chop two ply off the search tree.

It is also possible to compute a powerful *upper* bound on the number of comparisons that will be needed to complete the merging.

```
UpperBound(L[], U[], m, n)
{
   q[1..m][1..m] = infinity;
   for(i=1; i<=m; i++){
      q[i][i] = M( 1, U[i]-L[i]-1 );
   }

   for(j=1; j<m; j++){
      for(i=1; i<=m-j; i++){
         s = M( j+1, U[i+j]-L[i]-1 );
         for(k=0; k<j; k++){
            s = min(s, q[i][i+k]+q[i+k+1][i+j]);
         }
         q[i][i+j] = s;
      }
   }
   return( q[1][m] );
}
```

The idea here is that certainly, $\sum_i M(1, U_i - L_i - 1)$ is an upper bound; simply insert each element A_i within its interval of uncertainty by binary insertion. More cleverly, $A_i..A_{i+j}$ may be inserted into their combined interval of uncertainty in $\le M(j+1, U_{i+j} - L_i - 1)$ comparisons. The `UpperBound` procedure above uses "dynamic programming" to find the best upper bound that results from any combination of such observations.

In fact, in the game solver a slightly stronger upper bound than this was used, which arises from inserting the statements

```
if(j==1)
  s = min(s,
   M2[U[i+1]-L[i]-1][L[i+1]-L[i]][U[i]-L[i]]);
```

just before the `for(k` loop. Here, the function `M2[n][b][c]` represents the number of comparisons needed to merge 2 elements A_1 and A_2, where $A_1 < A_2$, with n elements $B_1..B_n$, assuming that it is known that $A_1 < B_c$ and $B_b < A_2$. ($c = n+1$ or $b = 0$ if no upper, respectively lower, bound is known. If $c > n+1$ or $b < 0$ it is understood that one should truncate to $c = n+1$ and $b = 0$.) This function is readily precomputed recursively as follows:

If $c \le b+1$, then `M2[a][b][c]`$= M(1, c-1) + M(1, a-b)$. Otherwise,

$$\mathtt{M2[a][b][c]} = 1 + \min\Big(\min_{1 \le x < c} \max\{\mathtt{M2[a-x][b-x][c-x]},\ \mathtt{M2[a][b][x]}\}, \min_{b < x \le a} \max\{\mathtt{M2[x][b][c]},\ \mathtt{M2[a][x][c]}\}\Big). \qquad (2)$$

In particular, $M(2, a) = \mathtt{M2[a][0][a+1]}$. Graham [5] (see also [9]) has found a closed form for the `M2` function, although it is sufficiently complicated that we prefer simply solving the recurrences above.

In the event that, at any Max node in the search, this upper bound is $\le \alpha$, then an alpha cutoff may be taken without any further search.

Also, this `UpperBound`, as improved above, is in fact *exact* if at most two of the intervals $[L_i, U_i]$ have non-minimal width. This observation can cut a substantial number of plies off the search tree, in many situations.

The above three ingredients, namely, alphabeta search, the lower bound based on `CountPermutations`, and the dynamic programming `UpperBound`, suffice to construct a simple, but still fairly powerful, mergegame solver[4]. We will now discuss, in less detail, some more complicated ideas that increase the power of the solver.

3.4 Another lower bound

Knuth's lower bound (see theorem 7, but do not utilize its last sentence) may be used at the nodes that are exactly 1 ply deep (i.e. the children of the root of the search tree). This can sometimes rule out almost every possible first move immediately, but in some cases, such as $M(10, 18)$, it has no effect whatsoever.

3.5 Partitioning

In the event that, at some point during the search, there arises some a, $1 \le a < m$, so that $U_a \le L_{a+1} + 1$, then the merging game may be partitioned into two disjoint smaller merging problems, namely, $A_1..A_a$ is to be merged with $B_1..B_{U_a - 1}$ with the aid of the restrictions implied by $L_1..L_a$ and $U_1..U_a$, and similarly $A_{a+1}..A_m$ is to be merged with the remaining B's. The value of the remaining part of the merging game is the sum of the values of the subgames. The work in solving these two smaller subgames is much smaller than the work involved in solving the entire game (without recognizing the partition), since again, informally speaking, a "multiplication

[4]Powerful enough for a skeptical reader to rederive some of our results.

has been replaced by an addition." To take maximal advantage of the alphabeta windows in subgames, notice that each successive subgame must (in order to be relevant) have value within a window defined by the sum S of the previously solved subgames, α, β, and a priori upper and lower bounds u and ℓ on the sum of the remaining subgames. Specifically, this window is

$$[\alpha - S - u, \;\; \beta - S - \ell]. \tag{3}$$

Just using infinite windows in the subgames is still quite effective, though.

Our experience has been that partitioning speeds up the basic game solver by a factor of 3-10.

3.6 *Transposition table*

In the event that the current board position has been solved (or partially solved) previously in the search, so that its game-theoretic value is known (or bounds on that value are known) then we can save work by *not* searching the position's descendants again, but instead merely retrieving the relevant information from a table. (Incomplete solutions may still be powerful enough as bounds to cause alpha or beta cutoffs.)

Our transposition table recognizes positions that are isomorphic under the operation of reversing the sense of all comparisons, which is a factor of 2 economization. It also recognizes translations of the L_i and U_i by constants.

Our experience has been that a primitive transposition table speeds up the game solver by an additional factor of at least 1.2, and sometimes by more than 10. (The factor depends on how much information has accumulated in the table in previous runs of the program.)

3.7 *Fuzzy Transposition table*

In the event that some entry θ in the transposition table had intervals $[L_i, U_i]$ which were subintervals of the intervals in a position θ', then the lower bound in the transposition table would be valid for θ'. (The same is true with "sub" replaced by "super" and "lower" by "upper.") We only considered alterations $\theta \to \theta'$ that involved widening or shrinking a single interval by 1. This speeds it up by an additional factor of 5-10.

3.8 *Other trans table ideas*

A more sophisticated way to use transposition tables would involve a system of "priority rules" intended to cause "useful" information to accumulate in the table and "useless" information to gradually disappear from it. Also, the table could remember the next two allegedly "best moves" from that position, which would speed up re-searching partially solved positions, and this could be combined with "iterative deepening."

We did not use any of these more sophisticated ideas, which would surely have resulted in additional speedups.

3.9 *Windows*

By using the game solver with α initially set to the best known lower bound on $M(m, n)$, and β set to the best known upper bound, further speedups result.

Another useful fact is that WLOG the first (i.e. root) comparison $a{:}b$ in the merging algorithm involves a b with $2b \le n+1$, due to the possibility of an overall sense-reversal. (Also if $2b = n+1$ then WLOG $2a \le m+1$.) This saves a factor of 2.

Due to previously mentioned bounding properties of alphabeta search, one could also use the game solver in a "speculative mode" by choosing the initial values of α or β more optimistically than was strictly justifiable. In that case the game solver would confirm or refute your speculation, hopefully quickly. In practice this idea is not very useful since the known bounds are usually very close anyway. One can also attempt to get upper bounds quickly by choosing the first few comparisons in Min's strategy by hand. This idea has also not been very useful because the present authors do not have enough wisdom to choose the comparisons better than the program itself, but it might be useful to somebody with more wisdom or better intuition than us.

3.10 *Iterative Widening*

This idea was described in the introduction and can result in tremendous speedups when seeking upper bounds on $M(m, n)$. Unfortunately it is useless when seeking lower bounds[5].

Actually, strictly speaking, our game solver, with width restriction W turned on, is not solving the restricted width-W merge game, because every time a cutoff occurs due to `Upperbound`, the part of the tree that was cut off may well have involved a less restricted Min player. Also, partitioning increases the effective power of restricted width Min players tremendously, and also the gamesolver doesn't enforce the width restriction at the root. For these reasons, the game solver, set to width W, can often produce a more powerful upper bound than the true value of the width-W merge game as strictly defined in §1.4.

We now point out one additional trick. Any *upper* bounds on nodes's values found during a restricted-width search and stored in the transposition table, can be reused in later searches with larger widths W, since they are entirely valid even in the unrestricted merging game. Lower bounds, however, have to be erased from the transposition table between widenings.

[5]In principle the same idea in reverse (that is, restricting the power of the Max player, and then noticing that the value of the restricted game is a lower bound on the value of the unrestricted game) would work to find lower bounds, but in practice this is worth little since this just makes the balance worse, does not greatly decrease the size of the game, and does not give a very strong bound.

3.11 Testing

The entire game-solving program (including implementations of previous upper and lower bounds on $M(m,n)$, see the next section) which produced the results in this paper comprises about 30 pages of C code.

We believe it is correct. Some evidence for this is that our program confirmed all the values of $M(m,n)$ for $\max(m,n) \leq 10$ tabulated by Knuth [9] upper left of page 203. Also, we mention that Hwang [6] managed to find the values of $M(3,n)$ in closed form by an extremely complicated argument, see theorem 2. Our program confirmed all Hwang's values up to $n = 82$, which may either be regarded as a confirmation of our program, or of Hwang's analysis, depending on your point of view. Some runs, including this one, were intentionally done with alphabeta windows that were weakened on both sides, in order to provide a tougher test. We have similarly confirmed Mönting's closed form for $M(4,n)$ up to $n = 55$, see theorem 3. The game solver also has confirmed every miscellaneous value of $M(m,n)$ given in the papers we've cited – *except* that, unfortunately, the game solver found that $M(7,12) = 17$, which contradicts two independent (?) statements in the literature [2] [1] that $M(7,12) = 18$. This led to considerable worry on the part of the first author (Smith) that the game solver had a bug, but:

- Neither of these two papers that claimed $M(7,12) = 18$ included a proof. They merely claimed that they *had* a proof, but implied that it was too complicated to include, or even sketch, in the paper.

- The game solver's claim that $M(7,12) = 17$ has been independently confirmed by a program written by the second author, Kevin Lang. (The game solver was written by Smith.) Lang's program is considerably different and simpler than the game-solver. It is written in Scheme instead of C, it does not view the problem as a game, and it does not keep track of partial orders, but rather of lists of permutations, and it was written entirely independently.

- Lang's program has also confirmed the game-solver in several other cases besides $(7,12)$ in which the game solver found a new record upper bound, e.g. $(8,14)$, $(9,15)$, $(7,26)$. Each of these confirmations confirmed every line in the (abbreviated) tree output by the gamesolver [which in the case of $(7,26)$ was over 50 such lines], and was thus quite a powerful check.

- The entire comparison tree for the merging algorithm $M(7,12)$ found by Lang's program, has 41851 comparator nodes, and he has it in electronic form. Anybody who wants it can get it from Lang electronically and check it themselves with a comparatively easy-to-write checking program which simply runs all the $\binom{19}{7} = 50388$ possible permutations through the comparison tree.

- Lang in fact wrote such a checking program and used it to check the tree.

For these reasons we believe that the game solver is correct and that the claims in those two papers are incorrect.

3.12 Results

When the game solver finds a new merging algorithm, it outputs the description of that algorithm as a "comparison tree," *except* that some subtrees are omitted. The subtrees that are omitted correspond to either

- a previously solved position (from the transposition table) denoted by "`T`,"

- The case denoted by "`U`" means that the a priori `UpperBound` will suffice to see that this position is solvable in the required number of comparisons (or fewer)

- if partitioning into independent subproblems occurs, the solution of the subproblems is not output; instead, output is the single letter "`P`."

These abbreviations (especially `T`) can sometimes represent a great deal of information which the unfortunate reader of this paper will probably never see.

(NOW PLEASE CONSULT THE PICTURES OF 26 COMPARISON TREES WE HAVE INCLUDED ON SEPARATE SHEETS.)

To describe the picture of the algorithm showing $M(5,9) = 12$ in words: to merge $A_{1..5}$ with $B_{1..9}$: first compare A_3 with B_5. If $A_3 > B_5$, that is equivalent to the position when $A_3 < B_5$, hence need not be investigated ("`T`"). Next compare A_1 with B_1. If $A_1 < B_1$, we may subdivide into subproblems ("`P`") and the rest of the analysis is omitted. Otherwise, compare A_5 with B_8 and A_1 with B_2:

If $A_5 < B_8$ and $A_1 < B_2$, partitioning.

If $A_5 < B_8$ and $A_1 > B_2$, compare A_3 with B_4 to reduce to either an upper bound or a partitioning.

If $A_5 > B_8$ then the comparison of A_1 with B_2 reduces matters to either an upper bound or a partitioning.

Even for this tiny example, the abbreviations we have used are essential, since the full comparison tree would have over a thousand nodes – too large to print here.

For five exceptionally large trees, even these abbreviations do not suffice to make a picture of the tree fit on one page. In those cases we have simply truncated the tree at depth 6.

The tree showing $M(5,42) = 22$ is important because it allows one to sort 47 items in 200 comparisons [12].

Also, various other upper bounds follow from these by applying known bounds-combining methods in §4.2. See the table in §2.

Frustratingly, the game solver was not able to establish any new lower bounds. The first few cases in the M table in which the lower bounds appear to be improvable are just outside the range of the gamesolver. But the game solver *has* confirmed some lower bounds obtainable by previous methods, and has exhaustively solved some *restricted width* merging games. In the below, let $M_w(a,b)$ denote the vlue of the restricted-width merging game with width w.

$M(4,40) = 19$ (confirming a result of Mönting) was simply done by brute force; when $M(m,n) \leq 20$, the game solver is efficient enough that no special tricks are required.

$M(4,48) = 20$ (also by Mönting) follows immediately from the previous result and "Splitting" (EQ 20).

$M_7(4,56) = 21$: Knuth's theorem 7 shows that in order for $M(4,56)$ to be 20, Min's first comparison must either be 1:8, in which case Max answers $A_1 > B_8$ forcing $1 + M(4,48) = 21$ comparisons total, or 2:b where $19 \leq b \leq 27$, in which case Max answers either that $A_1 < B_{20}$ or that $A_1 > B_{19}$ (one of these, or a weaker but nonconflicting statement is always possible). The game solver tried (but failed, even after 2 weeks) to search each of these two configurations exhaustively to determine that they both require at least 21 comparisons. Still, since an exhaustive search *was* completed for the restricted width merging game at width 7, any 20-comparison merging algorithm must be rather difficult. Thus the game solver may be regarded as having partially confirmed Mönting's [12] proof that $M(4,56) = 21$.

$M_{13}(5,37) = 22$: Knuth's theorem shows that if $M(5,37) = 21$, then the initial comparison must be 2:11, 2:12, or 2:13. It suffices for Max to answer $A_2 < B_{11}$.

$M_{12}(5,43) = 23$. Max forces the position $A_3 < B_21$ or $A_1 < B_6$.

$M_{11}(6,22) = 21$: Knuth's theorem shows that if $M(6,22) = 20$, then the initial comparison must be 2:5, to which Max prefers to answer $A_2 > B_5$, or 2:6, to which Max prefers to answer $A_2 < B_6$.

$M_6(6,25) = 22$: Knuth's theorem shows that if $M(6,25) = 21$, then the first comparison has to be 3:10 or 3:11. Max answers that $A_3 > B_{11}$.

$M_{12}(7,22) = 23$: Max forces the position $A_3 < B_9$.

To describe (roughly) the region of (m,n)-space that we explored with the gamesolver while preparing this paper: we tried to devote some computer time to every (m,n) with $M(m,n) \leq 30$ and $m \leq n \leq 50$, and also to every $m \leq n \leq 25$. A few sporadic choices were also explored outside these regions.

4 Other bounds on $M(m,n)$

4.1 Results for small values of m

Binary search shows that

$$M(1,n) = \lceil \lg(n+1) \rceil. \tag{4}$$

Hwang and Lin [8] (see also Graham [5]) showed that

Theorem 1 (Hwang and Lin 1971) $T_2(2n+1) = \lfloor 12N/7 \rfloor - 1$, $T_2(2n+2) = \lfloor 17N/7 \rfloor - 1$, *where* $n \geq 0$ *and* $N = 2^n$. *Equivalently,* $M(2,n) = \lceil \lg 7(n+1)/12 \rceil + \lceil \lg 14(n+1)/17 \rceil$ *when* $n \geq 3$.

Hwang [6] showed (we write some of his results in a form due to Mönting)

Theorem 2 (Hwang 1980)

1. $T_3(1) = 0$, $T_3(2) = T_3(3) = 1$, $T_3(4) = 2$;
2. $T_3(3n+5) = 2\lfloor 17N/7 \rfloor - 1$, *except that* $T_3(8) = 8$;
3. $T_3(3n+6) = \lfloor 43N/7 \rfloor - 2$
4. $T_3(3n+7) = \lfloor 59N/14 \rfloor + \lfloor 24N/7 \rfloor - 1$;

where $n \geq 0$ *and* $N = 2^n$.

Hwang's proof is extremely complicated. Also Nissenbaum [14] claims to have independently established the same results as Hwang, except that Nissenbaum gave the slightly different formula $T_3(3n+5) = \lfloor 34N/7 \rfloor - 1$. The game solver program has confirmed Hwang's values of $M(3,n)$ up to $n = 82$. Nissenbaum was in error.

Mönting [12] (theorem 3 page 37) claims that

Theorem 3 (Mönting 1981)

1. $T_4(1) = T_4(2) = 0$, $T_4(3) = T_4(4) = 1$, $T_4(5) = 2$, $T_4(6) = 3$;
2. $T_4(4n+7) = \lfloor 17N/7 \rfloor + \lfloor 59N/28 \rfloor + \lfloor 12N/7 \rfloor - 1$;
3. $T_4(4n+8) = 3\lfloor 17N/7 \rfloor - 1$, *except that* $T_4(12) = 12$;
4. $T_4(4n+9) = 2\lfloor 17N/7 \rfloor + \lfloor 27N/7 \rfloor - 1$, *except that* $T_4(13) = 15$;
5. $T_4(4n+10) = \lfloor 5N/7 \rfloor + \lfloor 545N/112 \rfloor + \lfloor 34N/7 \rfloor - 1$, *except that* $T_4(10) = 8$;

where $n \geq 0$ *and* $N = 2^n$.

Mönting's proof is *extremely* complicated and in fact, the full details are apparently deferred to a technical report [13] and are not present in his paper. The game solver confirms Mönting's values of $M(4,n)$ for all $n \leq 80$ except for $n = 56$ and $n = 68$, where it was unable to complete the exhaustive search. All it can say in these cases is $20 \leq M(4,56) \leq M_{14}(4,56) = 21$, $21 \leq M(4,68) \leq M_5(4,68) = 22$.

Theorem 4 (Mönting 1981)

1. $T_5(12) = 9$, $T_5(13) = 11$, $T_5(14) = 13$, $T_5(15) = 15$, $T_5(16) = 17$, $T_5(17) = 20$, $T_5(18) = 23$, $T_5(19) = 27$, $T_5(20) = 31$,
2. $T_5(5n+21) \geq \lfloor 40N/7 \rfloor + \lfloor 24N/7 \rfloor + \lfloor 118N/7 \rfloor + \lfloor 96N/7 \rfloor - 1$;
3. $T_5(5n+22) \geq \lfloor 59N/7 \rfloor + \lfloor 48N/7 \rfloor + \lfloor 136N/7 \rfloor + \lfloor 76N/7 \rfloor - 1$;
4. $T_5(5n+23) \geq \lfloor 38N/7 \rfloor + \lfloor 61N/7 \rfloor + 2\lfloor 136N/7 \rfloor - 1$;
5. $T_5(5n+24) \geq \lfloor 94N/7 \rfloor + \lfloor 61N/7 \rfloor + 2\lfloor 136N/7 \rfloor - 1$;

6. $T_5(5n+25) \geq \lfloor 118N/7 \rfloor + \lfloor 96N/7 \rfloor + 2\lfloor 136N/7 \rfloor - 1$; *where* $n \geq 0$ *and* $N = 2^n$.

Mönting conjectures that the inequalities are in fact equalities. The game solver has confirmed Mönting's upper bounds on $M(5,n)$ up to $n = 67$, and also has confirmed Mönting's conjecture that these bounds are tight, except that the game solver only can say $22 \leq M(5,43) \leq 23$, $23 \leq M(5,50) \leq 24*$, $24 \leq M(5,58) \leq 25$, and $25 \leq M(5,67) \leq 26*$. Here, if the upper bounds are tight in the non-$*$ cases, then they are automatically tight in the $*$ cases due to (EQ 20).

4.2 Upper bounds

The simplest general purpose merging algorithm is the "straight merge," in which A_1 is compared with B_1. The minimum of these is the minimum of the merged lists, and the merging problem is now reduced to $M(m, n-1)$ or $M(m-1, n)$, depending on which (B_1 or A_1) was the minimum. This shows that

$$M(m,n) \leq m + n - 1. \tag{5}$$

Christen [2] observes as his "lemma 0" that

Lemma 5 (Christen 1978) *Suppose* $A_i < B_{i+3}$ *is already known for* $1 \leq i \leq m$; *then at most* $2m$ *more comparisons are needed to complete the merging.*

Proof. Compare A_m with B_{m+1}. If $A_m < B_{m+1}$, finish with straight merge. Otherwise, A_m's position becomes known after one more comparison, this time with B_{m+2}, and then the problem is reduced to the situation of the lemma only with $m-1$ in place of m. This (plus the fact that the lemma is trivially true when $m = 1$) is a proof by induction. □

As a consequence, one may deduce the inequality[6]

$$M(a,b) \leq \max_{0 \leq k \leq a} 3k + 1 + M(a-k, b-k-4) - \delta_{a,k}, \quad \text{when } 1 < a \leq b-3. \tag{6}$$

By solving this recurrence, Christen arrived at his "theorem 0," which we may write (Christen's "h" is the last term on our right hand side) as

$$M(a,b) \leq 3a - 1 + \lfloor (b-a+1)/4 \rfloor, \quad b \geq a \geq 1. \tag{7}$$

In fact just using the recurrence (EQ 6) itself is presumably more powerful than (EQ 7) (since one can then use upper bounds arising from other merging algorithms), but (EQ 7) is more convenient for hand calculations of isolated values of $M(m,n)$.

Christen generalized his lemma by replacing "$i + 3$" by other linear functions of i. The next three cases are: Christen's lemma 1 with $k = 1$ and $k = 2$, and Christen's lemma 2 with $j = k = 1$ (correction to that lemma: "$T(k,j)$" should read "$T(k,i)$"). These are as follows.

[6] Throughout this paper $\delta_{a,b}$ is the Kronecker delta, that is, 1 if $a = b$ and 0 otherwise.

Lemma 6 (Christen 1978) *Suppose* $A_i < B_{2i+6}$ *is already known for* $1 \leq i \leq m$; *then at most* $3m$ *more comparisons are needed to complete the merging.*
Suppose $A_i < B_{4i+12}$ *is already known for* $1 \leq i \leq m$; *then at most* $4m$ *more comparisons are needed to complete the merging.*
Suppose $A_i < B_{10i+22}$ *is already known for* $1 \leq i \leq m$; *then at most* $5m$ *more comparisons are needed to complete the merging.*

These lead to the respective recurrences

$$M(a,b) \leq \max_{0 \leq k \leq a} 4k + 1 + M(a-k, b-2k-8) - \delta_{a,k} \tag{8}$$

when $1 < a$, $2a + 6 \leq b$,

$$M(a,b) \leq \max_{0 \leq k \leq a} 5k + 1 + M(a-k, b-4k-16) - \delta_{a,k} \tag{9}$$

when $1 < a$, $4a + 12 \leq b$, and

$$M(a,b) \leq \max_{0 \leq k \leq a} 6k + 1 + M(a-k, b-10k-32) - \delta_{a,k} \tag{10}$$

when $1 < a$, $10a + 22 \leq b$.

By solving these recurrences one may deduce

$$M(a,b) \leq 4a + \lfloor (b - 2(a-1))/8 \rfloor - 1, \quad b \geq 2(a-1) \geq 0, \tag{11}$$

$$M(a,b) \leq 5a + \lfloor (b - 4(a-1))/16 \rfloor - 1, \quad b \geq 4(a-1) \geq 0, \tag{12}$$

$$M(a,b) \leq 6a + \lfloor (b - 10(a-1))/32 \rfloor - 1, \quad b \geq 10(a-1) \geq 0, \tag{13}$$

analagously to (EQ 7). Example: (EQ 8) leads to $M(5,49) = 23$ and $M(6,51) = 27$, which are stronger upper bounds than one could obtain using (EQ 11)-(EQ 13). Christen actually has an infinite number of results of this type, each one proved with the aid of the previous ones. But in the range $3 < a \leq b \leq 100$ only the ones we have mentioned so far matter.

A powerful upper bound (e.g. [7]) arises by considering first comparing A_1 with B_x, and, if A_1 is larger, annexing $B_{1..x}$ to reduce the problem to $M(a, b-x)$, while if A_1 is smaller, binary inserting A_1 among $B_1..B_{x-1}$ to reduce the problem to $M(a-1, b-\delta)$, (for some $\delta \geq 0$). This yields the following recurrence:

$$M(a,b) \leq 1 + \min_{1 \leq x \leq b} \max\{M(1, x-1) + M(a-1, b),\ M(a, b-x)\}. \tag{14}$$

This is actually the special case $y = 1$ of the following recurrence

$$M(a,b) \leq 1 + \min_{1 \leq x \leq b} \min_{1 \leq y \leq a} \max \{M(y, x-1) + M(a-y, b),\ M(a-y+1, b-x) + M(b, y-1)\}, \tag{15}$$

but in the range $0 < a \leq b \leq 100$ apparently this more general result never improves upon the special case (EQ 14).

Another idea, related to (EQ 14), but involving inserting two elements instead of one, is

$$M(a,b) \le \min_{1\le x\le b} \min_{x<y\le b} \max \{2+M(a-2,b)+\texttt{M2[y-1][0][x]}, \quad (16)$$
$$2+M(1,x-1)+M(a-1,b-y),\ 1+M(a,b-x)\}.$$

The only case in the range $0 < a \le b \le 100$ in which this bound gives a result that cannot be obtained by simpler general purpose methods seems to be $M(5,96) \le 28$. But this is dominated by Mönting's $M(5,102) \le 28$.

By merging A with the even numbered elements of B first, we see that

$$M(a,b) \le M(a,\lfloor b/2\rfloor)+a. \quad (17)$$

Two successive merges show that

$$M(a,b+c) \le M(a,b)+M(a,c). \quad (18)$$

Some other merging algorithms have been proposed in [1], but they do not appear to yield any new results in the range $0 < a \le b \le 100$.

4.3 Lower bounds

We have already mentioned the information theoretic lower bound (EQ 1).

By regarding B_n as infinite

$$M(m,n-1) \le M(m,n). \quad (19)$$

Similarly to the way that upper bounds on $M(m,n)$ arise from merging algorithms (i.e. strategies for the Min player), lower bounds arise from counterstrategies for the Max player. One of the simplest, but still rather effective, such counterstrategies is the "splitting" strategy mentioned in [2].

The idea here is that after Max tells Min that $A_i < B_j$, Max splits the merging problem into two subproblems $A_{1..h}\&B_{1..k}$ and $A_{h+1..m}\&B_{k+1..n}$, where $h \ge i$ and $k < j$. Due to this splitting, the answer $A_i < B_j$ gave Min zero useful information to help him with the subproblems. Max chooses h and k. By no means is splitting necessarily the best strategy for Max to follow. But it is easier to analyse, because the splitting automatically ensures the consistency of Max's later answers and also leads to a simple recurrence relation:

$$M(m,n) \ge \min_{1\le i\le m} \min_{1\le j\le n} \max_{(h,k)\in P} 1+M(h,k)+M(m-h,n-k) \quad (20)$$

where "P" is

$$P \equiv (i \le h \le m \text{ and } 0 \le k < j) \text{ or } (0 \le h < i \text{ and } j \le k \le n). \quad (21)$$

Christen [2] was able to show that the lower bounds from a set of inequalities realted to this one meet the upper bounds from (EQ 7) or (EQ 11) when $5m-3 \le n \le 7m$.

Knuth [9] devised a more complicated, but more powerful counterstrategy, which Christen calls "the full disjunctive counterstrategy." We will not describe it here in full, since Knuth gives a complete description, but we will describe the basic ideas behind it and the recurrences that result. Knuth defines 9 functions "$\nabla\texttt{M}\phi(a,b)$" where ∇ and ϕ are each characters from the 3 member alphabet {'.' , '\' , '/' }. These represent the values of nine kinds of "constrained merging games." The meaning of ∇ and ϕ are:

- $\nabla =$. means no left constraint
- $\nabla = \backslash$ means Max's answers must be consistent with $A_1 < B_1$
- $\nabla = /$ means " " " " " " $A_1 > B_1$
- $\phi =$. means no right constraint
- $\phi = \backslash$ means Max's answers must be consistent with $A_m < B_n$
- $\phi = /$ means " " " " " " $A_m > B_n$.

For example, $.\texttt{M}.(a,b) = M(a,b)$ is just the value of the ordinary unconstrained merging game, $\backslash\texttt{M}\backslash(a,b)$ is the value of the merging game in which Max is constrained to make all his answers consistent with $A_1 < B_1$ and $A_m < B_n$, and $.\texttt{M}/(a,b)$ is the value of the merging game in which Max is constrained to make all his answers consistent with $A_m > B_n$. By convention if $m \le 0$ or $n \le 0$ then $\nabla\texttt{M}\phi(m,n) = 0$. It is important to note that, in these constrained games, Min *does not know* that Max is operating under a constraint, and thus may, in some variations, be forced to waste a comparison. Because of this, it is *not* generally the case that $.\texttt{M}/(a,b) = M(a-1,b)$, and the $.M/(a,b)$ function really does have a independent existence.

Knuth devised 6 possible ways for Max to split each of these kinds of merging games into two subgames (possibly of different types), and the full disjunctive counterstrategy consists of Max choosing the best of these 6 possible ways (and also maximizing over free parameters k and ℓ under his control which govern the location of the split) in response to any selection by Min of a comparison i:j. This leads to 54 lower bounds[7] on the $\nabla\texttt{M}\phi(a,b)$. Specifically, let

$$A(k,\ell) = \nabla\texttt{M}.(k,\ell-1)+.\texttt{M}\phi(m-k,n+1-\ell), \quad (22)$$

$$B(k,\ell) = \nabla\texttt{M}\backslash(k,\ell)+/\texttt{M}\phi(m-k,n+1-\ell), \quad (23)$$

$$C(k,\ell) = \nabla\texttt{M}/(k,\ell-1)+\backslash\texttt{M}\phi(m+1-k,n+1-\ell), \quad (24)$$

$$A'(k,\ell) = \nabla\texttt{M}.(k-1,\ell)+.\texttt{M}\phi(m+1-k,n-\ell), \quad (25)$$

$$B'(k,\ell) = \nabla\texttt{M}\backslash(k-1,\ell)+/\texttt{M}\phi(m+1-k,n+1-\ell), \quad (26)$$

$$C'(k,\ell) = \nabla\texttt{M}/(k,\ell)+\backslash\texttt{M}\phi(m+1-k,n-\ell). \quad (27)$$

Then

[7] Due to certain symmetries, the values 9 and 54 may be reduced.

Theorem 7 (Knuth) *In the $\nabla\mathtt{M}\phi(m,n)$ constrained merging game, assuming that Min's first move is i:j where $1 \le i \le m$, $1 \le j \le n$, Max can by using Knuth's counterstrategy force Min to use at least*

$$\max\{ \tag{28}$$
$$\max_{i\le k\le m}\max_{1\le\ell\le j}(1-\delta_{\ell,1}\delta_{\nabla,/})(1-\delta_{k,m}\delta_{\phi,/})A(k,\ell),$$
$$\max_{i\le k\le m}\max_{1\le\ell< j}(1-\delta_{\ell,1}\delta_{\nabla,/})(1-\delta_{k,m}\delta_{\phi,/})B(k,\ell),$$
$$\max_{i< k\le m}\max_{1\le\ell\le j}(1-\delta_{\ell,1}\delta_{\nabla,/})(1-\delta_{k,m}\delta_{\phi,/})C(k,\ell),$$
$$\max_{1\le k\le i}\max_{j\le\ell\le n}(1-\delta_{k,1}\delta_{\nabla,\backslash})(1-\delta_{\ell,n}\delta_{\phi,\backslash})A'(k,\ell),$$
$$\max_{1\le k\le i}\max_{j<\ell\le n}(1-\delta_{k,1}\delta_{\nabla,\backslash})(1-\delta_{\ell,n}\delta_{\phi,\backslash})B'(k,\ell),$$
$$\max_{1\le k< i}\max_{j\le\ell\le n}(1-\delta_{k,1}\delta_{\nabla,\backslash})(1-\delta_{\ell,n}\delta_{\phi,\backslash})C'(k,\ell)\}$$

more comparisons to complete the merging. Thus $1 + \min_{1\le i\le m}\min_{1\le j\le n}$ *of (EQ 28) is a lower bound on $\nabla\mathtt{M}\phi(a,b)$, regardless of what Min does.*

The strongest combination of these inequalities may be deduced automatically by "dynamic programming." Stockmeyer and Yao [15] were able to show that Knuth's lower bound meets the upper bound (EQ 5) from straight merge when $m \le n \le \lfloor 3m/2\rfloor + 1$.

By using the information theoretic bound (EQ 1), theorems 1, 2 or 3, or by actually solving for the value of some $\nabla\mathtt{M}\phi(a,b)$ with a game solver (in particular, for certain $M(a,b)$!), better lower bounds may sometimes be obtained than Knuth's recurrences would have otherwise provided. We point out that in this event, of course these improved values may be fed into Knuth's recurrences to produce additional improved lower bounds.

No improvement is possible for $\nabla\mathtt{M}\phi(1,n)$, since, as was shown in [9], $/M\backslash(1,n) = /M.(1,n) = M(1,n)$ for $n \ge 2$.

It is possible to compute the exact values of $\nabla\mathtt{M}\phi(2,n)$, by dynamic programming, and these *can* lead to improvements over the bounds one would obtain merely by naive use of Knuth's recurrences. We have already described how to compute $M(2,n)$ via M2[a][b][c] by use of (EQ 2). One may similarly define .M2\[a][b][c] and use it to compute $.\mathtt{M}\backslash(2,a) = .\mathtt{M2}\backslash[\mathtt{a}][0][\mathtt{a}+1]$ via

$$.\mathtt{M2}\backslash[\mathtt{a}][\mathtt{b}][\mathtt{c}] = 1 + \min(\tag{29}$$
$$\min_{1\le x<c}\max\{.\mathtt{M2}\backslash[\mathtt{a}-\mathtt{x}][\mathtt{b}-\mathtt{x}][\mathtt{c}-\mathtt{x}],\ .\mathtt{M2}\backslash[\mathtt{a}][\mathtt{b}][\mathtt{x}]\},$$
$$\min_{b<x\le a}\max\{\mathtt{M2}[\mathtt{x}][\mathtt{b}][\mathtt{c}],\ .\mathtt{M2}\backslash[\mathtt{a}][\mathtt{x}][\mathtt{c}](1-\delta_{x,a})\}).$$

There is a similar recurrence for /M2\[a][b][c] which depends upon .M2\[a][b][c] and M2[a][b][c]. Using these recurrences, we have verified that

Lemma 8

$$/M\backslash(2,n) = /M.(2,n) = M(2,n) \tag{30}$$

for $1 \le n \le 100$, except for $.\mathtt{M}\backslash(2,1) = 1$, $.\mathtt{M}\backslash(2,4) = 4$, $.\mathtt{M}\backslash(2,6) = 5$ *and* $/\mathtt{M}\backslash(2,1) = 0$, $/\mathtt{M}\backslash(2,2) = 2$, $/\mathtt{M}\backslash(2,4) = 4$, $/\mathtt{M}\backslash(2,6) = 5$.

(It is probably possible to remove the "≤ 100" from this lemma by using techniques like [8], but we have not done so. Also see the formula for $M(2,n)$ in theorem 1.)

As a consequence of lemma 8 and all the other boosts to Knuth's lower bound that we've just mentioned, a grand total of five new lower bounds on $M(m,n)$ arise in the range $0 < m \le n \le 50$: $M(14,28) \ge 39$, $M(19,36) \ge 52$, $M(20,40) \ge 56$, $M(25,48) \ge 69$, $M(27,46) \ge 71$. These correspond to the occurrences of "K" in the master table.

Also in the event that $M(6,22) = 21$, $M(6,25) = 22$, and $M(7,22) = 23$ were shown (these three conjectures have been mentioned previously as being just beyond the range of the game solver, but nevertheless supported by considerable evidence), it would follow from theorem 7 that $M(8,29) = 28$.

5 Conclusion and open problems

5.1 Understanding gap; patterns?

Whenever our game solver finds a new merging algorithm (or even more mysteriously, a new lower bound), one has a wierd combination of pleasant and unpleasant sensations. Pleasant because one has a new bound on some $M(m,n)$, and unpleasant because it is very hard for a human being to comprehend the proof of the new result, since the proof consists of a game tree (decision tree structured case analysis, if you prefer) with thousands (or possibly much more!) of nodes. Is there any underlying structure in this game tree that makes it possible for a human to understand what is going on, and to generalize the approach to other (m,n)?

It does not help matters that the game solver does not output the entire comparision tree for the merging algorithm it finds (due to cutoffs; the game solver of course attempts to avoid investigating vast chunks of the game tree). Kevin Lang's program, which we mentioned previously, actually produces the entire comparison tree representing Min's strategy, so that somebody with sufficient fortitude could in principle check its validity, or try to understand its structure. However, Lang's program (which does not use alphabeta cutoffs or partitioning) is substantially slower than the game solver. Also, of course nobody could hope to print out the entire comparison tree showing $M(15,27) = 40$, which would have $\approx 10^{11}$ nodes.

Understanding the results that the game solver has produced is a challenge. We note that the game solver has shown $M(n,2n+12) = 3n+3$ when $n \in \{3,4,5,6,7,8,9\}$, $M(n,2n) = 3n-3$ when $n \in \{10,11,12\}$, and $M(n,2n-3) = 3n-5$ when $n \in \{9,10,11,12,13,15\}$. Do these hold for all $n \ge 10$? Even more may be true than the last of these conjectures: perhaps $M(n,2n-3+k) = 3n-5+k$ when $n \ge 9$ and $0 \le k \le 2$. Also, notice these patterns: $M(n,4n+16) = 4n+1$ when $n \in \{1,2,3,4,5,6,7,8,9\}$, and $M(n,4n+22) = 4n+2$ when $n \in \{2,3,4,5,6\}$. Not all such patterns continue forever. For example

$M(n, 3n+27) = 4n+2$ when $n \in \{2,3,4,5,6,7,8\}$, but $M(9, 54) = 37$.

5.2 *Quick rigorous cutoffs in computer chess*

The "iterative widening" idea might be applicable in a different guise in computer game playing. Specifically, when computers play chess, they often encounter in their analysis situations in which one side has made a colossal mistake, such as losing a knight for no compensation. Unfortunately, instead of realizing this instantly and dismissing this line of play from consideration, the computer feels compelled to search the position all the way out to the preset search horizon. Then, and only then, it realizes that yes, indeed, there indeed *was* no compensation, and yes, indeed, in hindsight there was no reason to search that position at all.

In an effort to prevent this wasted effort, chess programmers sometimes adopt the "null move heuristic," which consists of making the assumption that the "null move" is not as strong a move as the best legal chess move. This (especially applied recursively) enables a lower bound on the lookahead value of that position to be obtained more cheaply than by the usual search procedure. Unfortunately, in chess it sometimes happens that the null move would be *better* than any legal chess move. This is called "zugzwang." This can cause chess programs with the null move heuristic to self-destruct! Also, zugzwang is so common in checkers that the null-move heuristic cannot be used at all.

So what is needed is a cheap way to get lower bounds – but without the flaw that sometimes, the lower bound is very incorrect! One possible solution is as follows: restrict one side's move choices (say white's) to, say, only include captures and the most plausible two noncaptures. Since white is playing "with one hand tied behind his back," the result of the tree search will be a lower bound on the value (to white) of that position. This lower bound will be achieved with considerably less effort than a full width search to the same horizon would have required, since the branching factor on white's moves is ≈ 7 instead of ≈ 30. The bound is also 100% rigorous and does not require any assumptions about the non-appearance of zugzwangs. The chess program should only seek such bounds in situations in which it is likely that they will be strong enough to cause beta cutoffs. In the event that a not-strong-enough lower bound is obtained, one must re-do the search with a larger white width to get a better bound, but hopefully, on average, this will be required sufficiently rarely that it is worth it.

5.3 *Other optimal algorithms via gameplaying?*

Next, it would be interesting to see if the approach of this paper can be used to make progress on other optimal algorithms problems besides merging. The obvious one to examine next is sorting.

Our mergegame solver (with our current hardware and software) can only solve for $M(m, n) \leq 20$, although it has found new upper bounds as much as 40 comparisons deep.

The first unknown cases in the sorting problem [9] involve more than 30 unknown comparisons. Hence it is just barely possible that an approach similar to ours will succeed in finding a new upper bound on the sorting function $S(N)$.

6 Acknowledgements

We would like to thank Henry Cejtin for the graphics program which drew the pictures of the trees.

References

[1] T.D. Bui and Mai Thanh: *Significant improvements to the Ford-Johnson algorothm for sorting*, BIT 25 (1985) 70-75

[2] C. Christen: *Improving the bounds on optimal merging*, Proc 19th IEEE conf on the foundations of computer science ["FOCS"] 259-266.

[3] G.B. Dantzig: *Linear programming and extensions*, Princeton University Press 1963.

[4] L.R.Ford Jr. and S.B. Johnson: *A tournament problem*, Amer. Math. Monthly 66,5 (1959) 387-389.

[5] R.L. Graham: *On sorting by comparisons*, pp. 263-269 in *Computers in Number theory* (Academic Press London 1971) A.O.Atkin and B.J. Birch eds.

[6] F.K. Hwang: *Optimal merging of 3 elements with N elements*, SIAM J Computing 9 (1980) 298-320

[7] F.K. Hwang and D.N. Deutsch: *A class of merging algorithms*, J. ACM 20,1 (1973) 148-159

[8] F.K. Hwang and S. Lin: *An optimal algorithm for merging an ordered set of length two with another ordered set*, Acta Informatica 1 (1971) 145-158.

[9] D.E. Knuth: *The art of computer programming vol 3, Sorting and Searching,* Addison Wesley 1973.

[10] D.E. Knuth and R.W. Moore; *Analysis of alphabeta pruning*, Artificial Intelligence 6 (1975) 293-326

[11] Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel: *Fast algorithms for finding randomized strategies in game trees*, 28th ACM Symposium of theory of computing ("STOC"), 1994.

[12] Jürgen Schulte Mönting: *Merging of 4 or 5 elements with n elements*, Theoretical Computer Science 14 (1981) 19-37

[13] Jürgen Schulte Mönting: *Optimal merging of 3, 4 and 5 elements with n elements*, Technical report at Universität Tubingen 1978.

[14] Y. Nissenbaum: *On merging N ordered elements with 3 elements*, SIGACT News 9,4 (1978) 14-16.

[15] P.K. Stockmeyer and F.F. Yao: *On the optimality of linear merge*, SIAM J. Computing 9 (1980) 85-90

[16] Mai Thanh and T.D. Bui: *An improvement of the binary merge algorithm*, BIT 22 (1982) 454-462

[17] A. C. Yao: *Probababilistic computation: Towards a unified measure of complexity*, Proc. 18th FOCS (1977) 222-227.