Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS 2017)

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 seveal 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.

```
class Game {
  int length, colors, rollout[V], used[V];
  Game() { colors = length = 0; }
  bool terminal() { return length == V; }
  double score () { return colors; }
  void play(int m) { if (m==colors) colors++; rollout[length++] = m; }
  int legalMoves (int moves[]) {
    int successors = 0;
    for (int j=0;j<colors;j++) used[j] = 0;
    for (int j=0;j<length;j++)
      if (adjacent[length][j]) used[rollout[j]] = 1;
    for (int j = 0; j < colors; j++)
      if (!used[j]) moves[successors++] = j;
    moves[successors++] = colors;
    return successors; }};
```

Figure 1: Framework Code for Graph Coloring.

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 χ. 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

Instance	χ	SAT_χ	UCT_χ	$NRPA_\chi$	$NMCS_\chi$	SAT_ω
anna	11	**11**	**11**	**11**	**11**	11–31
david	11	**11**	**11**	**11**	**11**	11–29
games120	9	**9**	**9**	**9**	**9**	9–14
homer	13			**13**	**13**	
huck	11	**11**	**11**	**11**	**11**	10–25
jean	10	**10**	**10**	**10**	**10**	21–23
fpsol2.i.1	65	8–66		**65**	**65**	–
fpsol2.i.2	30	13–**30**		**30**	**30**	–
fpsol2.i.3	30	11–**30**		**30**	**30**	–
inithx.i.1	54	9–54		**54**	**54**	–
inithx.i.2	31	10–**31**		31–32	**31**	–
inithx.i.3	31	9–31		**31**	**31**	–
le450_5a	5	**5**		5–10	5–8	5–43
le450_5b	5	**5**		5–10	5–8	5–43
le450_5c	5	**5**		5–8	5–8	..
le450_5d	5	**5**		5–8	5–7	–
miles250	8	**8**	**8**	**8**	**8**	8–17
miles500	20		**20**	**20**	**20**	20–39
miles750	31	12–**31**	31.6	**31**	**31**	31–65
miles1000	42	11–**42**	42.3	**42**	**42**	40–80
miles1500	73	10–**73**	**73**	**73**	**73**	60-102
mulsol.i.1	49	18–**49**		**49**	**49**	27–89
mulsol.i.2	31	26–**31**		**31**	**31**	22–88
mulsol.i.3	31	25–**31**		**31**	**31**	24–89
mulsol.i.4	31	27–**31**		**31**	**31**	17–89
myciel3	4	**4**	**4**	2–**4**	2–**4**	**2**
myciel4	5	**5**	**5**	2–**5**	2–**5**	**2**
myciel5	6	5–**6**	**6**	2–**6**	2–**6**	**2**
myciel6	7	5–**7**	**7**	2–**7**	2–**7**	**2**
myciel7	8	5–**8**		2–**8**	2–**8**	2
queens_5_5	5	**5**	**5**	**5**	**5**	**5**
queens_6_6	7	**7**	9	6–**7**	6–**7**	**6**
queens_7_7	7	**7**	9.4	**7**	**7**	7–25
queens_8_8	9	8–**9**	10.7	8–10	8–**9**	8–28
queens_8_12	12	**12**	13.6	**12**	**12**	12–33
queens_9_9	10	8–**10**	12	9–11	9–11	9–33
school1	?	9–**14**		14–15	14–16	–
school1_nsh	?	7–**14**		14–17	14–17	–
zeroin.i.1	49	15–**49**		**49**	**49**	26–92
zeroin.i.2	30	22–**30**		**30**	**30**	28–85
zeroin.i.3	30	21–**30**		**30**	**30**	24–85
DSJC125.1	?	**5**	7	4–6	4–6	**4**
DSJC125.5	?	10–20	21.9	10–21	10–19	10–76
DSJC125.9	?	12–48	50	34–49	34–47	31–118
DSJC250.1	?	6–9		4–10	4–10	4–39
DSJC250.5	?	8–36		12–36	12–35	–
DSJC250.9	?	8–88		43–87	43–84	–
DSJC500.1	?	6–15		5–17	5–16	–
DSJC500.5	?	8–64		12–65	12–63	–
DSJC500.9	?	8–172		52–161	52–156	–
DSJC1000.1	?	6–26		5–27	5–27	–
DSJC1000.5	?	–		14–116	14–114	–
DSJC1000.9	?	–		56–299	56–293	–
latin_square	?	90–121		90–138	90–132	–
le450_15a	15	9–**15**		15–17	15–16	–
le450_15b	15	9–**15**		15–17	15–16	–
le450_15c	15	9–23		15–25	15–24	–
le450_15d	15	9–23		15–25	15–24	–
le450_25a	25	9–**25**		**25**	**25**	–
le450_25b	25	9–**25**		**25**	**25**	–
le450_25c	25	9–27		25–30	25–29	–
le450_25d	25	9–27		25–30	25–29	–
flat300_20_0	20			11–40	11–39	–
flat300_26_0	26			11–40	11–39	–
flat300_28_0	28	9–40		11–31	11-31	–
flat1000_50_0	50	–		15–113	15–112	–
flat1000_60_0	60	–		15–114	15–113	–
flat1000_76_0	76	–		13–114	13–113	–
queens_10_10	?	10–12	13.5	10–13	10–12	10–36
queens_11_11	11	10–13	14.4	11–14	11–13	11–41
queens_12_12	?	12–14	15.9	12–15	12–15	12–44
queens_13_13	13	9–16		13–17	13–16	13–49
queens_14_14	?	10–17		14–18	14–17	–
r_1000.1c	?	–		80–111	80–120	–
r_1000.1	20	9–21		20–21	20–21	–
r_1000.5	234	–		234–246	234–271	–
r_250.1c	64	9–67		64–67	64–66	–
r_250.1	8	**8**		**8**	**8**	7–14
r_250.5	?	8–66		65–67	65–66	–
r_125.1c	46	12–**46**		**46**	**46**	31–88
r_125.1	5	**5**		**5**	**5**	**5**
r_125.5	36	13–**36**		38	**36**	–

Table 1: Graph Coloring Results.

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_χ (the subscript refers to finding the chromatic number, SAT_ω solves Clique for lower bound) show that $NRPA_\chi$, $NMCS_\chi$ and SAT_χ 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_ω, with 1h CPU time).

The CPU time bound for $NRPA_\chi$/$NMCS_\chi$ 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_χ .

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-containing 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

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2004. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Cazenave, T., and Teytaud, F. 2012. Beam nested rollout policy adaptation. In *ECAI-Workshop on Computer Games*, 1–12.

Cazenave, T. 2016. Nested rollout policy adaptation with selective policies. In *IJCAI-Workshop on Computer Games (CGW)*.

Edelkamp, S., and Cazenave, T. 2016. Improved diversity in nested rollout policy adaptation. In *KI*.

Karp, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 85–103. Springer US.

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