Quantum Algorithms for Evaluating Min-Max Trees

Richard Cleve[1,2], Dmitry Gavinsky[1], and D. L. Yonge-Mallo[1]

[1] David R. Cheriton School of Computer Science and Institute for Quantum Computing, University of Waterloo
[2] Perimeter Institute for Theoretical Physics

Abstract. We present a bounded-error quantum algorithm for evaluating Min-Max trees with $N^{\frac{1}{2}+o(1)}$ queries, where N is the size of the tree and where the allowable queries are comparisons of the form $[x_j < x_k]$. This is close to tight, since there is a known quantum lower bound of $\Omega(N^{\frac{1}{2}})$.

A Min-Max tree is a tree whose internal nodes are *minimum* and *maximum* gates, at alternating levels, and whose leaves are values from some underlying ordered set. The size N of such a tree is the number of its leaves, whose values are referred to as $x_1, \ldots, x_N$. The value of a Min-Max tree is the value of its root, a function of $x_1, \ldots, x_N$. In the *input value* query model, queries explicitly access the values of the leaves. In the *comparison* query model, the values of $x_1, \ldots, x_N$ are not directly accessible; rather, queries are comparisons of the form $[x_j < x_k]$. In this latter model, the appropriate output is any $j \in \{1, \ldots, N\}$ such that x_j is the value of the tree. The comparison query model is commonly used in the analysis of classical parallel algorithms for searching and sorting.

Note that, when the ordered set is $\{0, 1\}$, a Min-Max tree reduces to an And-Or tree. This implies that Barnum and Saks's lower bound of $\Omega(N^{\frac{1}{2}})$ [3] for the quantum query complexity of And-Or trees applies to Min-Max trees.

Recent results initiated by Farhi *et al.* have shown that quantum algorithms can evaluate all And-Or trees with order $N^{\frac{1}{2}+o(1)}$ queries [2,6,7,9]. We show that these results carry over to Min-Max trees in both the input value model and the comparison model.

Let $W(N)$ be the query complexity for And-Or trees of size N. We show that Min-Max trees can be evaluated with $O(W(N)\log(N))$ queries in both the input value model and the comparison model. Our algorithm combines the results on And-Or trees in Refs. [2,7] with the lemma below and Grover's search algorithm [10].

Lemma 1. *Let $\mathcal{T}$ be a* Min-Max *tree with inputs $x_1, x_2, \ldots, x_N$. Let $\mathcal{T}^v$ be an* And-Or *tree with identical structure to $\mathcal{T}$, but with* And *and* Or *gates in place of* Min *and* Max *gates (respectively), and with the k^{th} input assigned to* 1 *if and only if $x_k \geq v$. Then* $\text{value}(\mathcal{T}^v) = 1$ *if and only if* $\text{value}(\mathcal{T}) \geq v$.

Y. Kawano and M. Mosca (Eds.): TQC 2008, LNCS 5106, pp. 11–15, 2008.

Lemma 1 is easy to prove by induction. It implies that, if the underlying ordered set is a numerical range of size $N^{O(1)}$, then the tree can be evaluated in $\log(N)$ stages by a simple binary search. Each stage can be implemented with $O(W(N)\log\log(N))$ queries, which reflects the cost of evaluating an AND-OR tree amplified so that its error probability is $O(1/\log(N))$. The result is an $O(W(N)\log(N)\log\log(N))$ query algorithm.

A complication arises in performing such a binary search in the comparison model, where it is not possible to directly compute the midpoint of an interval like $[x_j, x_k]$. Problems can also arise in the input value model when the numerical range is too large: the binary search may not converge in a logarithmic number of steps. For this reason, we avoid the standard binary search approach where a midpoint is chosen as a pivot. Instead, we take a *random* input value among those that lie within a current interval as our pivot. What is noteworthy about this simple approach is that *it does not work efficiently in the classical case*: given an interval $[x_j, x_k]$, finding an interior point is as hard as searching, which can cost $\Omega(N)$ queries to do even once [4]. In the setting of *quantum* algorithms, we can utilize Grover's search algorithm [5,10] which costs $O(\sqrt{N})$.

As an aside, we note that there is a classical reduction from MIN-MAX trees to AND-OR trees that yields an $O(N^{0.753})$ query algorithm for balanced MIN-MAX trees [11]. We can use that reduction with an $N^{\frac{1}{2}}$ query *quantum* algorithm for balanced AND-OR trees; however, the resulting algorithm for MIN-MAX costs $\Omega(N^{0.58})$, due to the recursive structure of the algorithm[1]. Our alternate approach yields exponent $\frac{1}{2} + o(1)$ and is not restricted to balanced trees.

What follows is a description of our algorithm with the analysis of its error. For convenience, let $\perp$ and $\top$ be such that $x_\perp < x_j$ and $x_\top > x_j$ for any $j \in \{1, \ldots, N\}$ and let c be a constant.

QUANTUM MIN-MAX TREE EVALUATION

1. Let $\gamma \leftarrow \perp$ and $\delta \leftarrow \top$, and initialize the stack.
2. Repeat the following steps for $c\log(N)$ iterations, then go to Step 3:
   (a) Find a random pivot:
   Call the quantum search subroutine to find a random pivot index j with $x_\gamma < x_j < x_\delta$. If no value is found, go to Step 2(c).
   (b) Refine the search:
   Call the AND-OR tree subroutine to check if $\text{value}(\mathcal{T}) < x_j$. If so, let $\delta \leftarrow j$; otherwise, let $\gamma \leftarrow j$.
   (c) Backtrack if out of range:
   Call the AND-OR subroutine to check if $x_\gamma \leq \text{value}(\mathcal{T}) < x_\delta$. If so, push (γ, δ) onto the stack. Otherwise, pop (γ, δ) off the stack. (If the stack is empty, let $\gamma \leftarrow \perp$ and $\delta \leftarrow \top$.)
3. Return γ as an index corresponding to the value of the MIN-MAX tree.

[1] The expected number of calls to the binary subtrees is 3/2, essentially yielding a recurrence of the form $C(N) = (3/2)C(N/2) + N^{\frac{1}{2}}$ for the cost.

Clearly, the algorithm makes $O(W(N)\log(N))$ queries. We claim the following.

Theorem 1. *The algorithm returns the value of the* Min-Max *tree with probability at least* $\frac{2}{3}$.

To prove Theorem 1, we must consider the progress made by the random choices of pivots as well as the error probabilities of the subroutines for And-Or and the searches (each errs with constant probability).

To begin with, assume that the subroutines for And-Or and search never err (thus, $x_\gamma \leq \text{value}(\mathcal{T}) < x_\delta$ at all times). Under this assumption, the progress of the algorithm is determined by how quickly the subinterval converges. Once no value in Step 2(a) is found the algorithm has *converged* (with $x_\gamma = \text{value}(\mathcal{T})$) and can go to Step 3 and terminate (however it is harmless to perform more iterations before doing this).

Let $C(m)$ denote the expected number of iterations of the algorithm until it converges, assuming that m of its inputs are within its current range.

Then, for $m > 1$, $C(m)$ satisfies the recurrence

$$C(m) \leq \frac{2}{m}\left(\sum_{k=\lfloor m/2 \rfloor}^{m-1} C(k)\right) + 1. \tag{1}$$

This can be seen by assuming that the pivot is uniformly selected among all m possible positions within the subinterval and that value$(\mathcal{T})$ always lies in the larger side of the pivot. It is straightforward to verify that the recurrence implies $C(m) \in O(\log(m))$. Therefore, the expected number of iterations of Step 2 made by the algorithm before $x_\gamma = \text{value}(\mathcal{T})$, under the assumption that the subroutines never err, is $O(\log(N))$. By the Markov bound, $O(\log(N))$ iterations suffice to obtain error probability less than any particular constant.

We now consider the fact that the subroutines for And-Or and searching can fail. First, note that, by incurring a multiplicative factor of only $O(\log\log(N))$, each call to the And-Or and search algorithm can be amplified so that its error probability is $O(1/\log(N))$. This results in an $O(W(N)\log(N)\log\log(N))$ algorithm for Min-Max.

These amplification costs are not necessary in our algorithm, since it can cope with a constant fraction of errors in subroutine calls. To see why this is so, let ε be the probability that one or more subroutines err during one iteration of Step 2 of the algorithm. The algorithm begins some $O(\log(N))$ steps away from reaching a *good* state — of the form (α, δ) such that $x_\alpha = \text{value}(\mathcal{T})$. Before reaching a good state, an "incorrect" step for the algorithm places value$(\mathcal{T})$ outside the search interval, and a "correct" step either narrows the search interval or backtracks from a previous error. After reaching a good state, a "correct" step pushes a pair of the form (α, δ) onto the stack and an "incorrect" step pops it off. In each iteration, the algorithm takes a correct step with probability at least $1-\varepsilon$ and an incorrect step with probability at most ε. Therefore, with all but exponentially small probability, the number of correct steps minus the number of incorrect

ones after $c\log(N)$ iterations is at least $\frac{c}{2}\log(N)$. For suitably large c this means that, with constant probability, when the algorithm terminates, $\gamma = \alpha$ (typically with many copies of pairs of the form (α, δ) on the top of its stack).

Finally, we note that, in game-playing contexts, it is useful to determine optimal moves. This corresponds to finding the subtree of a MIN-MAX tree that attains its value. If the leaf values $x_1, \ldots, x_N$ are distinct, this is easily deduced from value($\mathcal{T}$). Otherwise, we can combine the MIN-MAX tree evaluation algorithm with the quantum minimum (maximum) finding algorithm of of Dürr and Høyer [8] to obtain the correct minimax decision. Suppose that we have a balanced MIN-MAX tree and that its root is a MAX gate with c children (each of which is a MIN-MAX tree with a MIN node at its root and N/c leaves). Then the Dürr-Høyer algorithm will make $O(\sqrt{c})$ evaluations of the subtrees, and each subtree will cost $O((N/c)^{\frac{1}{2}+\varepsilon})$ to evaluate. The case of unbalanced trees is more elaborate, but can be solved with the same asymptotic cost by using a generalization of Grover's algorithm that accommodates variable query times [1]. Thus, the minimax decision can be obtained in the same asymptotic cost that it takes to evaluate the tree.

Acknowledgements

We would like to thank Peter van Beek, Peter Høyer and Pascal Poupart for helpful discussions, and the anonymous reviewers for their comments. This research was supported in part by Canada's NSERC, CIAR, MITACS, QuantumWorks, and the U.S. ARO/DTO.

References

1. Ambainis, A.: Quantum search with variable times (arXiv:quant-ph/0609168)
2. Ambainis, A.: A nearly optimal discrete query quantum algorithm for evaluating NAND formulas (arXiv:quant-ph/0704.3628)
3. Barnum, H., Saks, M.: A lower bound on the query complexity of read-once functions. Journal of Computer and System Science 69(2), 244–258 (2004)
4. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing. SIAM Journal on Computing 26(5), 1510–1523 (1997)
5. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte Der Physik 46(4–5), 493–505 (1998)
6. Childs, A.M., Cleve, R., Jordan, S.P., Yeung, D.L.: Discrete-query quantum algorithm for NAND trees (arXiv:quant-ph/070 (2160))
7. Childs, A.M., Reichardt, B.W., Špalek, R., Zhang, S.: Every NAND formula on N variables can be evaluated in time $O(N^{\frac{1}{2}+\varepsilon})$ (arXiv:quant-ph/0703015)
8. Dürr, C., Høyer, P.: A quantum algorithm for finding the minimum (arXiv:quant-ph/9607014)
9. Farhi, E., Goldstone, J., Gutmann, S.: A Quantum Algorithm for the Hamiltonian NAND Tree (arXiv:quant-ph/0702144)

10. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC 1996), pp. 212–219 (1996)
11. Saks, M., Wigderson, A.: Probabilistic Boolean Decision Trees and the Complexity of Evaluating Game Trees. In: Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1986), pp. 29–38 (1986)