Solving Graph Optimization Problems in a Framework for Monte-Carlo Search

Stefan Edelkamp, Eike Externest, Sebastian Kühl, Sabine Kuske
Universität Bremen

Abstract
In this paper we solve fundamental graph optimization problems like Maximum Clique and Minimum Coloring with recent advances of Monte-Carlo Search. The optimization problems are implemented as single-agent games in a generic state-space search framework, roughly comparable to what is encoded in PDDL for an action planner.

Introduction
In this paper we propose Nested Monte-Carlo Search for solving hard graph problems and chose a search framework that—in analogy to domain-independent planning—links a domain-specific combinatorial problem to a domain-independent search algorithm.

As Clique, Independent Set, Vertex Cover, and Hitting Set are widely known (Karp 1972), for the sake of brevity from Karp’s set we take Graph Coloring as an example. In the encoding as a single-player game the player starts at an arbitrary graph node and chooses in each step a next node until all nodes are selected. The components of the game induce a tree in the natural way with the color assignment as nodes, and the final coloring as leaves. The input graph is stored in an adjacency matrix. A game is a path in the tree from the root to some leaf. A move (play) corresponds to a selection of a graph node. The game is ended by a Boolean condition (terminal). The length and score are recorded and the score is either minimized or maximized. Finding the potential set of successors (legalMoves), finalizes the encoding.

Monte Carlo Search Framework
The randomized optimization scheme we consider belongs to the wider class of Monte-Carlo search algorithms (Browne et al. 2004). The main concept is the random playout (or rollout) of a position, whose outcome, in turn, changes the likelihood of generating successors in subsequent trials.

Beam-NRPA (Cazenave and Teytaud 2012) is an extension of NRPA that maintains B instead of one best solution in each level of its recursion. The motivation is to warrant search progress by an increased diversity of existing solutions to prevent the algorithm from getting stuck in local optima. High-Diversity NPRA (HD-NRPA) (Edelkamp and Cazenave 2016) elaborates on this observation to increase the diversity of the beam, so that according to some specification of distance solutions too close to existing ones are removed from the beam. HD-NRPA provides several further algorithmic advances that prevent us from revisiting its entire implementation. E.g., instead of the moves executed in a rollout the policy table address of the chosen move and the code of its successors is stored. Additionally, the length of the rollout and its score is stored for each bucket in the beam.

Fig. 1 shows the framework implementation for Graph Coloring. The code has been slightly extended to optimize the permutation order based on a greedy coloring algorithm. It is well known that the chromatic number can be determined exactly if the best possible order of nodes for this algorithm has been found. We may also compute the maximum clique for initializing the coloring process. First, because the size of any clique is —of course—a natural lower bound on $\chi$. Then, because it turns out that a maximum clique to be a good point for starting the coloring process. The resulting clique is stored into a file, which is included as a solution prefix in the Graph Coloring solver. We also adapted a selective policy (Cazenave 2016) based on maintaining the remaining degree of uncolored nodes, with a preference given to the ones, whose number of colored neighbors are maximal.

Experiments
For the evaluation we used a single core of a desktop PC (Intel Core i7-4500U, 1.8 GHz, 16 GB), and chose a known
DIMACS benchmark. As competitors we choose UCT (averaging leaf scores), NRPA (HD-NRPA, recursion level 5), NMCS (for nested Monte-Carlo search, recursion level 5), and SAT (calling Lingeling), while applying a binary search on the solution cost value $k$.

Results are shown in Table 1. The SAT solving process had a 1h timeout while UCT was stopped after a hundred thousand rollouts. Solution qualities highlighted in bold are optimal. A dash indicates that no solution has been found. Solution with X-Y denote upper and lower bounds found with the approach. Fractional solution in UCT are averaged over 10 runs. The few results of UCT (the subscript refers to finding the chromatic number, SAT$_{\chi}$ solves Clique for lower bound) show that NRPA$_{\chi}$, NMCS$_{\chi}$ and SAT$_{\chi}$ are superior.

Prior to its own search, the Monte-Carlo solver for Graph Coloring calls the Clique solver NMCS$_{\omega}$ to compute the lower bound and to initialize the rollout with the enforced coloring. Moreover, for the Clique part in Tables 1 NMCS$_{\omega}$ (with at most 100s CPU time) turn out to be much better than the SAT solver (SAT$_{\omega}$, with 1h CPU time).

The CPU time bound for NRPA$_{\chi}$/NMCS$_{\omega}$ was set to 100s for clique finding and for 30m for coloring. The results were close to optimal in many cases. For the harder results with several exceptions, we may conclude that NMCS$_{\chi}$ performs slightly better than NRPA$_{\chi}$ and SAT$_{\chi}$.

**Conclusion**

We have seen a flexible framework for conducting randomized search. A framework implementation of an optimization problem at hand is often as simple as deriving a PDDL or SAT encoding, and, with only a few lines of self-containting code, the resulting performance can be significantly better. (Porco, Machado, and Bonet 2011) generated translations of several NP-hard graph problems to planning, and demonstrated limitation of current technology. Whether or not we will see Monte-Carlo search action planners competing in the near future, thus, will largely depend on the benchmark domains being used.

**References**


Porco, A.; Machado, A.; and Bonet, B. 2011. Automatic polynomial reductions of NP problems into a fragment of STRIPS. In *ICAPS*.