Semi-Heap and Its Applications in Tournament Ranking

Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
jie@cse.fau.edu

September 21, 2001
1. Introduction and Motivation

2. Preliminaries

3. Semi-Heap

4. Generalized Sorting Using Semi-Heap

5. Parallel Generalized Sorting Using Semi-Heap

6. Other Results

7. Conclusions
1 Introduction

Two different worlds

- Hardyism

  Utility as a goal is inferior to elegance and profundity.

- Maoism

  Scientific research should serve proletarian politics, ..., and be integrated with productive labor.
Golden-ratio-based Search:

Golden ratio: \( \phi = \frac{\sqrt{5} - 1}{2} = 0.618... \)

Questions:

- Why golden-ratio-based search?
- Golden-ratio-based search or binary-tree-based search?
• **Tournament**: $n$ players where every possible pair of players plays one game to decide the winner (and the loser) between them.

• **Graph representation**: A directed graph with a complete underlying graph.

![Graph diagram]

• **Hamiltonian path** (also called **generalized sorted sequence**):

\[ n_3 > n_4 > n_2 > n_5 > n_1 \]

• Lower bound: $\Theta(n \log n)$

• Sorting algorithms: bubble sort, binary insertion sort, merge sort

• We extend a heapsort algorithm using a **semi-heap** and then generalize it to a cost-optimal parallel algorithm under the EREW PRAM model with $\Theta(n)$ run time using $\Theta(\log n)$ processors.
Tournament Ranking Problem:

- generalized sorted sequence ($\Theta(n \log n)$)
- sorted sequence of kings (Wu 2000, conjectured to be $\Theta(n^2)$)
- local median order ($O(n^4)$)
- median order (NP-complete)

\[ n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_4 \rightarrow n_5 \]

**king**: other players are beaten by the king directly or indirectly (via a third player).

**median order**: ranking of players with a minimum number of total upsets.
2 Preliminaries

Tournament

Existence of a Hamiltonian path in any tournament:

Preposition: Consider a set $N$ ($|N| = n$) with any two elements $n_i$ and $n_j$, either $n_i > n_j$ or $n_j > n_i$. Elements in $N$ can be arranged in a linear order

$$n_1' > n_2' > \ldots > n_{n-1}' > n_n'$$

- Assume that the preposition holds for $n = k$:

  $$n_1' > n_2' > \ldots > n_k'$$

- When $n = k + 1$, we insert the $(k + 1)$th element $n_{k+1}'$ in front of $n_i'$, where $i$ is the smallest index such that $n_{k+1}' > n_i'$:

  $$n_1' > n_2' > \ldots > n_{k+1}' > n_i' > \ldots > n_k'$$

- If such an index $i$ does not exist, $n_{k+1}'$ is placed as the last element:

  $$n_1' > n_2' > \ldots > n_k' > n_{k+1}'$$
Heap

- Heap is an array $A$ that can be viewed as a **complete binary tree**.
- **Heap property**: For every node $i$ other than the root:
  \[
  A[\text{Parent}(i)] \geq A[i]
  \]
Faculty Recruiting Process:

A k-round selection process

Cost function:

<table>
<thead>
<tr>
<th>Type</th>
<th>Random</th>
<th>Sorted</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construct</td>
<td>(\Theta(1))</td>
<td>(\Theta(n \log n))</td>
<td>(\Theta(n))</td>
</tr>
<tr>
<td>Select</td>
<td>(\Theta(n))</td>
<td>(\Theta(1))</td>
<td>(\Theta(1))</td>
</tr>
<tr>
<td>Maintain</td>
<td>(\Theta(n))</td>
<td>(\Theta(1))</td>
<td>(\Theta(\log n))</td>
</tr>
</tbody>
</table>

Overall cost:

<table>
<thead>
<tr>
<th>Type</th>
<th>Random</th>
<th>Sorted</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k = \Theta(1))</td>
<td>(\Theta(n))</td>
<td>(\Theta(n \log n))</td>
<td>(\Theta(n))</td>
</tr>
<tr>
<td>(k = \Theta(n))</td>
<td>(\Theta(n^2))</td>
<td>(\Theta(n \log n))</td>
<td>(\Theta(n \log n))</td>
</tr>
<tr>
<td>(k = \Theta(n/\log n))</td>
<td>(\Theta(n^2))</td>
<td>(\Theta(n \log n))</td>
<td>(\Theta(n))</td>
</tr>
</tbody>
</table>
3 Semi-Heap

**Definition 1:** $n_1 = \max, \{n_1, n_2, n_3\}$ if both $n_2 = \max, \{n_1, n_2, n_3\}$ and $n_3 = \max, \{n_1, n_2, n_3\}$ are false.

Four possible configurations of a triangle in a semi-heap.

**Definition 2:** A semi-heap for a given intransitive total order $\succ$ is a complete binary tree. For every node $n'$ in the tree, $n' = \max, \{n', L(n'), R(n')\}$. 

---

10
Construct a semi-heap from a random array:

- SEMI-HEAPIFY\((A, i)\) constructs a semi-heap rooted at \(A[i]\), provided that binary trees rooted at \(A[l(i)]\) and \(A[r(i)]\) are semi-heaps.
  (Its cost is \(\Theta(\log n)\), where \(n = \text{heapsize}\).)

- BUILD-SEMI-HEAP\((A)\) uses the procedure SEMI-HEAPIFY in a bottom-up manner to convert an arbitrary array \(A\) into a semi-heap.
  (Its cost is \(\Theta(n)\))

SEMI-HEAPIFY\((A, i)\)
1. if \(A[i] \neq \max_{\sim} \{A[i], A[l(i)], A[r(i)]\}\)
2. then find winner such that
   \[A[\text{winner}] \leftarrow \max\{A[i], A[l(i)], A[r(i)]\}\]
3. exchange \(A[i] \leftarrow A[\text{winner}]\)
4. SEMI-HEAPIFY\((A, \text{winner})\)

BUILD-SEMI-HEAP\((A)\)
1. for \(i \leftarrow \left\lfloor \frac{\text{heapsize}}{2} \right\rfloor \) downto 1
2. do SEMI-HEAPIFY\((A, i)\)
The description of the SEMI-HEAPIFY algorithm:
**Theorem 1:** BUILD-SEMI-HEAP constructs a semi-heap for any given complete binary tree.

An example of using BUILD-SEMI-HEAP:
4 Generalized Sorting Using Semi-Heap

Why the traditional heapsort cannot be used?

- With the transitive property, root $A[1]$ “beats” all the other “players”.
- When the root is discarded, it is replaced by the last element $A[n]$ in the heap.
- Then the heap is reconstructed by pushing $A[n]$ down in the heap if necessary so that the new root is the maximum element among the remaining ones.
Generalized sorting using semi-heap

- Generalized sorting is done through SEMI-HEAP-SORT by repeatedly printing and removing the root of the binary tree (which is initially a semi-heap).

- The root is replaced by either its left child or right child through REPLACE.

- The selected child is replaced by one of its children. The process continues until a leaf node is reached and the entry for the leaf node is replaced by *.
• REPLACE($A, i$) repeatly replaces a node (starting from the root) by either its leftchild or rightchild until the current node is a leave node. (Its cost is bounded the height of the original semi-heap, $\Theta(\log n)$).

• SEMI-HEAP-SORT repeatly prints and removes the root of the binary tree (which is initially a semi-heap). (Its cost is $\Theta(n \log n)$.)

REPLACE($A, i$)

1. if $(A[l(i)] = *) \land (A[r(i)] = *)$
2. then $A[i] \leftarrow *$
3. else if $(A[i] > A[l(i)]) \land (A[l(i)] > A[r(i)])$
4. then $A[i] \leftarrow A[l(i)]$
5. REPLACE($A, l[i]$)
6. else $A[i] \leftarrow A[r(i)]$
7. REPLACE($A, r[i]$)

SEMI-HEAP-SORT($A$)

1. BUILD-SEMI-HEAP($A$)
2. while $(A[l(1)] \neq *) \lor (A[r(1)] \neq *)$
4. REPLACE($A, 1$)
5. print($A[1]$)
Theorem 2: For any given semi-heap, SEMI-HEAP-SORT generates a generalized sorted sequence.

Tournament representation:

- A tournament is represented by an $n \times n$ matrix $M$.
- $M[i, j] = 1$ if $n_i$ beats $n_j$ (i.e., $n_i \succ n_j$).
- $M[i, j] = 0$ if $n_i$ is beaten by $n_j$ (i.e., $n_j \succ n_i$).
- $M[i, i] = -$ represents an impossible situation.

$$M = \begin{pmatrix} - & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & - & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & - & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & - & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & - & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & - & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & - & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & - \end{pmatrix}$$
A step-by-step application of REPLACE($A, i$):

(a)  

(b)  

(c)  

(d)  

(e)  

(f)
5 Parallel Generalized Sorting Using Semi-Heap

A cost-optimal parallel algorithm

- A sorting algorithm is cost-optimal if the product of run time and the number of processors is $\Theta(n \log n)$.

- REPLACE$(A, 1)$ is pipelined level to level and this statement is called at every other step (since each node is shared by two processors at adjacent level, an idle step is inserted between two calls).

- The run time of SEMI-HEAP-SORT is reduced to $\Theta(n)$ with $\Theta(\log n)$ processors.

- This parallel algorithm runs on the CREW PRAM model, but can be easily modified to the EREW PRAM model without additional cost.
From CREW PRAM to EREW PRAM: resolve memory access conflict

The network model: a linear array of processors $P_0$, $P_1$, $P_2$, ..., $P_h$, where $h = \lceil \log n \rceil$. 
Active and passive steps:

- At an even step, processors $P_0$, $P_2$, $P_4$, ... take the active step and processors $P_1$, $P_3$, $P_5$, ... take the passive step.
- The role of active and passive among these processors exchanges in the next step.
Theorem 3: The proposed parallel implementation is cost-optimal with a run time of $\Theta(n)$ using $\Theta(\log n)$ processors.

$P_0$ at an active step (starts from step 0):


2. If both child nodes are $*$, $A[1]$ is replaced by $*$ and then $P_0$ sends a termination signal to $P_1$ and stops.


   - If $A[2]$ is selected, $P_0$ sends $id = 2$ to processor $P_1$; otherwise, $id = 3$ is sent.
   - In the next step (a passive step), $P_0$ receives $(id, \text{replacement})$ from $P_1$, and then, performs the update $A[id] \leftarrow \text{replacement}$.
\(P_i, \ i \neq 0\), at a passive step:

- If \(P_i\) receives \((id, \text{replacement})\) from \(P_{i+1}\), it performs the update \(A[id] := \text{replacement}\).

- If \(P_i\) receives signal \(id = j\) from \(P_{i-1}\), it performs the following activities in next active step:

  1. If both children are \(*\), \(A[j]\) is replaced by \(*\); otherwise, \(A[j]\) is replaced by either \(A[2j]\) or \(A[2j + 1]\) based the replacement rule.

  2. Send \((j, A[j])\) to \(P_{i-1}\).

  3. If either \(A[2j]\) or \(A[2j + 1]\) is selected to replace \(A[j]\), the corresponding id (\(2j\) or \(2j + 1\)) is sent to \(P_{i+1}\), provided \(P_i\) is not the last processor (i.e., \(i \neq h - 1\)); otherwise, the selected element is replaced by \(*\).

- If \(P_i\) receives the termination signal, it forwards the termination signal to the next processor \(P_{i+1}\) (if it exists) in the next active step, and then, \(P_i\) stops.
A step-by-step illustration:
6 Other Results

Sorted Sequence of Kings:

Quicksort

- \( \text{in}(u) \): a set of players that beat \( u \).
- \( \text{out}(u) \): a set of players that are beaten by \( u \).
Parallel Merge:

An EREW PRAM model with running time $O(\log^2 n)$ using $O(n/\log^2 n)$ processors.
7 Conclusions

- A data structure called semi-heap.
- An optimal solution to the generalized sorting problem.
- A cost-optimal EREW PRAM algorithm with $\Theta(n)$ in run time using $\Theta(\log n)$ processors.
- An implementation of the proposed parallel algorithm under the network model using a linear array of processors.