Supplemental Material

MDTS: Automatic Complex Materials Design using Monte Carlo Tree Search

Thaer M. Dieb\textsuperscript{a,b}, Shenghong Ju\textsuperscript{c}, Kazuki Yoshizoe\textsuperscript{d}, Zhufeng Hou\textsuperscript{a}, Junichiro Shiomii\textsuperscript{a,c} and Koji Tsuda\textsuperscript{a,b,d}

\textsuperscript{a}National Institute for Materials Science, 1-2-1 Sengen, Tsukuba, Ibaraki 305-0047 Japan; \textsuperscript{b}Graduate School of Frontier Sciences, The University of Tokyo, 5-1-5 Kashiwanoha, Kashiwa, Chiba 277-8561, Japan; \textsuperscript{c}Department of Mechanical Engineering, The University of Tokyo, 7-3-1 Hongo, Bunkyo, Tokyo 113-8656, Japan; \textsuperscript{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.

\textbf{Input}: N:number of positions in the structure, AT: List of atom types, AC: List of atom constraints, T:number of calculations

\textbf{Output}: List of positions optimally assigned with atoms

\textbf{function} \textit{MAKE\_NODE}(parent, atom)

\begin{verbatim}
    node ← object  \hspace{1cm} \textcircled{1} create abstract object
    node.atom ← atom \hspace{1cm} \textcircled{2} a node is an atom assignment to a position
    node.v ← 0 \hspace{1cm} \textcircled{3} visit count
    node.f ← 0.0 \hspace{1cm} \textcircled{4} immediate merit
    node.w ← 0.0 \hspace{1cm} \textcircled{5} accumulated merit
    node.zmax ← 0.0 \hspace{1cm} \textcircled{6} maximum immediate merit observed
    node.zmin ← 0 \hspace{1cm} \textcircled{7} minimum immediate merit observed
    node.J ← 1 \hspace{1cm} \textcircled{8} adjust hyper-parameter C
    node.parent ← parent \hspace{1cm} \textcircled{9} node parent
    node.children ← 0 \hspace{1cm} \textcircled{10} node children

    return node
\end{verbatim}

CONTACT Koji Tsuda. Email: tsuda@k.u-tokyo.ac.jp
function Select(node)
    if node has no children then
        return node
    else
        C ← \sqrt{2/3} (node.z_{max} - node.z_{min})
        bst_child ← argmax(node.w + C \sqrt{2 \ln(node.parent.v) / node.v})
        return Select(bst_child)
    end if
end function

function Expand(node)
    for all atom ∈ AT do
        child ← MakeNode(node, atom)
        node.children ← [node.children, child]
    end for
    return node.children
end function

function Simulate(node)
    structure ← 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 ← node.w + e
    node.v ← node.v + 1
    if e > node.z_{max} then
        node.z_{max} ← e
    end if
    if e < node.z_{min} then
        node.z_{min} ← 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, 0.1}
        AdjuctC(currentnode, j)
    end if
end while
return max(allsolutions)
End