Toward a Search Strategy for Anytime Search in Linear Space Using Depth-First Branch and Bound

Carlos Hernández
Departamento de Ingeniería Informática
Universidad Católica de la Santísima Concepción
Concepción, Chile

Jorge A. Baier
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Depth-First Branch and Bound (DFBnB) is an anytime algorithm for solving combinatorial optimization problems. In this paper we present a weighted version of DFBnB, wDFBnB, which incorporates standard techniques for using weights in heuristic search and offers suboptimality guarantees. Our main contribution drawn from a preliminary evaluation is the observation that wDFBnB, used along with automated or hand-crafted weight schedules, can significantly outperform DFBnB both in terms of anytime behavior and convergence to the optimal. We think this small study calls for more research on the design of automated weight schedules that could provide superior anytime performance across a wider range of domains.

Introduction

Depth-First Branch and Bound (DFBnB) (Balas and Toth 1985) is an anytime search algorithm used in combinatorial optimization. It is the algorithm of choice when the problem at hand requires setting a fixed number of variables. One of its advantages is that its memory requirements are linear on the problem's description. It explores the search space in a depth-first manner, pruning a node s when $g(s)+h(s) \geq U$, where $g(s)$ is the cost incurred to get to s, and $h(s)$ is a lower bound on the cost to reach a solution from s. Initially, U is set to ∞ and whenever a solution node is found, U is set to the cost of such as solution. DFBnB can be seen as an anytime algorithm; indeed, it can be trivially modified to print a solution as soon as it is found.

Using techniques proposed in the literature, there are various straightforward ways in which DFBnB can be modified to produce w-optimal solutions, i.e., solutions whose cost may exceed optimal by at most a factor of w. Using a technique proposed by Pohl (1970) for the Weighted A* algorithm, one can modify DFBnB to prune node s whenever $g(s) + wh(s) \geq U$. Alternatively, one can prune using stricter rules, such as $wg(s)+wh(s) \geq U$ (cf., Hatem, Stern, and Ruml, 2013). A further generalization, which is the one we propose in this paper, is wDFBnB(w_g, w_h), an algorithm that prunes a node whenever $w_g g(s) + w_h h(s) \geq U$ can be proven W-optimal, where $W = \max\{w_g, w_h\}$. Clearly, wDFBnB(w_g, w_h) is an anytime algorithm.

When looking for a w-optimal solution, it is not obvious what algorithm to use. Intuitively, wDFBnB(w, w) has the strictest pruning rule and thus may, at first sight, seem the best because it should be expected to prune more nodes than e.g. wDFBnB$(1, w)$. However, there are other alternatives, such as running wDFBnB$(1, w')$ for an arbitrary $w' > w$, restart with a lower weight, and repeat the process until we can *prove* that the solution is w-optimal. These restarting policies have been studied before in the anytime heuristic search literature (Likhachev, Gordon, and Thrun 2003; Hansen and Zhou 2007; Richter, Thayer, and Ruml 2010).

What is the best algorithm for finding w-optimal solutions? Which is the best way to run these algorithms in order to get the best anytime behavior? This paper aims at providing some clues towards answering those two questions. From our preliminary experimental analysis, which we carry out on travelling salesman problem (TSP), we conclude that the answer is that when a w-optimal solution is sought it might be best to run wDFBnB$(1, w')$ for a decreasing *weight schedule*, i.e., a sequence of decreasing weights. Interestingly, our findings also hold when we are looking for an optimal solution, that is wDFBnB, used with a good weight policy, does not only seem to find an optimal solution faster than DFBnB, but also has a better anytime behavior.

Below we present wDFBnB's pseudocode, analyze its properties, and describe our experimental analysis on the TSP. We finish with brief conclusions.

Weighted Depth-First Branch-and-Bound

Algorithm 1 provides a pseudocode for wDFBnB. Unlike standard DFBnB, it returns the sum of $min_{s \in P} g(s) + h(s)$, where P is the set of pruned nodes. This modification is inspired by the way IDA* (Korf 1985) sets the cost bound for the next iteration, and we use it to design adaptive weight schedules, that we explain below. We assume $Succ(s)$ is the set of successors of a node s, and that $c(s, s')$ is the cost of the action that transforms s into s'.

There are a number of interesting properties about this algorithm. For simplicity, below we use wDFBnB(w_g, w_h) instead of wDFBnB$(w_g, w_h, init, 0)$, where $init$ denotes the initial state, and we assume that h is an admissible heuristic. The first two results provide suboptimality guarantees.

Theorem 1 *All solutions printed by wDFBnB(w_g, w_h) are $\max\{w_g, w_h\}$-optimal.*

Algorithm 1: Weighted Depth-First Branch-and-Bound

```
1  procedure wDFBnB-Driver(init, w_g, w_h, w_target)
2      U ← a provably correct upper bound (i.e., ∞)
3      repeat
4          L ← wDFBnB(w_g, w_h, init, 0)
5          print "current suboptimality is U/L"
6          ⟨w_g, w_h⟩ ← getNextWeights()
7      until U/L ≤ w_target
8  function wDFBnB(w_g, w_h, s, cost)
9      min_f ← ∞
10     if s is a goal node then
11         if cost < U then
12             U ← cost
13             print current solution s and cost
14         min_f ← cost
15     else
16         foreach s' ∈ Succ(s) do
17             g ← cost + c(s, s')
18             if w_g g + w_h h(s') ≥ U then
19                 f ← g + h(s')
20             else
21                 f = wDFBnB(w_g, w_h, s', g)
22             if min_f > f then min_f ← f
23     return min_f
```

Theorem 2 *Let L and sol be respectively the return value of and the last solution printed by wDFBnB(w_g, w_h). Then sol is (U/L)-optimal.*

The following result establishes that if a strictly decreasing weight policy is used, then we eventually find a w_{target}-optimal solution.

Theorem 3 *Let function getNextWeights be such that it returns a pair $\langle w'_g, w'_h \rangle$, such that $1 \leq \max\{w'_g, w'_h\} < \max\{w_g, w_h\}$. Then wDFBnB-Driver terminates printing a w_{target}-optimal solution.*

Finally, the following theorem sets the basis for implementing adaptive weight schedules that guarantee termination.

Theorem 4 *Let w_g and w_h be such that $w_g \neq w_h$, and let L and sol be respectively the return value of and the last solution printed by the call to wDFBnB in the wDFBnB-Driver procedure. Then $U/L < \max\{w_g, w_h\}$.*

In particular Theorem 4 establishes that implementing *getNextWeights* such that it returns a pair $\langle w'_g, w'_h \rangle$ that is such that $\max\{w'_g, w'_h\} \leq U/L$, and such that $w'_g \neq w'_h$ guarantees termination (and thus w_{target}-optimality).

Experimental Evaluation

The objective of our experiments was to evaluate the effect of different weight schedules on the anytime performance of wDFBnB(w, w) and wDFBnB$(1, w)$. We used 2 symmetric TSP instances (27 and 51 cities). We tested 4 weight schedules: two fixed weight schedules and two adaptive weight schedules based on Theorem 4. Schedule p1 and p2 decrement w by 0.05 and 0.1, respectively. Schedule p3 sets the next value as $w = U/L$ while p4 sets it to $w = .99U/L$. We used 60 seconds as a deadline, and $w = 1.5$ as an initial weight. w_{target} was set to 1.

Our first observation is that wDFBnB$(1, w)$ is clearly superior to wDFBnB(w, w). In the 51-city problem wDFBnB(w, w) prints only one solution, which it cannot improve with any of the weight schedules. In the 27-city problem wDFBnB(w, w) requires more than one order of magnitude more time to converge to the optimal solution than wDFBnB$(1, w)$.

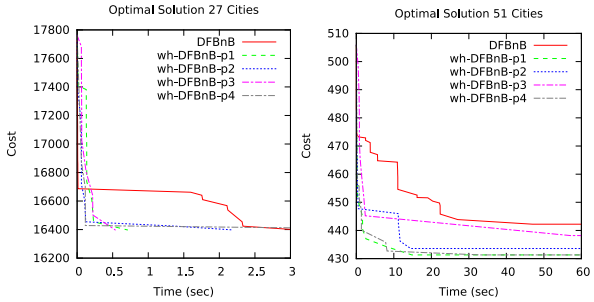

Figure 1: wDFBnB's anytime behavior superior to DFBnB

We observe the anytime behavior of wDFBnB$(1, w)$ is superior to that of DFBnB. Furthermore, in the 27-city problem wDFBnB converges faster to the optimal solution. The fixed schedule p1 and the adaptive schedule p4 seem to obtain best results.

Summary and Conclusions

We presented wDFBnB an anytime weighted version of DFBnB that allows finding suboptimal solutions with quality guarantees. Our preliminary evaluation is promising and supports the fact when looking for an optimal, wDFBnB is superior to DFBnB both in its anytime behavior and in total runtime. Our evaluation shows the potential of both handcrafted and automated weight schedules. Automated weight schedules seem advantageous since they do not need tuning and might be built automatically by exploiting the structure of the search space. However, more experimental evidence across different benchmarks is needed to draw stronger conclusions. Future work will also consider comparing to dovetailing (Valenzano et al. 2010) and other related methods.

References

Balas, E., and Toth, P. 1985. *Branch and Bound Methods*. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. John Wiley & Sons, Inc.

Hansen, E. A., and Zhou, R. 2007. Anytime heuristic search. *JAIR* 28:267–297.

Hatem, M.; Stern, R.; and Ruml, W. 2013. Bounded suboptimal heuristic search in linear space. In *Proc. SoCS*.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *AIJ* 27(1):97–109.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2003. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *Proc. NIPS*.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *AIJ* 1(3):193–204.

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *Proc. ICAPS*, 137–144.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *Proc. ICAPS*, 177–184.