Binarized Monte Carlo Search for Selection Problems

Matthieu Ardon[1], Yann Briheche[1], and Tristan Cazenave[2]

[1] Thales Research & Technology - Palaiseau, France
[2] LAMSADE, Université Paris Dauphine, PSL, CNRS - Paris, France

Abstract. In this paper, we consider adaptations of Monte Carlo Search methods on binary decision trees where actions are simulated using heuristics and where choices are made deterministically or stochastically. We explain how these adaptations are fitted for combinatorial problems such as element selection problems in order to compete with other approximate resolution methods such as metaheuristics. We present results on a theoretical problem (Set Covering) and on an applied problem (Pulse Repetition Frequency Selection) with different simulation heuristics. We then discuss the usefulness of these new methods based on the characteristics of the problems and on the quality of the simulation heuristics used to construct the decision tree.

Keywords: Tree Search · Monte Carlo Search · Set Cover Problem · Radar · Decision Trees · Binary Choices · Heuristic · Simulation

After a general presentation of the selection problems, we detail the two problems considered. We then introduce Binarized Monte Carlo Search, based on Monte Carlo Search on particular decision trees. Adaptations of two different methods are developed. Finally, we present the experimental results obtained with the new methods on our selection problems and discuss their interests.

1 Selection Problems

Selection problems arise as problems in which a subset of elements must be selected from a given set, usually under given constraints, in order to optimize a given objective function.

Due to the size of the initial set of candidates and the large number of possible combinations among them, these problems often manifest as combinatorial optimization problems. Exact solution methods are therefore not always suitable for these problems. Approximate solution methods can be an efficient alternative.

A greedy algorithm can quickly and iteratively produce an approximate solution to a selection problem. Its basic idea is to make a locally optimal choice at each step, without considering the consequences of this choice on future steps, with the hope of finding a globally optimal solution. As long as the solution is not complete, the algorithm assigns a *greedy score* to all valid candidates based on a computationally inexpensive function. The validity of those candidates depends on constraints preventing specific combinations of candidates in some selection problems. The candidate with the highest score is then added to the solution and removed from the set of available candidates if a solution cannot be formed by the same candidate more than once. Greedy algorithms are easy to implement but can provide less than optimal solutions on certain instances.

Another alternative is to use other heuristics, metaheuristics or tree search procedures. Combinations of these different resolution methods are also increasingly being considered, this is the case in this paper.

1.1 Set Cover Problem

The Set Cover Problem (SCP) is a classic optimization problem in which the goal is to select, from a collection of subsets whose union equals the universe, the smallest number of subsets such that their union is also equal to the universe. This problem is known to be NP-hard [13], meaning that there is no known efficient algorithm to solve it optimally in polynomial time.

One variant of this problem is the Weighted Set Cover Problem, in which every set is assigned a positive weight, indicating its cost, and the objective is to identify a set cover which minimizes the sum of the costs of the chosen sets.

Formulation. The problem can be formally defined as follows:

$$\text{minimize} \sum_{j \in J} c_j x_j \tag{1}$$

$$\text{under the constraints} \sum_{j \in J} a_{ij} x_j \geq 1, \forall i \in I \tag{2}$$

$$\text{where } x_j = \begin{cases} 1 & \text{if subset } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{and } a_{ij} = \begin{cases} 1 & \text{if item } i \text{ is covered by subset } j \\ 0 & \text{otherwise} \end{cases}$$

Simply put, the candidates j are binary column vectors, and the set to cover is initially a zero vector formed of as many items/rows/cells as each candidate column vector.

Chvatal's Greedy Procedure. The classic greedy algorithm follows the ideas of Chvatal's algorithm [9], where the next candidate is selected based on its *effective cost*. This is the score function used to determine which candidate to add incrementally.

With I_j representing the set of still uncovered rows that column j covers and J_i representing the set of candidates which cover row i, the candidate is chosen at each step from the following formula:

$$\arg\min_{j \in J} \frac{c_j}{|I_j|} \tag{3}$$

Surprisal-Based Greedy Heuristic. The previous procedure is simple and intuitive but can be improved to assign a more relevant score to candidates based on the characteristics of all others. The Surprisal-Based Greedy Heuristic is derived from Information Theory and can yield better results when applied to the SCP than the Chvatal's procedure [1]. Using the same notation, the candidate is chosen according to:

$$\arg\min_{j \in J} \frac{c_j}{|I_j|} \prod_{i \in I_j} \frac{|J_i| - 1}{|J_i|} \tag{4}$$

Thus, a candidate column is always chosen if it is the only one covering a given row, since one of the terms in the product is zero. However, if alternative candidates exist for all the rows covered by a candidate column, the score of this column will be higher and tend toward the Chvatal's greedy score.

1.2 Pulse Repetition Frequency Selection Problem

A major topic in radar engineering is target detection. Pulse Doppler radars can determine the range and radial velocity of a target. They use multiples series of pulses, called bursts, to achieve detection, each requiring the selection of its Pulse Repetition Frequency (PRF, representing the number of pulses transmitted by the radar each second).

The periodicity of pulse Doppler radars cause several problems:

- *range blind zones*: the radar cannot receive a pulse when another is emitted.
- *range ambiguities*: the radar use detection delays to determine the target range, but has no way to identify the original pulse, and thus only determine range within a modulo.

A burst with a given PRF is characterized by blind zones and ambiguities on the range axis. Sending a waveform made of an ordered sequence of several bursts associated with different PRFs can solve the aforementioned problems:

- blind zones of different bursts can compensate each other if they do not overlap.

– different bursts measure range with different modulos. According to the Chinese Remainder Theorem, if the Pulse Repetition Intervals (PRIs, the inverse of PRFs) have a high enough lowest common multiple, then detection of the same target between both bursts will overlap on a unique range below the maximum radar detection range, yielding the actual range of the target. Ambiguity removal, also called *decodability*, thus requires the prohibition of some PRFs combinations [12].

In this paper, we will consider that all bursts have a constant and identical duration regardless of the PRF of each burst. This means that the number of pulses in each burst is different and deduced from its PRF and that constant duration.

Methods for resolving this problem from the literature have been tested with evolutionary algorithms [12] for the selection of PRFs for bursts with constant numbers of pulses, as well as with metaheuristics such as simulated annealing [2] for the selection of PRFs for bursts with constant emission durations.

We use an M of N scheme: we select a set of N bursts and target detection is achieve if M bursts can detect a given target. In this paper, all the tests have been performed with the same scheme. We use a recursive approach to compute the overall probability when adding a new burst [4]:

$$P(m,n) := p_n \cdot P(m-1,n-1) + (1-p_n) \cdot P(m,n-1) \tag{5}$$

with p_n the detection probability of the n-th burst, $P(m,n)$ the overall detection probability of m among n bursts, with the following initial conditions:

$$P(0,0) = 1 \text{ and } \forall m,n \geq 1, P(m,0) = P(-1,n) = 0 \tag{6}$$

2 Binarized Monte Carlo Search

In the field of operations research, tree search methods can explore the search space systematically in a deterministic or stochastic manner. Different traversal strategies can be considered, exhaustive or not, depending on the complexity and the restrictions of the problem. The Limited Discrepancy Search [11] (LDS) lends itself well to problems with a clear heuristic on the possible actions that can occur. Indeed, it finds a solution deterministically by following a heuristic while allowing a limited number of deviations from this heuristic to encourage the exploration of alternative choices. Monte Carlo Tree Search uses random sampling for part of its exploration and statistical analysis which allows it to deepen the research in the most promising parts of the tree. Originally developed for game-playing programs, especially those with vast search spaces like Go [10], Monte Carlo Tree Search has since found applications in various fields such as optimization.

2.1 Nested Monte Carlo Search

Recursive approaches on Monte Carlo Search methods seems promising, especially for single-player problems [8], whose objectives can be similar to selection problems. The addition of nested search procedures allows in some cases the intensification of the search in parts of the decision tree (Nested Monte Carlo Search), and in other cases prevent a premature convergence caused by a learning that slows down the exploration too quickly (Nested Rollout Policy Adaptation).

Nested Monte Carlo Search. The Nested Monte Carlo Search (NMCS) is a method that performs recursive improvements of playouts using nested levels [5]. At level zero, the playouts are random. At higher levels, they are guided by results obtained at lower levels.

Nested Rollout Policy Adaptation. The Nested Rollout Policy Adaptation (NRPA) is a method that learns a policy which presents itself as a weight vector influencing the probabilities of choosing each candidate [14]. NRPA has set new world records in Morpion Solitaire and Crosswords Puzzles. The nested levels are now associated to the best sequence found at this level. During the learning (called the adaptation) of the policy, the weights of candidates present in the best sequence are incremented, and the others are reduced. The learning rate α is an hyperparameter controlling the update rate of these weights. The probability to choose a child node *child* created from a parent node *parent* with the selection of a given action *action* is:

$$p_{child} = \frac{e^{policy(child_{action})}}{\sum\limits_{brother \in \{children(parent)\}} e^{policy(brother_{action})}} \tag{7}$$

Calls to lower nested levels are limited to a given number of iterations. Nested Rollout Policy Adaptation with Limited Repetitions [7] (NRPALR) ends the playouts of a level when the best performance remains the same for a given number of times. After a lower level call, the best sequence is returned to the upper level and the upper level policy is updated then passed again to a lower level call. Calls at level 0 use stochastic playouts.

Generalized Nested Rollout Policy Adaptation [6] (GNRPA) is another way to integrate a heuristic or problem feature to guide the search in the tree. In this method, computation of probabilities for choosing a node accounts for features of the chosen node state. However this requires computing for each node its state and features.

NRPA, GNRPA or GNRPALR perform a search that is partially random and partially directed. They allow the integration of multiple heuristics during initialization phase, and during search phase. In the search phase, they guide exploration at a chosen speed (learning rate) with regular updates of the probabilities of choosing children node (adaptation of the initial policy).

2.2 Adapted Representations of Decision Trees

Decision trees can be well-suited representations for solving selection problems with the following assumptions. A node represents a solution formed by the set of candidates used in the sequence leading to that node. A branch represents the choice of a specific candidate from the available set of candidates. Each candidate is an action mapping the state of the parent node to the state of the child node, through the branch connecting both. The root of the tree is the neutral state. Regardless of the chosen representation, the tree is pruned at the end of every sequence when no candidate can comply with the problem constraints: for the SCP the tree is pruned when the state of a node is entirely covered, and for the PRF Selection Problem when either the desired number of bursts is reached or when there is no remaining burst which can comply with decodability.

Classic exhaustive tree search methods and some less parameterizable Monte Carlo Search methods do not always require the computation of the state of each simulated node to traverse the tree and retain the best sequences. Sometimes, only evaluating the leaves is sufficient. However, in the case of constrained problems, states must be computed each time. For example, to solve the Set Cover Problem (SCP), it is necessary to check at each state whether the covering condition has been satisfied, in order to stop the search, return the result corresponding to the traversed sequence, and prune the tree.

Thus, reducing the number of explored states tends to reduce the overall computational complexity. A tree with a high branching factor requires the computation of many states at each step. Long sequences will also increase the number of computed states. While the length of the sequences cannot be reduced, we can try to use more efficient representations for solving selection problems. These representations should have a low branching factor and should not introduce biases that favor certain candidates over others based solely on their initial ordering or on an arbitrary position in the tree.

Binary Decision Trees. All branches leading to a parent node's children correspond to a specific action and its associated candidate. The parent node has two children: a first node where the candidate is rejected (with the same state as the parent node) and a second where the candidate is added to the solution (the state is thus the parent node state with the addition of the candidate).

Simulation Heuristics. The simulation of the next nodes takes into account the construction of the current sequence, and greatly impact the quality of research in binary trees.

Nodes can be simulated according to a heuristic which will determine an interesting action to choose among all available actions. Random simulations/moves can quickly produce solutions. Greedy simulations are likely to yield better solutions.

In a binary decision tree, a traversal ignoring each action would be possible to favor the exploration of actions that are locally less interesting, but may be globally interesting. Simulations can also be based on domain-specific rule-based heuristics in order to intensify the quality of returned candidates. Some heuristics can perform better than another depending on the problem.

In the simulation function, the heuristic assigns a score to each possible action, in the same ways as the previously cited *greedy score*. The chosen action is the one with the best score. If multiple actions achieve the same best score, one can selected either through a predefined order or either randomly. As we will see, the latter can introduce a stochastic aspect to a deterministic method.

The binary tree can be constructed with an initial ordering of the actions. However, this will significantly decrease the effectiveness of the heuristic and thus the quality of the search. Despite the need for greater computing resources, we therefore favored a call to the heuristic each time a node is traversed for the first time.

2.3 Dynamic Binarized Nested Monte Carlo Search (DBNMCS)

We adapted NMCS to perform the search partly on the binary decision tree, as presented above, and partly through an iterative heuristic used to complete solutions, see Algorithm 5.

The search occurs in two steps, each based on a heuristic which finds the best action to consider in the rest of the current sequence, see Algorithm 4. In our experiments, we used the same heuristic for both steps, but different heuristics could be used for each step.

In the first step, we search through a nested level on a binary tree (starting from the root), where each node, both simulated by the first heuristic, represents a choice to add a candidate to the solution or not. For each child node of a given parent node, the second step will perform a deterministic playout by completing the solution using the second step heuristic. We select the node with the best solution among the two children. The first step starts again from that node.

Recursively, the two steps are chained. Once we reach a zero nested level, we perform two deterministic playouts. We store the node with the best solution among the two as a starting point for an in-depth search (binary search then playout), to try to find a better solution from that starting point.

Since each of the 2 branches is always traversed in the first step, and solutions are obtained using in the second step using an deterministic heuristic, there is no randomness in the algorithm, which is therefore completely deterministic.
Furthermore, after reaching the zero nested level, the playout are performed until a leaf and then, the first step on a binary tree takes over from the node corresponding to the best playout. The tree is therefore dynamic.

To summarize, the Dynamic Binarized NMCS involves a two-step process. In the first step, a heuristic-based search is performed on the binary tree, exploring different sequences and forming partial solutions. In the second step, in the form of local searches, playouts are conducted at the lowest nesting levels of the tree to complete these partial solutions. The performances of the resulting solutions are compared. The process repeats recursively until a leaf node is reached, potentially refining the solution by considering variations of the initial choices. The entire process is deterministic and dynamic. Ultimately, DBNMCS allows for a non-exhaustive search of the decision tree guided by the most interesting choices to focus on the parts of the tree that appear most promising. It does not need a lot of hyperparameter tuning or precise information about the problem to initially guide the search with complex heuristic.

2.4 Binarized Nested Rollout Policy Adaptation (BNRPA)

Another interesting possibility for exploring binary decision trees with heuristic-based simulations are NRPA methods.

Unlike NMCS methods, playouts here always start at the root node, see Algorithm 2. Each node in the binary tree representing whether a candidate is added to the solution or not. We also use heuristic-based simulations to select the available action with the best score, see Algorithm 4.

A policy vector is initialized before the start of the search, as is done in classic NRPA. However, since the representation is now a binary tree where half of the branches represent the choice to ignore a candidate, the encoding of the policy needs to be modified. Thus, each candidate is now linked to a binary policy. One component represents the choice to select it and the other represents the choice not to select it. Before the start of the search, if the policy is initialized uniformly, its values no longer depend on the number of candidates: for each candidate, it's set to 1/2 for the component representing the choice to select it and similarly for the one representing the choice not to select it.

Like in classic NRPA, we copy the policy from higher levels to lower levels, and we adapt the policy by updating its values after each playout. This improves the probabilities of the choices which might be proposed again in subsequent playouts on the same nesting level. In a binary tree, the best solution found so far is made of choices of some actions and refusal of other actions. Depending on the choice or refusal of each action in the best found solution among all playouts of a given nested level, we will increment its associated policy for selection or its associated policy for refusal and decrement by the same factor the policy that was not incremented, see Algorithm 3. The learning rate α controls the speed and intensity of the update of the policy.

Note that at the end of any nesting level, the policy is not kept, and does not become the initial policy of the higher level. Otherwise, it would prevent exploration due to adaptation occurring in the same way to favor the same choices. Instead, only the best solution is returned to the higher nested level, and the policy of the higher level is adapted based on the best solution found in the lower level.

The number of calls to the lower nested level per level is set in our first implementation of Binarized NRPA (BNRPA). Another approach is to implement a stop criteria on this level based on the number of times the exact best performance is returned, hinting that further improvement is unlikely. This is done in BNRPA with Limited Repetitions (BNRPALR), see Algorithm 1.

We can construct the tree statistically or dynamically. In the first case, the root is always the same. The tree is built gradually as the playouts progress. Playouts increasingly pass through existing branches and nodes as the probabilities of traversing those branches increase. These branches become mandatory paths at the top of the tree since all available paths have already been simulated. In the second case, the root is reset with each new playout. The child nodes must be simulated again at each depth until the playout stops at a leaf. Thereby, the sequences are dynamic and can differ between playouts when multiple actions obtain the same heuristic score, since we do not always choose the same action. This dynamic construction can favor exploration and diversification. However, reaching the repetition limit on the best performance can become harder, since finding the same solution twice becomes unlikely and some problems will rarely have the same performance for two different solutions.

Empirical evidences indicates that BNRPALR performs better than BNRPA on static binary trees, and that BNRPALR performs better on static trees rather than dynamic ones, for the same computation time. The variance observed on different runs with BNRPALR is lower than the variance of BNRPA.

BNRPA with heuristic-based simulations uses a heuristic to choose the actions of the following nodes. Classic NRPA allows for the integration of another heuristic or a prior in policy initialization. This is more difficult to do with BNRPA due to the binary policy. GNRPA also allows the integration of additional information to guide the search by modifying the node choice probabilities based on a weighted prior. With BGNRPA, it might be interesting to construct the prior based on the average of a statistic across all remaining candidates to be considered for the node that does not add a candidate. For the other node, it would be the statistic specific to that candidate. Yet, with BGNRPA, actions are not simulated in the same order in each sequence and biases then cannot be precomputed. Nevertheless, this step is often the most time-consuming with GNRPA. However, this method still represents a promising approach for improving BNRPA because it allows for the integration of additional information to guide the search.

3 Experimental Results

We compare new Binarized Monte Carlo Search methods (DBNMCS and BNRPALR) with Limited Discrepancy Search (LDS) using the same simulations heuristics and a discrepancy parameter set to 7 (SCP) or 8 (PRF Selection Problem).

3.1 Weighted Set Cover Problem

OR-library provides datasets of problem instances for benchmarking combinatorial optimization, including the Weighted Set Cover Problem [3].

Each problem instance is characterized by the number of columns candidates (or variables) $|J|$, the number of cells to cover (or constraints) $|I|$, and the density of the covered cells by the candidates. A weight has been assigned to each candidate, and the objective is to minimize the sum of the weights of a subset of chosen candidates.

We perform tests on OR-Library instance sets "*4*" to "*NRH*", with the number of instances in each set indicated in Table 1 under the column |Set|.

We used DBNMCS and BNRPALR with both Chvatal's Greedy Procedure (CGP) and Suprisal-Based Greedy Heuristic (SBH) as simulation heuristics. Each had 25 runs, under the same computation time limit. For BNRPALR, we set the learning rate at 0.75, the nested level at 9 and the repetition limit at 5, with a uniform initial policy. For DBNMCS, we set the nested level at 4. All the results are displayed and compared to the optimal or best known score (BS) on Table 3. The stars indicates that at least one run has reached the best performance (BS) with the corresponding method and simulation heuristic. The GAP compares the average of the obtained performances from the 25 runs (AVG) to the BS, such as:

$$GAP = 100 \times \frac{AVG - BS}{BS} \tag{8}$$

The smaller the average gap across a set of instances, the better the method. Table 1 displays the average gap of each method over the sets of instances, and the characteristics (number of variables and constraints, density) of each set.

Table 1. Comparison of Binarized Monte Carlo Search methods with CGP heuristic on the sets of instances of the SCP from OR-library

Set	**|Set|**	**|I|**	**|J|**	**Density**	**GAP averages**	
					DBNMCS	BNRPALR
4	10	200	1000	2	4.36	**0.87**
5	10		2000		5.43	**1.25**
6	5		1000	5	4.04	**1.18**
A	5	300	3000	2	6.62	**2.14**
B	5			5	3.19	**1.19**
C	5	400	4000	2	8.56	**5.31**
D	5			5	3.45	**1.39**
E	5	50	500	20	**0**	**0**
NRE	5	500	5000	10	1.88	**1.05**
NRF	5			20	2.97	**2.05**
NRG	5	1000	10000	2	**9.91**	22.5
NRH	5			5	**7.17**	9.99

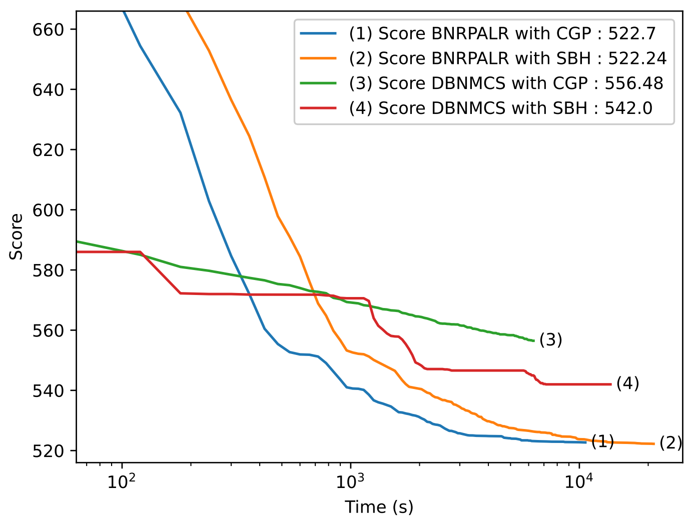

Fig. 1. Evolution of the best performance found over time for instance *scp43* from OR-Library (BS = 516)

Figure 1 highlights the performances of DBNMCS and BNRPALR methods. Each one was tested with the Chvatal's Greedy Procedure and the Surprisal-Based Heuristic as simulation heuristics, with a number of calls set at 3,000,000. 100 runs were performed for each configuration on the *scp43* instance. It's a moderately sized and low-density instance which proved to be quite discriminative in the various tests.

3.2 PRF Selection Problem

We performed tests on two different PRF Selection Problem instances, for which we aim to maximize the detection probability at a given target range. We used the same radar parameters (mean power, signal duration, carrier bandwidth...) for both instances, and only changed the detection range, with the first problem (*Instance 1*) at low range (thus being easier to optimize) and the second (*Instance 2*) at long range (thus being harder to optimize). These instances are made up of 140 candidates and the search is stopped as soon as we have added N candidates to our solution in accordance with the chosen M of N scheme.

We use both DBNMCS and BNRPALR with two different simulation heuristics: one based on a simple greedy heuristic (GH) and the other on based an initial random rating (RR). For BNRPALR, we set the learning rate at 0.75, the nesting level at 9 and the repetition limit at 5, with a uniform initial policy. For

DBNMCS, we set the nested level at 8 (more important than the SCP since the sequences are made up of only 6 candidates and therefore much shorter).

Greedy Resolution. The greedy resolution of the SCP relies on additional information unrelated to candidate costs (which is the optimization criterion) such as the number of additional rows that could be covered by a candidate. We can still use a basic greedy process even when such information is not available: at each step, we compute the solutions with each possible candidate (in other words, each PRF compatible with the PRFs already chosen in the solution), and select the one which maximize our optimization criterion, the detection probability.

On *Instance 1*, the basic iterative greedy resolution of the problem gives poor results. On *Instance 2*, they are even worse.

Randomized Rating. Here, the problem lacks additional information that can be used to compute specific priorities for each candidate (such as CGP or SBH). But there are other ways to determine the interest of some candidates over others. Before the optimization, we can assign an interest score to each candidate based on the average performance of randomly sampled solutions that contained that candidate. Those interest scores will help favor certain candidates over others when they achieved the same score for a simulation-based heuristic. It is particularly suitable here because the candidates are not extremely numerous (140). The initial random scoring RR of each candidate $c \in C$ was computed over 3,000 random simulations, by normalizing the sum of the scores they obtained during the runs in which they were selected in the waveform, by this number of runs, as:

$$RR_{c\_norm} = \frac{\frac{\sum\limits_{r \in \{run|x \in run\}} score(r)}{card(\{run|x \in run\})} - \min\limits_{c \in C} RR_c}{\max\limits_{c \in C} RR_c - \min\limits_{c \in C} RR_c} \tag{9}$$

This prior can be combined with the score of the greedy simulation heuristic to weight it according to the initial interest of the candidates (GH & RR).

The results are compared in Table 2 with those of Simulated Annealing (SA), obtained using *scipy.optimize.dual_annealing*, and the Limited Discrepancy Search (LDS).

Table 2. PRF Selection Problem results

		SA	**LDS**			**BNRPALR**			**DBNMCS**		
			GH	RR	GH & RR	GH	RR	GH & RR	GH	RR	GH & RR
Instance 1	AVG	0.913	0.961	0.943	0.960	**0.966**	0.944	0.959	0.964	0.887	0.955
	MAX	0.965	**0.972**	0.969	0.969	**0.972**	0.969	0.971	**0.972**	0.950	0.969
Instance 2	AVG	0.579	0.569	0.592	0.604	0.585	0.588	**0.605**	0.586	0.576	0.594
	MAX	0.608	0.602	**0.614**	**0.614**	0.602	**0.614**	**0.614**	0.607	0.608	**0.614**

4 Discussion

Weighted Set Cover Problem. BNRPALR seems to be more efficient than DBNMCS on most instances for the same calculation time, see Table 1.

A possible explanation is the search getting lost in the resolution of instances from the NRG and NRH sets which are very large and not very dense. Since each candidate does not cover a lot of rows, the sequences are longer before reaching the coverage condition. Escpecially for BNRPALR, whose entire search is carried out on a binary tree with an initial non-negligible probability of ignoring actions. In contrast, DBNMCS is better because playouts at nested level zero are shorter and allow testing of many small variations when constructing solutions.

Moreover, regardless of the instance, BNRPALR takes more time to find a better solution than a basic iterative greedy algorithm, whereas DBNMCS finds quickly a close solution from its first traversed sequence, see Figure 1.

However, for all more moderate instances, BNRPALR give much better results, see Table 3. It often comes close to, and sometimes even achieves, the best-known results for several instances. Generally the most efficient policy learning and the most frequently optimal results are obtained on the densest instances (sets 4, 5, 6, B, D, E, NRE, NRF).

Standard deviations in Table 3 are much larger with BNRPALR due to its stochastic nature (each branch choice depends on a probability linked to the simulated action policy). Non-zero standard deviations for DBNMCS, despite it being deterministic, are due to random choice when several candidates have the same score in the simulations. Additionally, these standard deviations are bigger with CGP heuristic than with SBH, which has a more complex formula. Thus two different solutions are less likely to have the same score.

Despite similar computational complexities, SBH takes a bit more time than CGP. Under the same computation time limit, CGP is often more efficient at it allows simulation of more nodes, and thus exploration of more solutions, see Table 3. Within the same number of calls to the simulation heurisitc, BNRPALR-SBH outperforms BNRPALR-CGP, see Figure 1.

For DBNMCS, SBH is significantly better than CGP. Indeed, the actions will be chosen more scrupulously with SBH and therefore the numerous variations around solution constructions lead to better performance. The evolution of the DBNMCS performance with CGP is slow but regular, see Figure 1. The evolution of the performance of the same method with SBH is sometimes very quickly improved thanks to the more efficient simulation heuristic.

Pulse Repetition Frequency Selection Problem. On its own, greedy iterative resolution performs poorly on the PRF Selection Problem, especially on *Instance 2*. Binarized Monte Carlo Search methods can greatly improves the results.

According to the SCP results, BNRPALR is more effective when an efficient simulation heuristic is available. This is not the case for this problem, and this explains the small difference between the performances of DBNMCS and BNRPALR, see Table 2. According to the conclusion of the original paper [5], NMCS can be used even without a good heuristic to guide the search, which is the case with the PRF Selection Problem.

The search can also be improved with an initial randomized rating as the simulation heuristic. This approach can be suitable even without an effective greedy heuristic.

LDS also performs better because the size of the instances lends itself better to this type of search. LDS beats Simulated Annealing but is not overall better than Binarized Monte Carlo Search.

In summary, Dynamic Binarized NMCS requires little hyperparameter tuning and is therefore suitable when the knowledge of the problem is limited or when there is no existence of a good simulation heuristic that is not too computationally intensive. Binarized NRPA(LR) is suitable when a good result needs to be achieved over a slightly longer period of time, or when additional information from the data can be used to help improve this result continually. Possible improvements could come from GNRPA [6] and should primarily improve its robustness (variance).

DBNMCS and BNRPA(LR) significantly outperform the basic iterative greedy algorithm since they use it as a simulation heuristic for choosing actions while being more extensive on the exploration. DBNMCS will directly surpass this result, while BNRPA(LR) may take some time before exceeding it.

A good simulation heuristic applied to these methods significantly improves the final result. However, there is a tradeoff to find between the efficiency of the heuristic and its computational cost. Here, for an equivalent computation time, the simulation with the least effective heuristic (and consequently the fastest) will be preferred with BNRPA(LR).

The characteristics of the instances can also influence the choice of the resolution method. On instances of reasonable sizes ($<\sim$ 5000 candidates) and balanced (consistent density for the SCP), BNRPALR is very effective. On more aberrant instances, DBNMCS is more appropriate. Furthermore, LDS is worth considering on small instances.

A deterministic method with a more discriminative simulation heuristic will improve the robustness (variance) of the optimization.

5 Conclusion

In this paper, we have presented adaptations of two Monte Carlo Search methods, Dynamic Binarized Nested Monte Carlo Search (DBNMCS) and Binarized Nested Rollout Policy Adaptation (BNRPA) including its variants BNRPALR and BGNRPA. With the use of binary decision trees and heuristic-based simulations, these adaptations are suitable for solving selection problems and potentially other optimization problems . We have shown that it's the case for the Weighted Set Cover Problem, where the most relevant method managed to find the best score on certain suitable instances, and for the Pulse Repetition Frequency Selection Problem, where we beat the results of other tested methods. In addition , we have also highlighted and explained the interest of each method depending on the characteristics and knowledge of the problem and instances to be solved and the quality of the heuristic on which the simulation is based. Further work may focus on accelerating the playouts of these methods and improving DBNMCS's performance and BNRPA's variance.

References

1. Adamo, T., Ghiani, G., Guerriero, E., Pareo, D.: A surprisal-based greedy heuristic for the set covering problem. Algorithms **16**(7) (2023). https://doi.org/10.3390/a16070321, https://www.mdpi.com/1999-4893/16/7/321
2. Ahn, S., Lee, H., Jung, B.: Medium prf set selection for pulsed doppler radars using simulated annealing. In: 2011 IEEE RadarCon (RADAR). pp. 090–094 (2011). https://doi.org/10.1109/RADAR.2011.5960505
3. Beasley, J.: An algorithm for set covering problem. European Journal of Operational Research **31**(1), 85–93 (1987). https://doi.org/https://doi.org/10.1016/0377-2217(87)90141-X, https://www.sciencedirect.com/science/article/pii/037722178790141X
4. Brunner, J.: A recursive method for calculating binary integration detection probabilities. IEEE Transactions on Aerospace and Electronic Systems **26**(6), 1034–1035 (1990). https://doi.org/10.1109/7.62256
5. Cazenave, T.: Nested monte-carlo search. In: IJCAI International Joint Conference on Artificial Intelligence. pp. 456–461 (2009)
6. Cazenave, T.: Generalized nested rollout policy adaptation. In: Monte Search at IJCAI (2020)
7. Cazenave, T.: Generalized nested rollout policy adaptation with limited repetitions. arXiv preprint arXiv:2401.10420 (2024)
8. Cazenave, T., Teytaud, F.: Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In: Learning and Intelligent Optimization - 6th International Conference, LION 6. pp. 42–54 (2012)
9. Chvatal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research **4**(3), 233–235 (1979), http://www.jstor.org/stable/3689577
10. Gelly, S., Silver, D.: Monte-carlo tree search and rapid action value estimation in computer go. Artif. Intell. **175**(11), 1856–1875 (2011). https://doi.org/10.1016/j.artint.2011.03.007, https://doi.org/10.1016/j.artint.2011.03.007
11. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In: IJCAI. pp. 607–615. Morgan Kaufmann (1995)
12. Hughes, E., Alabaster, C.: Medium prf radar prf optimisation using evolutionary algorithms. In: Proceedings of the 2003 IEEE Radar Conference (Cat. No. 03CH37474). pp. 192–197 (2003). https://doi.org/10.1109/NRC.2003.1203401
13. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US, Boston, MA (1972). https://doi.org/10.1007/978-1-4684-2001-2_9, https://doi.org/10.1007/978-1-4684-2001-2_9
14. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo Tree Search. In: IJCAI. pp. 649–654 (2011)

Algorithm 1: BNRPALR(level, policy)

```
if $level = 0$ then
  return BNRPALR_Playout($policy$)
else
  $bestScore \leftarrow -\infty$
  $repetitions \leftarrow 0$
  while $repetitions \leq R$ do
    $(score, sequenceActions) \leftarrow$ BNRPALR($level - 1, policy$)
    if $score = bestScore$ then
      $repetitions \leftarrow repetitions + 1$
    end if
    if $score > bestScore$ then
      $repetitions \leftarrow 0$
      $bestScore \leftarrow score$
      $bestSequenceActions \leftarrow sequenceActions$
    end if
    $policy \leftarrow$ BNRPALR_Adapt($policy, bestSequenceActions$)
  end while
  return $(bestScore, bestSequenceActions)$
end if
```

Algorithm 2: BNRPALR_Playout(policy)

```
$node \leftarrow root$
$sequenceActions \leftarrow \{\}$
while true do
  if $state(node)$ is a complete solution then
    return $(score(state(node)), sequenceActions)$
  else
    if $node$ has not yet been traversed then
      $action \leftarrow$ Simulate($node$)
      $state(child(node, 0)) \leftarrow state(node)$
      $state(child(node, 1)) \leftarrow state(node) \cup action$
    else
      $action \leftarrow action(children(node))$
    end if
  end if
  $sequenceActions \leftarrow sequenceActions + [action]$
  $z \leftarrow \exp(policy[(action, 0)]) + \exp(policy[(action, 1)])$
  $child \leftarrow$ choose child $i$ with probability proportional to
          $\exp(policy[(action, i)])$
  $node \leftarrow child$
end while
```

Algorithm 3: BNRPALR_Adapt(policy, sequenceActions)

```
$policy' \leftarrow policy$
for $num \leftarrow 1$ to length($sequenceActions$) do
  $action \leftarrow sequenceActions[num]$
  if $action$ is not selected then $policy'$ $[(action, 0)]$ += $\alpha$
  elif $action$ is selected then $policy'$ $[(action, 1)]$ += $\alpha$
  $z \leftarrow$ exp($policy[(action, 0)]$) + exp($policy[(action, 1)]$)
  $policy'$ $[(action, 0)]$ -= $\alpha \times$ exp($policy[(action, 0)]$) / z
  $policy'$ $[(action, 1)]$ -= $\alpha \times$ exp($policy[(action, 1)]$) / z
end for
```

Algorithm 4: Simulate(node)

```
$greedyScores \leftarrow \{\}$
for each available $action$ do
  $greedyScores \leftarrow greedyScores$ + [greedy score of $action$ on $state(node)$
  according to the simulation heuristic]
return $action$ corresponding to argmax($greedyScores$)
```

Algorithm 5: DBNMCS(level, node)

```
if $level = 0$ then
  while $state(node)$ is not a complete solution do
    $action \leftarrow$ Simulate($node$)
    $state(child(node)) \leftarrow state(node) \cup action$
    $node \leftarrow child(node)$
  end while
  return score($state(node)$)
else
  $bestScore \leftarrow -\infty$
  while $node$ is not a leaf do
    $action \leftarrow$ Simulate($node$)
    $state(child(node, 0)) \leftarrow state(node)$
    $state(child(node, 1)) \leftarrow state(node) \cup action$
    for children $i$ of $node$ do
      $temp \leftarrow child(node, i)$
      $score \leftarrow$ DBNMCS($level - 1, temp$)
      if $score > bestScore$ then
        $bestScore \leftarrow score$
        $bestChild \leftarrow temp$
      end if
    end for
    $node \leftarrow bestChild$
  end while
  return $bestScore$
end if
```

Table 3. Set Cover Problem results

Instance	BS	**LDS**				**DBNMCS**						**BNRPALR**					
		CGP		SBH		CGP			SBH			CGP			SBH		
		AVG	GAP	AVG	GAP	AVG	STD	GAP	AVG	STD	GAP	AVG	STD	GAP	AVG	STD	GAP
41	429	465.9	8.6	465	8.4	443	0.4	3.3	442	0	3	433	0.9	0.9	437.2	1.9	1.9
42	512	567.1	10.8	560	9.4	543.9	0.8	6.2	534.2	0.1	4.3	514*	0.8	0.4	515.2*	1.1	0.6
43	516	572.4	10.9	560	8.5	550.4	1.2	6.7	542	0	5	522.6	1.2	1.3	523	1.3	1.4
44	494	533.2	7.9	534	8.1	522.3	0.5	5.7	521	0	5.5	498.5	0.3	0.9	499.6	0.4	1.1
45	512	570.1	11.4	568	10.9	534.8	0.6	4.5	532	0	3.9	515	0.2	0.6	516.3	0.8	0.8
46	560	596.1	6.4	599	7	575	0.4	2.7	574	0	2.5	564.4	0.4	0.8	565.1	0.5	0.9
47	430	469.4	9.2	476	10.7	445.2	0.5	3.5	441	0.5	2.6	433.5	0.1	0.8	433.4	0.1	0.8
48	492	525.9	6.9	524	6.5	505.8	0.5	2.8	502	0	2	499.2	0.5	1.5	499.2	0.6	1.5
49	641	715.9	11.7	700	9.2	674.7	0.5	5.3	672	0	4.8	647.1	0.7	0.9	649.1	1.4	1.3
410	514	544.3	5.9	543	5.6	529.6	0.4	3	529	0	2.9	517	0.2	0.6	517.2	0.3	0.6
51	253	283.8	12.2	282	11.5	269.6	0.9	6.6	265	0.1	4.8	259	0.7	2.4	259.7	1	2.7
52	302	337.5	11.8	335	10.9	322.7	0.7	6.8	315	0	4.3	308.4	0.9	2.1	310.2	1.3	2.7
53	226	243.9	7.9	241	6.6	236	0.4	4.4	233	0	3.1	228.8	0.3	1.2	229.4	0.7	1.5
54	242	257.2	6.3	259	7	253	0.4	4.6	254.2	2.1	5	244.5	0.8	1	247.5	1.6	2.3
55	211	234.2	11	229.3	8.7	222.4	0.9	5.4	220	0	4.3	213.4	0.6	1.1	215.2	1.4	2
56	213	242.1	13.7	243	14.1	228.4	0.9	7.2	221.9	0.1	4.2	214.5	0.4	0.7	215	0.6	0.9
57	293	314.1	7.2	319	8.9	303.8	0.7	3.7	304	0	3.8	296.1	0.5	1.1	297.1	0.6	1.4
58	288	315.3	9.5	318	10.4	303.4	0.5	5.4	300	0	4.2	290.7	0.3	0.9	291.8	0.8	1.3
59	279	297.4	6.6	295	5.7	293	0.6	5	287	0	2.9	280.4*	0.6	0.5	281.8*	0.8	1
510	265	284.7	7.4	284	7.2	278.7	0.3	5.2	277	0	4.5	268.7	0.5	1.4	268.6	0.5	1.4
61	138	148.9	7.9	148	7.2	142.4	0.4	3.2	149.6	0.2	8.4	139*	0.8	0.7	139.3*	1	1
62	146	163.3	11.8	163	11.6	150.5	0.6	3.1	154	0	5.5	147.1*	0.6	0.7	147*	0.6	0.7
63	145	157.9	8.9	158	9	152	0	4.8	153	0	5.5	149.8	0.9	3.3	149.9	1	3.4
64	131	140.5	7.3	140	6.9	134.2	0.7	2.4	136	0	3.8	132	0	0.8	132.2	0.3	0.9
65	161	183.3	13.8	182	13	171.8	0.7	6.7	175	0	8.7	161.6*	0.3	0.4	161.2*	0.2	0.1
A1	253	281.6	11.3	280	10.7	267.2	0.6	5.6	261.5	0.4	3.4	258	0.5	2	260.3	0.8	2.9
A2	252	282.9	12.2	279	10.7	265.4	0.5	5.3	264	0	4.8	257.1	0.9	2	260.1	1.1	3.2
A3	232	260	12.1	256	10.3	249.2	1.9	7.4	244	0	5.2	238.3*	1.4	2.7	240.6	1.3	3.7
A4	234	269.9	15.4	266	13.7	256.1	1	9.5	253	0	8.1	239.3	0.7	2.3	242.9	1.2	3.8
A5	236	256.7	8.8	254.4	7.8	248.5	0.4	5.3	244	0	3.4	240.1	0.5	1.7	242.1	0.8	2.6
B1	69	73.9	7.1	75	8.7	71.9	0.4	4.2	73	0	5.8	70.2	0.6	1.8	70.3	0.5	1.9
B2	76	83	9.2	82	7.9	77.8	0.6	2.4	78	0	2.6	76.2*	0.3	0.3	76.8*	0.8	1.1
B3	80	85.4	6.8	86	7.5	82.4	0.3	3.1	86	0	7.5	81.7	0.4	2.2	81.9	0.4	2.4
B4	79	86.5	9.5	87	10.1	82	0.5	3.8	85	0	7.6	79.5*	0.3	0.6	79.6*	0.4	0.7
B5	72	78	8.3	78	8.3	73.8	0.2	2.4	74	0	2.8	72.8*	0.8	1.1	73*	0.7	1.4
C1	227	256.4	13	252	11	244.6	0.5	7.8	243	0	7	239.5	0.9	5.5	241.8	0.9	6.5
C2	219	251.5	14.8	246	12.3	238.5	0.7	8.9	234.1	0.7	6.9	231.4	1.2	5.6	234.2	2	7
C3	243	267.8	10.2	266	9.5	263.2	1.3	8.3	265.6	2	9.3	257.1	1.5	5.8	261.4	2.1	7.6
C4	219	256.3	17	252	15.1	241.7	1.4	10.4	241	0	10	229.3	1.5	4.7	233.1	1.9	6.4
C5	215	234.9	9.2	235	9.3	231	0.8	7.4	233	0	8.4	225.6	1.4	4.9	230.8	1.5	7.3
D1	60	68.1	13.4	68	13.3	63.3	0.5	5.5	64	0	6.7	61.7	0.5	2.8	62.4	0.6	3.9
D2	66	70.5	6.8	70	6.1	67.6	0.3	2.4	68	0	3	66.3*	0.3	0.5	66.7*	0.4	1
D3	72	79.1	9.9	81	12.5	74.9	0.3	4	75	0	4.2	73.3*	0.5	1.8	73.5*	0.7	2.1
D4	62	66.1	6.5	65.5	5.7	62.8*	0.5	1.3	63.4	0.3	2.2	62.7*	0.5	1.1	63.4*	0.7	2.3
D5	61	66.9	9.7	69	13.1	63.5	0.4	4.1	64	0	4.9	61.5*	0.4	0.8	62.1*	0.7	1.8
E12345	5	5*	0	5*	0	5*	0	0	5*	0	0	5*	0	0	5*	0	0
NRE1	29	30.3	4.4	30	3.4	29.1*	0.3	0.3	30	0	3.4	29.1*	0.3	0.3	29.2*	0.3	0.6
NRE2	30	32.8	9.3	33	10	31.1*	0.5	3.6	32	0	6.7	30.5*	0.5	1.6	31.3*	0.5	4.3
NRE3	27	28.3	4.7	28	3.7	27.9*	0.3	3.3	28	0	3.7	27.6*	0.5	2.1	27.2*	0.4	0.7
NRE4	28	30.8	10	31	10.7	28.6*	0.4	2.3	29	0	3.6	28.4*	0.5	1.3	28.6*	0.5	2.1
NRE5	28	31	10.7	31	10.7	28*	0	0	28*	0	0	28*	0	0	28*	0.2	0.1
NRF1	14	14*	0	14*	0	14*	0	0	14*	0	0	14*	0	0	14*	0	0
NRF2	15	15*	0	15*	0	15*	0	0	15*	0	0	15*	0	0	15*	0	0
NRF3	14	15	7.1	15	7.1	15	0	7.1	15	0	7.1	14.9*	0.4	6.3	14.8*	0.6	5.4
NRF4	14	14*	0	14*	0	14*	0	0	14*	0	0	14*	0	0	14*	0	0
NRF5	13	13.5*	3.7	14	7.7	14	0	7.7	14	0	7.7	13.5*	0.7	4	13.8*	0.5	6.2
NRG1	176	202.1	14.8	197	11.9	194	0.6	10.2	193	0	9.7	217.4	1.9	23.5	232.9	2.3	32.3
NRG2	154	176.3	14.4	171	11	169.5	0.6	10.1	169	0	9.7	188.2	2.6	22.2	203.4	3.2	32.1
NRG3	166	189.1	13.9	187	12.7	182.7	0.6	10	179.2	0.2	8	203.7	3.1	22.7	220.7	2.8	33
NRG4	168	188.6	12.3	192	14.3	184	0.5	9.5	184.4	0.2	9.8	204.9	2.3	22	221.8	2.7	32
NRG5	168	189.9	13	194	15.5	184.3	0.5	9.7	182	0	8.3	205.2	2.7	22.2	218.8	2.4	30.3
NRH1	63	72.7	15.4	71	12.7	68.1	0.5	8.1	68	0	7.9	69.6	1	10.5	70.9	1	12.6
NRH2	63	71.9	14.1	71	12.7	68.2	0.5	8.3	69	0	9.5	69.9	0.6	10.9	70.9	1	12.6
NRH3	59	66.3	12.4	67	13.6	63.4	0.4	7.5	64	0	8.5	65.6	0.8	11.1	66.8	1	13.2
NRH4	58	64.6	11.3	65	12.1	61.8	0.5	6.6	63	0	8.6	63.6	0.8	9.7	65.2	1.1	12.4
NRH5	55	61.6	12	61	10.9	57.9	0.4	5.2	56	0	1.8	59.2	1	7.6	61.5	1.4	11.8