Guiding Combinatorial Optimization with UCT*

Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy

IBM Watson Research Center, Yorktown Heights, NY 10598, USA
{ashish.sabharwal,samulowitz,creddy}@us.ibm.com

Abstract. We propose a new approach for search tree exploration in the context of combinatorial optimization, specifically Mixed Integer Programming (MIP), that is based on UCT, an algorithm for the multi-armed bandit problem designed for balancing exploration and exploitation in an online fashion. UCT has recently been highly successful in game tree search. We discuss the differences that arise when UCT is applied to search trees as opposed to bandits or game trees, and provide initial results demonstrating that the performance of even a highly optimized state-of-the-art MIP solver such as CPLEX can be boosted using UCT's guidance on a range of problem instances.

1 Introduction

The order in which a search tree is explored can have a dramatic impact on the performance of a solver designed to solve challenging combinatorial search and optimization problems. Various strategies for search tree traversal have been proposed and shown to exhibit different trade-offs. For instance, extensions of depth-first traversal work best in the context of propositional satisfiability (SAT), while best-first, fastest descent, and various heuristic combinations of the above work better in other contexts such as state space search and mixed-integer programming or MIP optimization [cf. 8, 12]. In these efforts, the goal is to find a way to balance exploration and exploitation in a manner that is most beneficial to the solver under consideration.

Upper Confidence bounds for Trees (UCT) [7] is an exciting technique for balancing exploration and exploitation in search. It has received much attention during the past few years due to its success in game playing agents, especially for Go [4, 5] and Kriegspiel [2], as well as for general game playing [3]. UCT is based on the Upper Confidence Bounds (UCB1) selection strategy introduced by Auer et al. [1] for the multi-armed bandit problem, which guarantees asymptotically optimal regret. In this work, we address the following question: *Can UCT inspired exploration-exploitation techniques help boost the performance of state-of-the-art combinatorial search and optimization solvers?*

* A preliminary version of this paper appeared at the Workshop on Monte-Carlo Tree Search held in Freiburg, Germany in June 2011. The current implementation relies on a newer version of the CPLEX solver, capitalizing on additional cuts learned during search and resulting in significantly improved performance.

N. Beldiceanu, N. Jussien, and É. Pinson (Eds.): CPAIOR 2012, LNCS 7298, pp. 356–361, 2012.

Specifically, we consider optimization in the context of MIP and explore the impact of UCT as a node-selection heuristic for the CPLEX solver [6]. UCT has recently been applied to the Boolean Satisfiability (SAT) problem, albeit in a very basic search setting [9]. We emphasize that CPLEX is a highly optimized commercial solver for MIP problems, obtaining a consistent improvement upon which on a variety of instances through general, domain-independent heuristic strategies is an extremely challenging task. Nevertheless, we pursue this goal rather than working with a limited set of problem domains or with, e.g., a self-designed branch-and-bound solver.

This agenda raises several interesting challenges due to the inherent differences between combinatorial optimization and game tree search. For instance, while UCT was originally introduced for single-agent tree search, its success and application have mainly been in the context of two-agent adversarial search. Further, UCT's "random playout" based sampling technique for evaluating the utility of a given state has been an appealing strategy in games such as Go where known heuristic functions for state evaluation are still quite weak. This is in stark contrast with tree search in the context of MIP optimization, where not only does the linear programming (LP) relaxation often serve as a very strong heuristic, this heuristic value is in fact a guaranteed upper or lower bound on the true objective value (depending on whether it is a maximization or a minimization problem, respectively). Finally, while the UCB1 strategy underlying UCT is designed to exploit (with some balance) a good "branch" once it discovers one, in the context of MIP search, one does not gain anything by revisiting and repeatedly exploiting a "terminal state" even if it always returns the optimal value. UCT must therefore be carefully adapted when applied to our setting.

We show that a UCT-inspired node selection strategy, appropriately modified to take the above mentioned differences into account, can have a positive impact even on sophisticated MIP solvers such as CPLEX. Given the additional overhead of maintaining our own "shadow" search tree for UCT computations, we find that the most benefit is achieved when UCT is used to provide guidance mostly near the top of the tree (we use it to select the first 128 nodes). Overall, UCT still reduced the runtime by 3.6%, the number of search tree nodes by 11.5%, and the number of simplex iterations by 7.4% (geometric mean over the test set). We also find that the overhead of "log" and "square root" computations in the UCB1 formula underlying UCT can be substantial, and that a simpler ϵ-*greedy* version also introduced by Auer et al. [1] works just as well in this setting. One of our key modifications to UCT is the use of a max-style update rule (the "backup operator") rather than the usual additive update rule when a new node is added to the UCT tree. While previous work in the context of game tree search has found max-style update rules to be too brittle, max-style update has clear benefits in our setting because the heuristic value used, namely the LP relaxation objective value, is a guaranteed upper or lower bound on the true value of the node. For completeness, we also compare against best-first search (based on LP relaxation values) and breadth-first search.

UCT is generally thought of as being tied to stochastic sampling of the space via random playouts. Nonetheless, when a good heuristic function is available, it can in fact work better for UCT. For example, Ramanujan et al. [10] demonstrated this in the game of Chess where, unlike Go, very strong heuristic functions are available. More recently, Ramanujan and Selman [11] evaluated such trade-offs for the game of Mancala. We observe the same trend for CPLEX.

2 MIP Search, Node Selection, and UCT

We begin with a brief discussion of the basic mechanisms underlying search tree exploration by a MIP solver, specifically, CPLEX 12.3. The search starts with an empty root node, marked as *open*. It proceeds in general by selecting an open node N for expansion using a *node selection heuristic* $\mathcal{H}$. At this point, the solver tests the sub-problem associated with N for being infeasible, being worse than current best solution (the incumbent), or resulting in a new incumbent; it processes these cases appropriately and marks N as *closed*. If the test fails, assuming binary branching (e.g., bisection domain splitting), node N is split into two open nodes N_{left} and N_{right} by branching on some variable x and restricting its value to a subset of its domain, using a *branching heuristic*; N is marked as closed and N_{left} and N_{right} are marked as open. The search now continues by selecting another open node using $\mathcal{H}$. While the solver usually maintains only the list of open nodes, there is clearly an underlying *search tree* T that is being explored, with all internal nodes and some leaves marked as closed.

In this work, we explore the use of UCT operating on the underlying search tree T as the node selection heuristic $\mathcal{H}$. There are several natural candidates for $\mathcal{H}$ besides UCT. For example, Best-first search would always greedily expand the node with the highest "quality" value (e.g., objective value of the LP relaxation) while breadth-first or depth-first would always expand an open node at the shallowest or deepest level, respectively. Combinations of these basic approaches, such as best-first mixed with depth-first "diving", often work well for MIP. On its own, best-first guides the search towards a solution and proof of optimality in the minimum possible number of explored nodes, but its greedy nature often results in an overhead due to rapid context switches for the solver. Furthermore, for solvers that support learning new information during search (e.g., additional cuts through conflict analysis), best-first search is not guaranteed to minimize the number of nodes. Breadth-first search, on the other hand, is purely exploratory and ignores node quality information. Here we consider UCT as a promising candidate for balancing such exploration and exploitation.

Briefly, at a high level, the UCT algorithm works as follows on an underlying tree T. It alternates between a node selection phase and a tree update phase. *Node selection phase*: Traverse T from the root to a leaf by following, at each node N, the child N' whose *UCT score* is higher (breaking ties arbitrarily). The UCT score of a node N with parent P is defined by the UCB1 formula: $\text{estimate}(N)+\Gamma\cdot\sqrt{\log \text{visits}(P)/\text{visits}(N)}$, where Γ is a fixed constant balancing exploration and exploitation, $\text{visits}(N)$ indicates the number of times N has been

visited by UCT so far (similarly for visits(P)), and estimate(N) is an estimate of the "quality" of N if N is currently a leaf node of T and is otherwise the value resulting from previous tree update phases. *Tree update phase*: Once node selection reaches a leaf L of T, the estimate for L is computed and propagated upwards in T towards the root so that each node N now on the path from L to the root has a value that equals the *average* value seen in the entire subtree rooted at N, and visits is incremented by 1; this is known as the *backup operator* for UCT. L is now further expanded by branching, if possible. For further details, we refer the reader to Kocsis and Szepesvári [7].

3 Guiding MIP Optimization with UCT

In order to perform UCT-based node selection within CPLEX, additional infrastructure must be put in place. We maintain a "shadow" search tree T' whose open leaves coincide with the open nodes list maintained internally by CPLEX.[1] Each node maintains a counter for the number of UCT visits to it so far, and a measure of quality or "estimate" — which for a newly created node is taken to be its LP objective value normalized by the root LP value.[2]

For simplicity, let's assume we have a maximization problem. In contrast to the common averaging backup operator used in UCT, we propagate the *maximum* of the current estimates of the two children when updating the UCT tree. This is motivated by the fact that we do not perform sampling to estimate a node's quality, but instead use a guaranteed LP bound. Hence, averaging would simply blur the knowledge that one has at a given (internal or leaf) node in T'. Further, for computational efficiency, we replace the UCB1 selection criteria (involving log and square root computations) with the following simpler version for a node N with parent P: $\text{score}(N) = \text{estimate}(N) + \Gamma \cdot \frac{\text{visits}(P)/100}{\text{visits}(N)}$.

As mentioned above, estimate(N) is initialized as the normalized LP objective value when N is a leaf node; it is then updated using the maximum backup rule. The parameter Γ was tuned with some small experimentation in our MIP setting to the value 0.7. The UCT score aims at balancing exploration and exploitation. While nodes with very promising objective values are pursued because of the high "estimate" term in the expression, sub-optimal nodes begin to get priority if they have been visited much less compared to their siblings.

Nodes that fail or are pruned by CPLEX are removed from T' (without any objective estimate "penalty" propagated upwards) and never visited again by UCT. For nodes that do yield a feasible solution, we do not treat their resulting objective value in any special fashion when back propagating and we remove

[1] Maintaining a search tree that properly mimics CPLEX's open nodes is somewhat more complex than one might expect because of issues related to capturing every event that may cause CPLEX to close nodes in-between node- and branch-callbacks.

[2] We also experimented with more refined measures combining the LP objective value with the number of integer infeasibilities as a "confidence" guide or with pseudo-costs, but did not observe a clear improvement in performance.

Table 1. Performance of node selection strategies (170 instances, geometric mean)

	UCT	*default CPLEX*	*best-first*	*breadth-first*
runtime (sec)	**54.43**	56.44	56.63	64.08
search nodes	**6,930.62**	7,828.78	7,338.17	7,979.91
simplex iterations	**267,185.24**	288,644.04	282,247.80	323,370.20

them from further consideration in T' because, unlike the usual multi-armed bandit setting, the optimization process doesn't gain anything by revisiting them.

4 Experimental Evaluation

We compare the performance of our UCT based node selection strategy with CPLEX's default heuristic as well as alternative approaches. The experiments were conducted on Intel Xeon CPU E5410 machines, 2.33GHz with 8 cores and 32GB of memory, running Ubuntu. We use CPLEX 12.3 [6] with node and branch "callbacks" turned on (using empty callbacks) as our baseline ("default CPLEX").[3] Starting with a wide selection of publicly available benchmarks comprising 1028 instances, we kept the 170 (see Appendix), spanning a variety of problem domains, on which default CPLEX took between 10 and 900 seconds.

We use each custom node selection strategy for the first 128 nodes, and then revert back to CPLEX's default node selection heuristic. This is motivated by our belief that the most important decisions are made near the top of the search tree and by the fact that our current implementation has an overhead of maintaining a "shadow tree". Following CPLEX's node choice after the first thousand or so nodes turned out to be simply efficient.

The results, with a 600 second timeout, are summarized in Table 1. We compare default CPLEX with our UCT based node selection strategy, and with best-first as well as breadth-first node selection (for the first 128 nodes). The numbers reported are geometric means across the 170 instances.

We see that UCT based node selection improves upon CPLEX's default heuristic in all measures considered. It reduces the geometric mean of the runtime by 3.6% despite the overhead, the number of nodes in the search tree by 11.5%, and the number of iterations performed by dual simplex by 7.4%. Breadth-first search (i.e., pure exploration), on the other hand, is significantly worse. Best-first search (i.e., pure exploitation) shows merit but is not very different from default CPLEX in performance. Note that due to the cuts added during search, best-first search is not necessarily the best at minimizing the number of nodes.

In conclusion, these results suggest that the UCT method for balancing exploration and exploitation, used typically in adversarial game trees and stochastic settings, holds promise also in combinatorial optimization, specifically as a node selection strategy for MIP solvers.

[3] Callbacks cause some features of CPLEX to be turned off (e.g., dynamic search) but are the only way to enhance CPLEX with a custom node selection strategy without access to the internals of CPLEX.

References

[1] Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47(2-3), 235–256 (2002)
[2] Ciancarini, P., Favini, G.P.: Monte Carlo tree search techniques in the game of Kriegspiel. In: 21st IJCAI, Pasadena, CA, pp. 474–479 (July 2009)
[3] Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: 23rd AAAI, Chicago, IL, pp. 259–264 (July 2008)
[4] Gelly, S., Silver, D.: Combining online and offline knowledge in UCT. In: 24th ICML, Corvallis, OR, pp. 273–280 (June 2007)
[5] Gelly, S., Silver, D.: Achieving master level play in 9×9 computer Go. In: 23rd AAAI, Chicago, IL, pp. 1537–1540 (July 2008)
[6] IBM ILOG. IBM CPLEX Optimization Studio 12.3 (2011)
[7] Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
[8] Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization. Wiley-Interscience (1999)
[9] Previti, A., Ramanujan, R., Schaerf, M., Selman, B.: Applying UCT to Boolean Satisfiability. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 373–374. Springer, Heidelberg (2011)
[10] Ramanujan, R., Sabharwal, A., Selman, B.: Understanding sampling style adversarial search methods. In: 26th UAI, Catalina Island, CA (July 2010)
[11] Ramanujan, R., Selman, B.: Trade-offs in sampling-based adversarial planning. In: 21st ICAPS, Freiburg, Germany (June 2011)
[12] Wolsey, L.A.: Integer Programming. Wiley-Interscience (1998)

Appendix: Benchmark Set Used in Experiments

10teams ab51.40.100 ab71.20.100 acc-tight3 acc-tight4 acc-tight5 acc-tight6 air04 air05 aligninq arki001 atlanta-UUM bc1 berlin bienst1 binkar10_1 bley_xl1 bley_xs2 brasil dano3_3 dano3_4 dano3_5 dfn-gwin-DBE dfn-gwin-DBM dfn-gwin-UUE di-yuan-DBE eil33.2 eilB101 exp.1.1000.20.2 exp.1.500.20.1 exp.1.500.20.5 exp.1.500.50.2 exp.1.500.50.4 exp.1.500.50.5 exp.1.5000.5.2 exp.1.5000.5.3 fc.60.20.2 fc.60.20.6 france-DBM france-UUM g200x740 g200x740b g55x188 harp2 ic97_tension k20x380 l451x885b markshare_4_0 mas76 mik.250-20-75.1 mik.250-20-75.2 mik.250-20-75.3 mik.250-20-75.4 mik.250-20-75.5 misc07 mkc1 mod011 mzzv11 mzzv42z n12-3 n5-3 n7-3 neos-1109824 neos-1112782 neos-1112787 neos-1171737 neos-1200887 neos-1211578 neos-1215259 neos-1228986 neos-1337489 neos-1440225 neos-1440447 neos-1445738 neos-1445743 neos-1445755 neos-1445765 neos-1480121 neos-1582420 neos-1597104 neos-1620807 neos-430149 neos-476283 neos-480878 neos-503737 neos-504674 neos-504815 neos-512201 neos-522351 neos-530627 neos-538867 neos-547911 neos-555424 neos-570431 neos-584851 neos-585192 neos-593853 neos-595925 neos-686190 neos-785899 neos-801834 neos-803219 neos-803220 neos-806323 neos-807639 neos-807705 neos-808072 neos-810326 neos-820879 neos-825075 neos-827015 neos-829552 neos-839859 neos-860300 neos-862348 neos-906865 neos-912023 neos-916173 neos-935627 neos-935769 neos-936660 neos-937446 neos-937511 neos-941313 neos-941698 neos-960392 neos1 neos11 neos12 neos14 neos17 neos18 neos20 neos21 neos22 neos23 neos6 neos7 nexp.50.20.4.1 nexp.50.20.4.3 nexp.50.20.8.2 nexp.50.20.8.3 ns4-pr4 ns60-pr9 nu25-pr4 nu60-pr4 p50x288b p80x400 pdh-DBE pdh-DBM pdh-UUE pdh-UUM pk1 prod1 r20x200 r50x360 ran10x26 ran12x21 ran13x13 ran16x16 rout seymour1 sp98ir stein45 swath1 swath2 ta1-DBE ta1-DBM ta2-UUE ta2-UUM