Supplemental Material

MDTS: Automatic Complex Materials Design using Monte Carlo Tree Search

Thaer M. Dieb[a,b], Shenghong Ju[c], Kazuki Yoshizoe[d], Zhufeng Hou[a], Junichiro Shiomi[a,c] and Koji Tsuda[a,b,d]

[a]National Institute for Materials Science, 1-2-1 Sengen, Tsukuba, Ibaraki 305-0047 Japan; [b]Graduate School of Frontier Sciences, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa, Chiba 277-8561, Japan; [c]Department of Mechanical Engineering, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo 113-8656, Japan; [d]RIKEN, Center for Advanced Intelligence Project, 1-4-1 Nihombashi, Chuo, Tokyo 103-0027, Japan

MDTS algorithm

MDTS: Materials Design using Monte Carlo tree search is an algorithm developed to solve structure determination of substitutional alloys problem. MDTS does not need any parameter tuning and can be adapted to several problems. The candidates space is represented as a tree where each node represents a possible atom assignment. Within a predetermined number of simulations, MDTS iterates over several rounds. Each round consists of four steps, Selection, Expansion, Simulation and Backpropagation. In the selection step, a promising leaf node is chosen by following the node with the best Upper Bound Confidence (UCB) score in each branch. The expansion step adds a number of children nodes to the selected one. In simulation, solutions are created by random playouts from the expanded nodes. The backpropagation step updates nodes' information along the path back to the root.

Input: N:number of positions in the structure, AT: List of atom types, AC: List of atom constraints, T:number of calculations
Output: List of positions optimally assigned with atoms

function MAKENODE($parent, atom$)
  $node \leftarrow object$ ▷ create abstract object
  $node.atom \leftarrow atom$ ▷ a node is an atom assignment to a position
  $node.v \leftarrow 0$ ▷ visit count
  $node.f \leftarrow 0.0$ ▷ immediate merit
  $node.w \leftarrow 0.0$ ▷ accumulated merit
  $node.z_{max} \leftarrow 0.0$ ▷ maximum immediate merit observed
  $node.z_{min} \leftarrow \infty$ ▷ minimum immediate merit observed
  $node.J \leftarrow 1$ ▷ adjust hyper-parameter C
  $node.parent \leftarrow parent$ ▷ node parent
  $node.children \leftarrow \emptyset$ ▷ node children
  return $node$
end function

CONTACT Koji Tsuda. Email: tsuda@k.u-tokyo.ac.jp

function SELECT($node$)
  if $node$ has no children **then**
    return $node$
  else
    $C \leftarrow \frac{\sqrt{2}node.J}{4}(node.z_{max} - node.z_{min})$
    $bst\_child \leftarrow argmax(\frac{node.w}{node.v} + C\sqrt{\frac{2ln(node.parent.v)}{node.v}})$
    return SELECT(bst_child)
  end if
end function

function EXPAND($node$)
  for all $atom \in AT$ **do**
    $child \leftarrow$MAKENODE($node, atom$)
    $node.children \leftarrow [node.children, child]$
  end for
  return $node.children$
end function

function SIMULATE($node$)
  $structure \leftarrow$ list of all atoms on the path from the root to $node$
  if $node$ is not maximum depth leaf **then**
    fill rest of the positions randomly ▷ random playout with AC ensured to be met
  end if
  return $structure$
end function

function BACKPROPAGATE($node, e$)
  $node.w \leftarrow node.w + e$
  $node.v \leftarrow node.v + 1$
  if $e > node.z_{max}$ **then**
    $node.z_{max} \leftarrow e$
  end if
  if $e < node.z_{min}$ **then**
    $node.z_{min} \leftarrow e$
  end if
  if $node.parent$ is not None **then**
    return BACKPROPAGATE($node.parent, e$)
  end if
end function

```
  function AdjuctC(node, j)
      node.J ← node.J + j
      if node.parent is not None then
          return AdjuctC(node.parent, j)
      end if
  end function

Start
  Tree ← MakeNode(None, None)
  t ← 0
  allsolutions ← ∅
  while t ≤ T do
      currentnode ← Select(Tree)
      if currentnode is maximum depth leaf then
          solution ← Simulate(currentnode)
          e ← evaluate solution using experiment or computation
          Backpropagate(currentnode, e)
          allsolutions ← [allsolutions, solution]
      else
          children ← Expand(currentnode)
          for all child ∈ children do
              solution ← Simulate(child)
              e ← evaluate solution using experiment or computation
              Backpropagate(child, e)
              allsolutions ← [allsolutions, solution]
          end for
      end if
      if dead-end leaf then
          j ← max{(T−t)/T, 0.1}
          AdjuctC(currentnode, j)
      end if
  end while
  return max(allsolutions)
End
```