Nested Rollout Policy Adaptation for Monte Carlo Tree Search

Christopher D. Rosin
Parity Computing, Inc.
San Diego, California, USA
christopher.rosin@gmail.com

Abstract

Monte Carlo tree search (MCTS) methods have had recent success in games, planning, and optimization. MCTS uses results from rollouts to guide search; a rollout is a path that descends the tree with a randomized decision at each ply until reaching a leaf. MCTS results can be strongly influenced by the choice of appropriate policy to bias the rollouts. Most previous work on MCTS uses *static* uniform random or domain-specific policies. We describe a new MCTS method that dynamically *adapts* the rollout policy during search, in deterministic optimization problems. Our starting point is Cazenave's original Nested Monte Carlo Search (NMCS), but rather than navigating the tree directly we instead use gradient ascent on the rollout policy at each level of the nested search. We benchmark this new Nested Rollout Policy Adaptation (NRPA) algorithm and examine its behavior. Our test problems are instances of Crossword Puzzle Construction and Morpion Solitaire. Over moderate time scales NRPA can substantially improve search efficiency compared to NMCS, and over longer time scales NRPA improves upon all previous published solutions for the test problems. Results include a new Morpion Solitaire solution that improves upon the previous human-generated record that had stood for over 30 years.

1 Introduction

Monte Carlo tree search (MCTS) methods have had substantial recent success in two-player games [Gelly *et al.*, 2007; Finnsson *et al.*, 2010], planning [Nakhost *et al.*, 2009; Silver *et al.*, 2010], optimization and one-player games [Cazenave, 2009; Rimmel *et al.*, 2011; Méhat *et al.*, 2010] and practical applications [de Mesmay *et al.*, 2009; Cazenave *et al.*, 2009]. In this paper, we focus on deterministic optimization problems. For these, *nested* Monte Carlo search has been particularly successful, with world-record results in several problems [Cazenave, 2009; Bjarnason *et al.*, 2007].

MCTS guides search using results from rollouts. The base rollout policy is a key factor in MCTS success. Most prior MCTS work uses static policies, but some work has appeared on adapting rollout policies in two-player games [Silver *et al.*, 2009; Tesauro *et al.*, 1996; Finnsson *et al.*, 2010]. One recent study uses an evolutionary algorithm to choose policy parameters [Rimmel *et al.*, 2011]. Methods for adapting rollout policies also exist in control and reinforcement learning [Bertsekas, 1997; Fern *et al.*, 2003; Veness *et al.*, 2011].

In this paper, we describe a new MCTS algorithm that adapts its rollout policy during search. It resembles Cazenave's Nested Monte Carlo Search (NMCS) [Cazenave, 2009], in that every search step advances towards the current best known solution. But whereas NMCS makes these moves directly on the search tree, here we instead adapt a rollout policy towards the current best solution.

Section 2 describes the new algorithm. Section 3 describes the test problems: two Crossword Puzzle Construction instances for which the search tree has large branching factor but relatively small depth, and two versions of Morpion Solitaire with lower branching factor but larger depth. Section 4 has experimental results and comparison to NMCS.

2 Algorithm

We start with Cazenave's Nested Monte Carlo Search (NMCS) [Cazenave, 2009], and then proceed to the new Nested Rollout Policy Adaptation (NRPA) algorithm. For both, search is initiated at a *nesting level* n, which is the depth of recursion for the nested search. At each nesting level, the algorithm searches for a solution sequence (a sequence of child selections defining a path descending the tree to a leaf). In searching for improvements to its current best sequence, nesting level n makes recursive calls to nesting level $n-1$, bottoming out in the base rollout policy at level 0. When nesting succeeds, level n returns substantially higher-quality solutions than level $n-1$ [Cazenave, 2009].

2.1 Nested Monte Carlo Search

Fig. 1 shows NMCS. Lines 2-9 are the level 0 rollout from a given node of the search tree; a uniform random policy is shown. Nesting levels $n \geq 1$ start from a given node and iterate through successive ply (lines 14 and 22-23). Each iteration recursively calls level $n-1$ on each child of the current node (lines 15-17), replacing the current best solution from the current ply to the end if an improvement is found (lines 18-21). Each iteration concludes by advancing the current node one ply towards the current best solution (line 22).

```
NMCS(level,node):
  IF level==0: // base rollout policy
    ply = 0, seq = {} // count from initial node
    WHILE num_children(node)>0:
      CHOOSE seq[ply] = child i with probability
        1/num_children(node)
      node = child(node,seq[ply])
      ply += 1
    RETURN (score(node),seq)

  ELSE:   // for nesting levels>=1
    ply = 0, seq = {} // count from initial node
    best_score = -infinity
    WHILE num_children(node)>0:
      FOR children i of node:
        temp = child(node,i)
        (result,new) = NMCS(level-1,temp)
        IF result>best_score THEN:
          best_score = result
          seq[ply] = i
          seq[ply+1...] = new
      node = child(node,seq[ply])
      ply += 1
    RETURN (best_score,seq)
```

Figure 1: NMCS with Uniform Rollout Policy

```
 1 NRPA(level,pol):
 2   IF level==0: // base rollout policy
 3     node = root(), ply = 0, seq = {}
 4     WHILE num_children(node)>0:
 5       CHOOSE seq[ply] = child i with probability
 6         proportional to exp(pol[code(node,i)])
 7       node = child(node,seq[ply])
 8       ply += 1
 9     RETURN (score(node),seq)
10
11   ELSE:   // for nesting levels>=1
12     best_score = -infinity
13     FOR N iterations:
14       (result,new) = NRPA(level-1,pol)
15       IF result>=best_score THEN:
16         best_score = result
17         seq = new
18       pol = Adapt(pol,seq)
19    RETURN (best_score,seq)
20
21 Adapt(pol,seq):   // a gradient ascent step towards seq
22  node = root(), pol' = pol
23  FOR ply = 0 TO length(seq)-1:
24   pol'[code(node,seq[ply])] += Alpha
25   z = SUM exp(pol[code(node,i)]) over node's children i
26   FOR children i of node:
27    pol'[code(node,i)] -= Alpha*exp(pol[code(node,i)])/z
28   node = child(node,seq[ply])
29  RETURN pol'
```

Figure 2: NRPA with Global Parameters `Alpha` and `N`

An NMCS search tree is defined by operations root(), num_children(node), child(node,i) returning the i'th child of node, and score(node) at a leaf. Search at nesting level L is initiated by NMCS(L,root()). The only tunable parameter, nesting level, has just a few feasible settings. More domain-specific tuning can be done by reshaping the tree [Bjarnason *et al.*, 2007] and choosing a heuristic level 0 policy [Bjarnason *et al.*, 2007; Cazenave, 2007; Rimmel *et al.*, 2011].

2.2 Nested Rollout Policy Adaptation

Fig. 2 shows the new Nested Rollout Policy Adaptation (NRPA) algorithm. Lines 2-9 are the level 0 rollout, which follows (lines 5-6) a given weighted policy from the root to a leaf. Nesting levels $n \geq 1$ do a fixed number of iterations (line 13) starting from an initial given policy. Each iteration uses the current policy as a starting point in a recursive call to level $n-1$ (line 14), replacing the current best solution when a tie or improvement is found (lines 15-17), then advancing the policy one gradient ascent step towards the current best solution sequence (line 18 and Adapt(); see Section 2.3). Note level $n-1$'s final policy is not returned to level n.

Search at nesting level L is initiated by NRPA(L,pol) with all-zero vector pol. An NRPA search tree is defined by the same operations as NMCS, with the addition of code(node,i) which returns a domain-specific code for the *action* leading from the node to its i'th child. This provides the domain-specific representation for policy adaptation. No further domain-specific tuning of the policy is available; it is determined by adaptation starting from a uniform random policy. In addition to the nesting level parameter, NRPA allows tuning of `Alpha` and number of iterations `N` per level. Results here use `Alpha`=1.0 and N=100, across the test problems.

2.3 NRPA's Design

NMCS can be applied to any search tree, even when nodes and children have no known properties beyond their place in the tree. But many problems have additional structure. NRPA benefits when actions with the same code have similar value across a range of nodes. The general idea, that actions can have similar value in a range of contexts, has helped improve efficiency of other MCTS methods as well [Finnsson *et al.*, 2010; Gelly *et al.*, 2007; Akiyama *et al.*, 2010].

In our test problems, the same state can be reachable by multiple paths; this could be represented by a DAG [Saffidine *et al.*, 2010] rather than a tree. NRPA can benefit by exploring permutations of actions that reach the same state but offer unique departure points for further exploration. Section 4.4 suggests NRPA may also benefit by bringing actions into new contexts, beyond simple permutations of paths in a DAG.

NRPA may not obtain robust policies that give viable results starting from arbitrary nodes in the tree. Instead, NRPA's policies emphasize actions taken by the best solutions so far, and help push rollouts into the neighborhood of these solutions. This is how NRPA focuses search, since levels $n \geq 1$ do not maintain an explicit current tree node like NMCS does. Passing the policy recursively helps level n obtain results from level $n-1$ relevant to its current focus.

NRPA rollouts choose child i from node s with probability proportional to $e^{\mathrm{pol}[\mathrm{code}(s,i)]}$, where $\mathrm{pol}[x]$ is the adaptable weight on code x. This is comparable to forms used in past work on rollout policy adaptation [Finnsson *et al.*, 2010; Silver *et al.*, 2009; Rimmel *et al.*, 2011; Coulom, 2007].

Adapt() increases the probability of the current best sequence by raising the relative weight of its codes (also affecting other sequences that share some codes). Log of sequence probability is concave here, amenable to Adapt()'s gradient ascent. A sequence of children c_t chosen at nodes s_t has probability $\prod_t (e^{\mathrm{pol}[\mathrm{code}(s_t,c_t)]})/(\sum_i e^{\mathrm{pol}[\mathrm{code}(s_t,i)]})$; the log is $\sum_t \left(\mathrm{pol}[\mathrm{code}(s_t,c_t)] - \log(\sum_i e^{\mathrm{pol}[\mathrm{code}(s_t,i)]})\right)$. The gradient is a sum over t; t's contribution to component $\mathrm{code}(s_t,j)$ for $j \neq c_t$ is $-(e^{\mathrm{pol}[\mathrm{code}(s_t,j)]})/(\sum_i e^{\mathrm{pol}[\mathrm{code}(s_t,i)]})$ (lines 25-27) and for $j = c_t$ is $1-(e^{\mathrm{pol}[\mathrm{code}(s_t,j)]})/(\sum_i e^{\mathrm{pol}[\mathrm{code}(s_t,i)]})$ (first term is in line 24).

NRPA ensures search at a particular level eventually (given enough iterations) returns to the best score so far: as long as it does not, Adapt() keeps increasing the probability of the best

Table 1: Test Problems

	Depth	Branch	Old Record	New
MorpT	177	46	170 [Bruneau, 1976]	177
MorpD	82	46	80 [Cazenave, 2009]	82
CrossC	37	1197	36;256 [CrossC, 2006]	37;259
CrossP	28	988	28;236 [CrossP, 1994]	28;238

solution sequence so far until it is overwhelmingly likely to recapitulate that solution sequence. Adapt() can increase this probability exponentially quickly: very roughly, one call increases probability by close to a factor of $e{*}\texttt{Alpha}$ on each action that still has low probability. This rapid increase in sequence probability has only modest dependence on branching factor and sequence length, and in Section 4 a single value of `Alpha` works well across a range of problems.

This convergence towards recapitulation depends upon NRPA adapting only towards a single best-so-far solution. If it was based instead on a mix of solutions, representational limitations might cause convergence towards a suboptimal compromise amongst multiple solutions, preventing recapitulation of any one best-so-far solution score. Nonetheless, we do replace the current best solution with a new equally good one that ties its score. This preserves the property of return to the best score found so far, while aiding exploration. In preliminary experiments, taking ties gave NRPA substantially better results, compared to taking only strict improvements.

NRPA can be compared to other distribution-based search methods [Larranaga *et al.*, 2002].

2.4 Choosing Policy Representations for NRPA

The following two properties suffice so that NRPA's policy can converge towards any sequence. First, for any node s, code(s,i) should be distinct so that any child can be assigned arbitrarily high probability. Second, any solution sequence with nodes s_t in which children i_t are selected should have $\mathrm{code(s_t, i_t)}$ distinct from $\mathrm{code(s_{t'}, i)}$ for all $t' > t$ and all i. That is, selecting i_t in s_t "retires" $\mathrm{code(s_t, i_t)}$, so putting high probability on i_t doesn't interfere with later action selection.

In addition, if any legal permutation of a sequence $\mathrm{code(s_t, i_t)}$ yields the same score, NRPA can explore permutations more freely with less need to constrain ordering.

The test problems here each have a simple and natural representation that adheres to the above properties. Our final results use the same representations that were initially tested; no exploration of representational alternatives was needed.

Other problems may benefit from exploration of representational alternatives, as is often the case in machine learning. For NRPA, actions with same code should have similar value. But this only needs to be approximately true, and only within limited scope (as opposed to the entire state space). Actions with distinct codes are searched separately, so the total number of available codes should be minimized. Some problems may benefit from code(s,i) including limited additional context from global state s [Finnsson *et al.*, 2010].

3 Test Problems

3.1 Crossword Puzzle Construction

An instance of Crossword Puzzle Construction (CPC) provides a grid and a dictionary of allowed words. A feasible

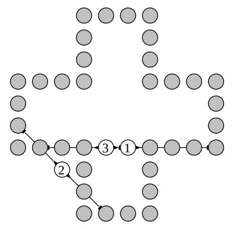


Figure 3: Morpion Solitaire: gray dots are the initial configuration. Moves 1 and 2 are legal in both versions, move 3 only in MorpT.

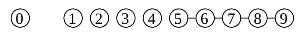


Figure 4: Dominated Move

solution is a legal crossword. A legal crossword places a subset S of the dictionary on the grid, with words oriented either horizontally left-to-right or vertically top-to-bottom, such that every two adjacent occupied grid positions are part of one of the placed words from S (Fig. 7a). All occupied grid positions must be connected by adjacency (no disjoint components). We seek a solution that uses as many words from the dictionary as possible, breaking ties according to the largest total number of letters used in the words in the solution. CPC scores are shown as #words;#letters, so 25;243 is 25 words and 243 letters and is superior to 24;251 and 25;239. Instances here have a square grid with all positions open to use.

Heuristic search has previously been applied [Ginsberg *et al.*, 1990] to a generalized decision version of CPC which is NP-complete [Garey and Johnson, 1979].

Instances here are from *GAMES Magazine* contests. Contests were diverse (not all CPC). Contests were approachable by humans without computer assistance, but computer assistance was allowed. Contests included monetary prizes, and attracted high-quality solutions. From over 12 years of contests, we select two CPC instances with largest grid (19x19). **CrossC** has 63 allowed words: 50 state, 10 provincial, and 3 territorial capitals in the U.S. and Canada with spaces, punctuation, and accents removed (e.g. Québec City becomes QUEBECCITY) [CrossC, 2006]. **CrossP** has 52 words: the 52 U.S. National Parks listed in *The 1994 World Almanac* with spaces and punctuation removed [CrossP, 1994]. Both yielded a unique winner (Table 1) from hundreds of entries.

Nodes in the CPC search tree are feasible solutions as defined above, starting with zero words at the root and adding one word per ply until reaching leaves in which no additional words can be added.[1] The first ply adds an initial word, constrained to be vertical and start in the top row. This is without loss of generality: any solution that uses the top row must have a vertical word starting there, and any solution that does not use the top row can be shifted upwards without changing score. Beyond the first word, children of a node consist of all single words which may be added to leave a new legal state.

For NRPA, code(node,i) identifies the word added by child i together with its grid position and orientation.

3.2 Morpion Solitaire

Morpion Solitaire [Boyer, 2011] is a pencil-and-paper puzzle on a square grid that is presumed to be infinite. Given a current configuration of dots (marked intersections on the grid), a legal turn requires the addition of one more dot such that

[1]This can't reach legal states having a 2x2 or larger block filled with letters. But such states are highly constrained, didn't appear in published solutions, and weren't needed to get our positive results.

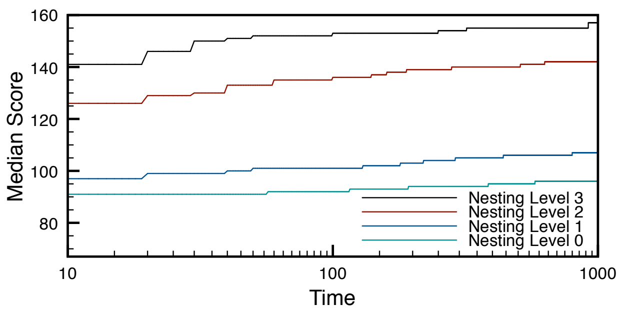


Figure 5: NRPA Median Timelines for MorpT

a new line of 5 dots in a row can be formed, horizontally or vertically or diagonally. The line is also added to the grid, and new lines cannot overlap any previous line. Solutions are scored by the number of lines they contain, and the goal is to maximize this. We consider two versions: the "Touching" version (**MorpT**) allows parallel lines to share an endpoint, but the "Disjoint" version (**MorpD**) does not. Generalized Morpion Solitaire is NP-hard [Demaine *et al.*, 2006]. Here we consider the common starting configuration (Fig. 3).

The best previous computer search results for Morpion Solitaire came from MCTS methods. A MorpD record of 80 was set by NMCS [Cazenave, 2009]. For MorpT, the best previously reported method was an NMCS variant that gave 146 [Akiyama *et al.*, 2010]. But a 1976 human-generated MorpT record of 170 [Bruneau, 1976] stood for over 30 years, until NRPA obtained 172 in August 2010 [Boyer, 2010; 2011]. This paper is the first to describe NRPA, and the record is improved to 177 in this paper.

Our search tree has the starting configuration at the root. Children of a node consist of all legal distinct lines that may be added to the grid; such a line uniquely determines the dot that is added. Leaves of the tree have no further legal moves.

We modify the tree slightly by replacing a type of *dominated move*. In Fig. 4, two moves add a new dot between 0 and 1: move A extends from 0 to 3, and move B from the new dot to 4. Move A *blocks* line segments 3-4 and 4-5; after move A these cannot be used by later moves. Move B blocks only 4-5. Move B occupies 3-4 whereas A does not, but since A blocks 3-4 it does not gain any advantage by leaving 3-4 unoccupied. Move A is dominated by B: any legal sequence of moves that includes A is still legal if A is replaced by B, and B may open up more possibilities (by leaving the segment from 0 to the new dot available). During NMCS and NRPA, nodes with an available dominated move (like A) retain the same number of children, and the dominated move may be selected and recorded as part of the solution sequence, but its effect is replaced by that of the dominating move (like B). This gave a small improvement in early experiments, more so than removing dominated moves entirely.

For NRPA, code(node,i) uniquely identifies the endpoints of the chosen line to be added by child i. In the case of a dominated move, it still codes the line represented by the dominated move (even though the line actually placed is different). This ensures all children have a unique code even when dominated moves exist. We use a 64x64 grid with the initial configuration near the center, which sufficed so that search did not approach the edges of the grid.

Table 2: Median Scores from Timed Runs

	Method	Level	10^2 sec	10^3 sec
MorpT	NRPA	3	153	157
MorpT	NMCS	3	116	130
MorpD	NRPA	3	79	81
MorpD	NMCS	3	71	74
CrossC	NRPA	3	33;229	34;244
CrossC	NRPA	2	32;240	33;242
CrossC	NMCS	2	31;223	32;229
CrossP	NRPA	3	26;222	27;223
CrossP	NRPA	2	26;219	26;232
CrossP	NMCS	2	25;208	26;215

3.3 Test Problem Summary

Table 1 summarizes the test problems. "Depth" and "Branch" are maximum depth and branching factors seen in a set of test runs. For CPC, maximum branching occurs at the root where any word can be placed at the top of any column, but large branching factors of over 400 occur deeper in the tree as well.

4 Experimental Results

All NRPA runs use `Alpha`=1.0 and `N`=100 iterations per level. These values were chosen via a limited set of initial experiments, and appeared to work well across problems.

4.1 Comparing Efficiency of Search

We wish to compare the effectiveness of NMCS and NRPA. To account for the methods' differing computation time and overhead, we compare based on actual execution times measured on a reference machine (3.6GHz Core i7 with 4 physical cores; at most 4 single-threaded jobs run at any time).

A fixed time horizon is used (1000 or 10000 seconds). If a search configuration completes before the time horizon is exhausted, the time horizon is filled with independent restarts of the same configuration. At each point in time, we track the max score seen so far among all restarts of the search configuration. This yields a single timeline for the search configuration; at least 15 independent timelines are generated for each configuration. Multiple independent timelines can then be combined to form a picture of the typical trajectory. This has been used to illustrate that NMCS typically becomes more efficient as nesting level increases [Cazenave, 2009], and NRPA shows a similar trend (Fig. 5). When displaying results, we combine multiple independent timelines at each time t by taking the median (defined here as the mid element in a sorted list, or the larger of the 2 mid elements of an even-length list). When comparing samples, we use a Mann-Whitney U test of statistical significance with $p < 0.05$.

Given a time horizon, we ran NRPA and NMCS at increasing nesting level, until it was no longer possible to complete the majority of a single search run within the horizon. NMCS consistently gave best results at the largest nesting level, and so this is the result we show. In some cases, NRPA can run at a larger nesting level than NMCS in the same horizon. Then, we show NRPA results at both the larger level as well as the same nesting level at which NMCS ran.

Table 2 shows median results at 100 and 1000 seconds for NRPA and NMCS. For all test problems, NRPA results at

1. If the previous move resulted in new lines becoming available: sample 4 such newly-available lines (with replacement) and pick one yielding the largest number of total available moves [Cazenave, 2007].
2. Otherwise, with 90% probability: from available moves which place their new dot at horizontal/vertical/diagonal distance 1 or 2 from the previously-added dot, sample 2 (with replacement) and pick the one yielding more total available moves.
3. Otherwise, sample 6 available moves (without replacement; take all if fewer than 6 exist) and pick one yielding the largest number of newly-available lines as described in 1 above, breaking ties via the largest number of total available moves.

Figure 6: Tuned MorpT Rollout Policy

100 seconds are statistically significantly better than NMCS at 1000 seconds. So, in these tests, NRPA consistently obtains better results than NMCS, even in $1/10$ the time used by NMCS. This was true whether NRPA was run at its highest nesting level, or at the same nesting level as NMCS.

Sample-k NMCS

For CPC, NMCS spends a large fraction of its time iterating through high branching factors near the search tree's root. This may not be the best use of limited time, and it reduces the nesting level that can fit in the time horizon. So, we also test an alternative "Sample-k NMCS" which at each node samples (with replacement) a fixed number k of children, regardless of branching factor. With k sufficiently low, Sample-k NMCS can fit higher nesting levels than NMCS in the same time horizon. Table 3 shows median results for CrossC with a time horizon of 10000 seconds that allows Sample-5 NMCS to run at nesting level 4 and Sample-15 to run at nesting level 3. Sampling enables improved results here compared to standard NMCS at nesting level 2. But NRPA level 3 at 1000 seconds still gives statistically significantly better results than Sample-k NMCS (and standard NMCS) at 10000 seconds. Again, NRPA obtains better results in $1/10$ of the time.

Table 3: CrossC Median Scores with Sampled NMCS

Method	Level	10^2 sec	10^3 sec	10^4 sec
NRPA	3	33;229	34;244	35;247
NMCS	2	31;223	32;229	33;235
Sample-5 NMCS	4	32;228	33;232	34;235
Sample-15 NMCS	3	32;231	33;235	33;248

Tuned NMCS Rollout Policy

Since NRPA adapts its rollout policy, it is natural to compare NMCS using a domain-specific policy. Before running NRPA on MorpT, we manually tuned a rollout policy via small experiments evaluating policy elements' impact on results.

Table 4: MorpT Median Scores with Tuned NMCS

Method	Level	10^2 sec	10^3 sec	10^4 sec
NRPA	4	152	155	160
NRPA	3	153	157	158
NMCS	3	116	130	137
Tuned NMCS	3	137	146	149

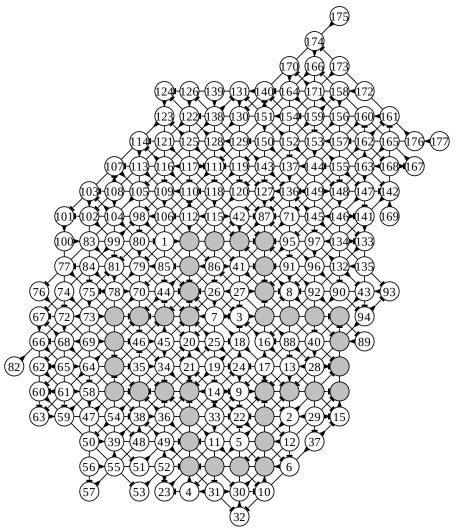
The tuned policy (Fig. 6) is effective in NMCS; Table 4 shows at 10000 seconds it already obtains a median score higher than the best prior MorpT computer search result

Figure 7: Two of the New Record Solutions and Scores

(a) CrossC: 37;259

```
C.....HALIFAX...I.D
OLYMPIA....T....Q.E
L...H.R..COLUMBIA.S
U..BOSTON..A....L.M
M.S.E.F..HONOLULU.O
BATONROUGE.T....I.I
U.J.I.R..L.A.AUSTIN
S.O.X.DOVER..U.A..E
..H......N.REGINA.S
CONCORD..A...U.T...
H.S...E.J..BISMARCK
A...JUNEAU...T.F...
R..T..V.C...SALEM..
L.BOISE.K.P.T.A.A.A
E..R..R.S.I.P.N.D.L
S..O...TOPEKA.S.I.B
TRENTON.N.R.U.I.S.A
O..T......R.LINCOLN
N.MONTPELIER..G.N.Y
```

(b) MorpT: 177 lines



[Akiyama *et al.*, 2010]. But, NMCS with the tuned policy is still much less efficient than NRPA, even though NRPA adapts from scratch and does not benefit from such manual tuning. NMCS, with or without the tuned policy, gives statistically significantly worse results at 10000 seconds than NRPA at 100 seconds (whether NRPA is run at level 3 or 4). So, NRPA obtains better MorpT results than NMCS in $1/100$ of the time, even when NMCS is using a well-tuned policy.

4.2 Results of Longer Runs

We ran NRPA over longer times, to compare NRPA's best solutions to the best previously published solutions for the test problems. At the nesting levels selected, approximate per-run reference machine time is 1 hour for MorpD, 15 hours for CrossP, 24 hours for CrossC, and 1 week for MorpT. As a rough comparison, best previous results were reported as taking 38 days for a MorpT run [Akiyama *et al.*, 2010] and 10 days for a MorpD run on a single machine [Cazenave, 2009].

Table 5: NRPA Results of Longer Runs

	Level	Runs	Median	Max(Rep)
MorpT	5	28	171	177 (2)
MorpD	4	40	82	82 (25)
CrossC	4	40	35;249	37;259 (1)
CrossP	4	40	27;234	28;238 (3)

Table 5 has results. For MorpT and MorpD, even the median is an improvement over the best previously published solutions. For CrossC and CrossP, the maximum is an improvement over the best previous (contest-winning) solutions (see Table 1). Fig. 7 has the new CrossC and MorpT records.

4.3 Distribution of NRPA Results

Fig. 8 has a distribution of results from NRPA nesting levels 0-5, for MorpT. Nesting level n required roughly 100x the time of $n-1$, so on the logarithmic vertical axis we can roughly conclude nesting level n is more efficient than $n-1$ if its cumulative histogram is over two decades higher. Nesting levels 1-3 show especially large efficiency gains.

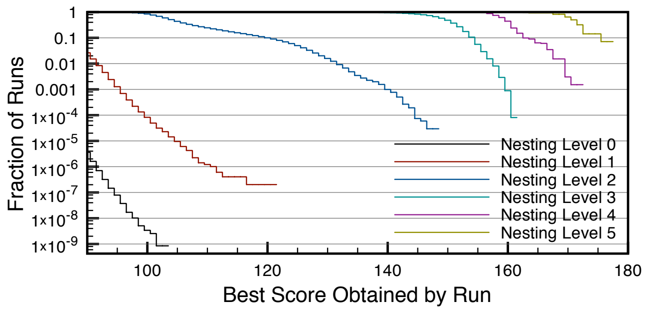


Figure 8: NRPA Cumulative Histograms for MorpT

4.4 Permutations and Hybrids

We examine the composition of intermediate solution sequences obtained by NRPA. From over 200 MorpT runs with nesting level 3, we gather each tie and improvement obtained at level 3 during its 100 iterations. For each, we examine the codes returned by code() for its sequence of actions, and categorize each code as one of: *Prefix* for the initial segment that exactly matches the previous solution sequence's initial segment (note NMCS strictly increases this through a level's iterations, but NRPA need not); *Permutation* matches the previous solution, but in a permuted order rather than as part of Prefix; *Hybrid* does not match the immediately previous solution, but does match an older solution at this level; and *New* were not used in any previous solution at this level.

Table 6: NRPA Intermediate Solution Content for MorpT

	Prefix	Permutation	Hybrid	New
Ties	0.1%	89.7%	8.3%	1.9%
Improvements	0.1%	65.0%	14.9%	20.0%

Table 6 has results. NRPA freely explores permutations, with little of the initial segment staying the same. Solutions also have substantial Hybrid content; NRPA's policy retains some emphasis on codes from older solutions (not just the current best). Hybrid content is higher in improvements than ties, suggesting it can contribute usefully to progress.

5 Conclusion

We have presented NRPA, an MCTS algorithm that uses gradient ascent on its rollout policy to navigate search. NRPA yields substantial search efficiency improvements as well as new record solutions on our test problems. NRPA is the first computer search method to improve upon a human-generated Morpion Solitaire record that had stood for over 30 years.

Ongoing work includes more complex applications, enabling code() to return a feature vector, and parallelization.

Acknowledgments

Thanks to Mark Land. This work was supported in part by the DARPA GALE project, Contract No. HR0011-08-C-0110.

References

[Akiyama *et al.*, 2010] H. Akiyama *et al.* Nested Monte-Carlo search with AMAF heuristic. In *TAAI*, 2010.

[Bertsekas, 1997] D. Bertsekas. Differential training of rollout policies. In *Allerton Conf.*, 1997.

[Bjarnason *et al.*, 2007] R. Bjarnason *et al.* Searching solitaire in real time. *ICGA J.*, 2007.

[Boyer, 2010] C. Boyer. *Science & Vie*, page 144, Nov. 2010.

[Boyer, 2011] C. Boyer. http://morpionsolitaire.com, 2011.

[Bruneau, 1976] C.-H. Bruneau. *Science & Vie*, April 1976.

[Cazenave, 2007] T. Cazenave. Reflexive Monte-Carlo search. In *CGW*, 2007.

[Cazenave, 2009] T. Cazenave. Nested Monte-Carlo search. In *IJCAI*, 2009.

[Cazenave *et al.*, 2009] T. Cazenave *et al.* Monte-Carlo bus regulation. In *ITSC*, 2009.

[Coulom, 2007] R. Coulom. Computing Elo ratings of move patterns in the game of Go. In *CGW*, 2007.

[CrossC, 2006] *GAMES Magazine*, page 76, August 2006. Winning solution: page 93, December 2006.

[CrossP, 1994] *GAMES Magazine*, page 8, June 1994. Winning solution: page 67, October 1994.

[de Mesmay *et al.*, 2009] F. de Mesmay *et al.* Bandit-based optimization for library performance tuning. *ICML*, 2009.

[Demaine *et al.*, 2006] E. D. Demaine *et al.* Morpion Solitaire. *Theory Comput. Syst.*, 2006.

[Fern *et al.*, 2003] A. Fern *et al.* Approximate policy iteration with a policy language bias. In *NIPS*, 2003.

[Finnsson *et al.*, 2010] H. Finnsson *et al.* Learning simulation control in GGP agents. In *AAAI*, 2010.

[Garey and Johnson, 1979] M. Garey and D. Johnson. *Computers and Intractability*. 1979.

[Gelly *et al.*, 2007] S. Gelly *et al.* Combining online and offline knowledge in UCT. In *ICML*, 2007.

[Ginsberg *et al.*, 1990] M. Ginsberg *et al.* Search lessons learned from crossword puzzles. In *AAAI*, 1990.

[Larranaga *et al.*, 2002] P. Larranaga *et al.* *Estimation of Distribution Algorithms*. Kluwer, 2002.

[Méhat *et al.*, 2010] J. Méhat *et al.* Combining UCT and NMCS for single-player GGP. *IEEE TCIAIG*, 2010.

[Nakhost *et al.*, 2009] H. Nakhost *et al.* Monte-Carlo exploration for deterministic planning. *IJCAI*, 2009.

[Rimmel *et al.*, 2011] A. Rimmel *et al.* Optimization of the Nested Monte-Carlo Algorithm on the Traveling Salesman Problem with Time Windows. In *Evostar*, 2011.

[Saffidine *et al.*, 2010] A. Saffidine *et al.* UCD: Upper confidence bound for directed acyclic graphs. In *TAAI*, 2010.

[Silver *et al.*, 2009] D. Silver *et al.* Monte-Carlo simulation balancing. In *ICML*, 2009.

[Silver *et al.*, 2010] D. Silver *et al.* Monte-Carlo planning in large POMDPs. In *NIPS*, 2010.

[Tesauro *et al.*, 1996] G. Tesauro *et al.* On-line policy improvement using Monte-Carlo search. In *NIPS*, 1996.

[Veness *et al.*, 2011] J. Veness *et al.* A Monte-Carlo AIXI approximation. *JAIR*, 2011.