

Notation	Definition/Examples	Notation	Definition/Examples
x	Node of a tree	\mathcal{T}	patient's geometry& fitness values
ζ	profile map (column generation fitness values)	v	value
BOLD UPPERCASE TEXT	procedure or class	bold lower case text()	sub-routine

Table 1: Table of notations commonly used in this article

Algorithm 1 BOO Search

```

function RUN_SEARCH(col_gen_net) %azar's network
    agent = SEARCH(col_gen_net)
    search_data = agent.extract_search_data()
    return search_data
end function

function SEARCH(network)
    depth = User.N % user defined
    agent  $\leftarrow$  AGENT(network, False)
     $x \leftarrow$  agent.select_leaf()
     $\mathcal{T} =$  agent.get_input()
     $\zeta, v =$  network.predict( $\mathcal{T}$ )
     $x$ .propagate_results( $\zeta, v, x$ )
    while True do
        start  $\leftarrow$  time()
        current_depth =  $x.N$ 
        while  $x$ .root.N < current_depth + depth do
            agent.tree_search( $x$ )
        end while
        move =  $x$ .pick_move()
         $x$ .play_move(move)
        if agent.done() then
            break
        end if
    end while
    return agent
end function

```

Algorithm 2 Tree Search Agent

```

% procedure may be implemented as a class
procedure AGENT(network, two_player_mode)
  AGENT.root = NULL
  AGENT.network = network
  AGENT.mode = two_player_mode
  AGENT.start_search()

  function START_SEARCH(beam_position)
    AGENT.root = MCTS_NODE(beam_position)
    % buffer for this tree's policy
    AGENT.policy.Buffer=EMPTY_ARRAY
    % current  $\gamma$  buffer
    AGENT. $\gamma$ .Buffer=EMPTY_ARRAY
  end function

  function GUESS_MOVE(position)
    depth = AGENT.root.N
    % User.N is a global to the whole algorithm
    while AGENT.root.N < depth + User.N do
      AGENT.tree_search(x)
    end while
    return AGENT.pick_move()
  end function

  function PICK_MOVE()
    c  $\leftarrow$  cdf(AGENT.root.children)
    select  $\leftarrow$  random(low=1, high=180)
    move  $\leftarrow$  sort(c[select])
    return move
  end function

  % "" implies a procedural attribute
  function PLAY_MOVE(move)
    .policy.BUFFER.add(.root.c_policy())
    . $\gamma$ .BUFFER.add(AGENT.root. $\gamma$ )
    .root  $\leftarrow$  .root.maybe_add_child(move)
    .position = .root.beam_position
  end function

  function TREE_SEARCH(num_threads=NULL)
    % buffer for leaves added so far
    leaves  $\leftarrow$  EMPTY_ARRAY
    num_threads = depth if NULL
    fail = 0
    while fail < num_threads  $\times$  2 do
      fail  $\leftarrow$  fail + 1
      leaf = .root.select_leaf()
      if leaf.done() then
        value = leaf.position.score()
        leaf.backup(value, .root)
      end if
      leaves.add(leaf)
    end while
    continued on next page
  end function

```

Algorithm 3 Agent's Tree Search Agent Continued

```

procedure AGENT(network, two_player_mode)
  function TREE_SEARCH(num_threads=NULL)
    if leaves: then
      probs, vals = .network.predict(leaves)
      for {leaf, move, val} ∈ (leaves, probs, vals) do
        leaf.propagate_results(move, val,
up_to=AGENT.root)
      end for
    end if
    return leaves
  end function
end procedure

```

Algorithm 4 Tree Search Node

```

% procedure may be e.g. a class
procedure PARENT_NODE
  % essentially, tree's root node placeholder
  NODE.parent = NULL
  NODE.child_N = set()
  NODE.child_Ag = set() % agent's child
end procedure

procedure NODE(position, profile, parent)
  % "." implies a procedural attribute
  NODE.parent = parent %null if root
  NODE.position = position %beamlets
  NODE.move_probs = profile
  NODE.child_N = 0180×1
  NODE.child_Ag = 0180×1
  NODE.nonterminal = False %terminal?
  NODE. $\mathcal{T}$  = position.get_input()
  NODE.prior = 0180×1
  NODE.child_prior = 0180×1
  NODE.children = set() % e.g. a python dict

  % calculates the child action-value score
  @property
  function CHILD_ACTION_SCORE()
    % .to_play negates score for oppo. ag.
    return .child_Q × .position.to_play + .child_U
  end function

  function CHILD_Q()
    return .child_Ag / (1 + child_N)
  end function

  %essentially exploration bonus term. Diverges moves
  from early play. See alpha go paper
  @property
  function CHILD_U()
    % broadcast scalars to vectors for vector ops.
    return  $c\sqrt{\max(1, \text{NODE}.N - 1) \frac{\text{child\_prior}}{1 + \text{child}_N}}$ 
  end function

  @property
  function Q()
    return NODE.Ag / 1 + Node.N
  end function

  @property
  function Ag()
    return .parent.child_Ag[probs]
  end function

  @property
  function N()
    return .parent.child_N[probs]
  end function

```

continued on next page

Algorithm 5 MCTS Node Procedure Continuation

```

procedure PARENT_NODE % continuation
  function SELECT_LEAF()
    current = NODE
    mutate=False
    mutate_count=0
    while current.non_terminal do
      previous = current
      if mutate_count % 20 == 0 then
        mutate = True
      end if
      if current.expanded: then
        break
      else
        move = argmax(current.child_action_score)
        current.may_be_add_child(move, mutate)
      end if
    end while
    best_move = argmax(current..child_action_score)
    current.may_be_add_child(move, mutate)
    mutate_count += 1
    return current
  end function

  function MAY_BE_ADD_CHILD(move, mutate=False)
    %mutate controls the selection of new patients
    new_position = NODE.position.play_move(move,
    mutate)
    NODE.children[move] = NODE(new_position, probs=move,
    NODE)
    return NODE.children[move]
  end function
end procedure

```
