



WIKIPEDIA
The Free Encyclopedia

Main page

Contents

Featured content

Current events

Random article

Donate to Wikipedia

Wikipedia store

Interaction

Help

About Wikipedia

Community portal

Recent changes

Contact page

Tools

What links here

Related changes

Upload file

Special pages

Permanent link

Page information

Wikidata item

Cite this page

Print/export

Create a book

Download as PDF

Printable version

Languages

العربية

Čeština

Deutsch

Español

فارسی

Français

한국어

Italiano

日本語

Português

Русский

Српски / srpski

Srpskohrvatski /

српскохрватски

Українська

Edit links

Create account Log in

Article

Talk

Read

Edit

View history

Search



Branch and bound

From Wikipedia, the free encyclopedia

Branch and bound (**BB** or **B&B**) is an [algorithm](#) design paradigm for [discrete](#) and [combinatorial optimization](#) problems, as well as [general real valued problems](#). A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions by means of [state space search](#): the set of candidate solutions is thought of as forming a [rooted tree](#) with the full set at the root. The algorithm explores *branches* of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated *bounds* on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far by the algorithm.

The algorithm depends on the efficient estimation of the lower and upper bounds of a region/branch of the search space and approaches exhaustive enumeration as the size (*n*-dimensional volume) of the region tends to zero.^[*clarification needed*]^[*citation needed*]

The method was first proposed by A. H. Land and A. G. Doig^[1] in 1960 for [discrete programming](#), and has become the most commonly used tool for solving [NP-hard](#) optimization problems.^[2] The name "branch and bound" first occurred in the work of Little *et al.* on the [traveling salesman problem](#).^[3]^[4]

Contents

[hide]

- Overview
 - Generic version
- Applications
- Relation to other algorithms
- See also
- References

Overview

[edit]

In order to facilitate a concrete description, we assume that the goal is to find the *minimum* value of a function *f*(*x*), where *x* ranges over some set *S* of *admissible* or *candidate solutions* (the *search space* or *feasible region*). Note that one can find the *maximum* value of *f*(*x*) by finding the minimum of *g*(*x*) = −*f*(*x*). (For example, *S* could be the set of all possible trip schedules for a bus fleet, and *f*(*x*) could be the expected revenue for schedule *x*.)

A branch-and-bound procedure requires two tools. The first one is a *splitting* procedure that, given a set *S* of candidates, returns two or more smaller sets *S*₁, *S*₂, . . . whose union covers *S*. Note that the minimum of *f*(*x*) over *S* is **min**{*v*₁, *v*₂, . . .}, where each *v*_{*i*} is the minimum of *f*(*x*) within *S*_{*i*}. This step is called **branching**, since its recursive application defines a *search tree* whose *nodes* are the subsets of *S*.

The second tool is a procedure that computes upper and lower bounds for the minimum value of *f*(*x*) within a given subset of *S*. This step is called **bounding**.

The key idea of the BB algorithm is: if the *lower* bound for some tree node (set of candidates) *A* is greater than the *upper* bound for some other node *B*, then *A* may be safely discarded from the search. This step is called **pruning**, and is usually implemented by maintaining a global variable *m* (shared among all nodes of the tree) that records the minimum upper bound seen among all subregions examined so far. Any node whose lower bound is greater than *m* can be discarded.

The recursion stops when the current candidate set *S* is reduced to a single element, or when the upper bound for set *S* matches the lower bound. Either way, any element of *S* will be a minimum of the function within *S*.

Graph and tree search algorithms

α-β · A* · B* · Backtracking · Beam · Bellman–Ford · Best-first · Bidirectional · Borůvka · Branch & bound · BFS · British Museum · D* · DFS · Depth-limited · Dijkstra · Edmonds · Floyd–Warshall · Fringe search · Hill climbing · IDA* · Iterative deepening · Johnson · Jump point · Kruskal · Lexicographic BFS · Prim · SMA*

Listings

Graph algorithms · Search algorithms · List of graph algorithms

Related topics

Dynamic programming · Graph traversal · Tree traversal · Search games

v · t · e

When \mathbf{x} is a vector of \mathbb{R}^n , branch and bound algorithms can be combined with [interval analysis](#)^[5] and [contractor](#) techniques in order to provide guaranteed enclosures of the global minimum.^{[6][7]}

Generic version [\[edit\]](#)

The following is the skeleton of a generic branch and bound algorithm for minimizing an arbitrary objective function f .^[2] To obtain an actual algorithm from this, one requires a bounding function g , that computes lower bounds of f on nodes of the search tree, as well as a problem-specific branching rule.

- Using a [heuristic](#), find a solution x_h to the optimization problem. Store its value, $B = f(x_h)$. (If no heuristic is available, set B to infinity.) B will denote the best solution found so far, and will be used as an upper bound on candidate solutions.
- Initialize a queue to hold a partial solution with none of the variables of the problem assigned.
- Loop until the queue is empty:
 - Take a node N off the queue.
 - If N represents a single candidate solution x and $f(x) < B$, then x is the best solution so far. Record it and set $B \leftarrow f(x)$.
 - Else, *branch* on N to produce new nodes N_i . For each of these:
 - If $g(N_i) > B$, do nothing; since the lower bound on this node is greater than the upper bound of the problem, it will never lead to the optimal solution, and can be discarded.
 - Else, store N_i on the queue.

Several different queue data structures can be used. A [stack](#) (LIFO queue) will yield a [depth-first](#) algorithm. A [best-first](#) branch and bound algorithm can be obtained by using a [priority queue](#) that sorts nodes on their g -value.^[2] The depth-first variant is recommended when no good heuristic is available for producing an initial solution, because it quickly produces full solutions, and therefore upper bounds.^[8]

Applications [\[edit\]](#)

This approach is used for a number of [NP-hard](#) problems

- [Integer programming](#)
- [Nonlinear programming](#)
- [Travelling salesman problem](#) (TSP)^{[3][9]}
- [Quadratic assignment problem](#) (QAP)
- [Maximum satisfiability problem](#) (MAX-SAT)
- [Nearest neighbor search](#)^[10]
- [Cutting stock problem](#)
- [False noise analysis](#) (FNA)
- [Computational phylogenetics](#)
- [Set inversion](#)
- [Parameter estimation](#)
- [0/1 knapsack problem](#)
- [Feature selection in machine learning](#)^[11]
- [Structured prediction in computer vision](#)^{[12]:267–276}

Branch-and-bound may also be a base of various [heuristics](#). For example, one may wish to stop branching when the gap between the upper and lower bounds becomes smaller than a certain threshold. This is used when the solution is "good enough for practical purposes" and can greatly reduce the computations required. This type of solution is particularly applicable when the cost function used is [noisy](#) or is the result of [statistical estimates](#) and so is not known precisely but rather only known to lie within a range of values with a specific [probability](#).

Relation to other algorithms [\[edit\]](#)

Nau *et al.* present a generalization of branch and bound that also subsumes the A^* , B^* and [alpha-beta](#) search algorithms from [artificial intelligence](#).^[13]

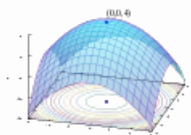
See also [\[edit\]](#)

- [Backtracking](#)
- [Branch-and-cut](#), a hybrid between branch-and-bound and the [cutting plane](#) methods that is used extensively for solving [integer linear programs](#).

References [\[edit\]](#)

1. [^] A. H. Land and A. G. Doig (1960). "An automatic method of solving discrete programming problems". *Econometrica* **28** (3). pp. 497–520. doi:[10.2307/1910129](https://doi.org/10.2307/1910129)[↗].
2. [^] ^a ^b ^c Clausen, Jens (1999). *Branch and Bound Algorithms—Principles and Examples*  (PDF) (Technical report). University of Copenhagen.
3. [^] ^a ^b Little, John D. C.; Murty, Katta G.; Sweeney, Dura W.; Karel, Caroline (1963). "An algorithm for the traveling salesman problem"  (PDF). *Operations Research* **11** (6): 972–989. doi:[10.1287/opre.11.6.972](https://doi.org/10.1287/opre.11.6.972)[↗].
4. [^] Balas, Egon; Toth, Paolo (1983). *Branch and bound methods for the traveling salesman problem*  (PDF) (Report) (Management Science Research Report MSRR-488). Carnegie Mellon University Graduate School of Industrial Administration.
5. [^] Moore, R. E. (1966). *Interval Analysis*. Englewood Cliff, New Jersey: Prentice-Hall. ISBN 0-13-476853-1.
6. [^] Jaulin, L.; Kieffer, M.; Didrit, O.; Walter, E. (2001). *Applied Interval Analysis*. Berlin: Springer. ISBN 1-85233-219-0.
7. [^] Hansen, E.R. (1992). *Global Optimization using Interval Analysis*. New York: Marcel Dekker.
8. [^] Mehlhorn, Kurt; Sanders, Peter (2008). *Algorithms and data structures: The basic toolbox*. Springer. p. 249.
9. [^] Conway, Richard Walter; Maxwell, William L.; Miller, Louis W. (2003). *Theory of Scheduling*. Courier Dover Publications. pp. 56–61.
10. [^] Fukunaga, Keinosuke; Narendra, Patrenahalli M. (1975). "A branch and bound algorithm for computing k -nearest neighbors". *IEEE Transactions on Computers*: 750–753. doi:[10.1109/t-c.1975.224297](https://doi.org/10.1109/t-c.1975.224297)[↗].
11. [^] Narendra, Patrenahalli M.; Fukunaga, K. (1977). "A branch and bound algorithm for feature subset selection". *IEEE Transactions on Computers* **C-26** (9): 917–922. doi:[10.1109/TC.1977.1674939](https://doi.org/10.1109/TC.1977.1674939)[↗].
12. [^] Nowozin, Sebastian; Lampert, Christoph H. (2011). "Structured Learning and Prediction in Computer Vision". *Foundations and Trends in Computer Graphics and Vision* **6** (3–4): 185–365. doi:[10.1561/06000000033](https://doi.org/10.1561/06000000033)[↗]. ISBN 978-1-60198-457-9.
13. [^] Nau, Dana S.; Kumar, Vipin; Kanal, Laveen (1984). "General branch and bound, and its relation to A* and AO*"  (PDF). *Artificial Intelligence* **23** (1): 29–58. doi:[10.1016/0004-3702\(84\)90004-3](https://doi.org/10.1016/0004-3702(84)90004-3)[↗].

v · t · e	Optimization: Algorithms, methods, and heuristics		[hide]
	Unconstrained nonlinear: Methods calling ...		[show]
	Constrained nonlinear		[show]
	Convex optimization		[show]
	Combinatorial		[hide]
Paradigms	Approximation algorithm · Dynamic programming · Greedy algorithm · Integer programming (Branch & bound or cut)		
Graph algorithms	Minimum spanning tree	Bellman–Ford · Borůvka · Dijkstra · Floyd–Warshall · Johnson · Kruskal	
Network flows	Dinic · Edmonds–Karp · Ford–Fulkerson · Push–relabel maximum flow		
	Metaheuristics		[show]
Categories (Algorithms and methods · Heuristics) · Software			



Categories: Optimization algorithms and methods | Combinatorial optimization

This page was last modified on 28 August 2015, at 18:23.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.