Advanced analysis of branch and bound algorithms

Turkensteen, Marcel

Chapter 1

Introduction

1.1 Introduction

Advanced Analysis of Branch and Bound: the title suggests that this thesis deals with a subject that is disconnected from daily life. However, the problems studied in this thesis are frequently encountered in practice. To give the reader a feel for the type of problems discussed in this dissertation, this introduction starts gently with two practical and easily understandable applications, namely Google and route planners. In this thesis, we do not consider these specific applications. Instead, we consider some similar practical problems that are computationally very difficult to solve. Solution times are typically so long, that it is a major challenge to improve methods for obtaining optimal solutions to such problems, such as *Branch and Bound methods*.

When we browse the internet, it is very likely that we use a search engine such as Google. This application searches through billions of web sites and produces a set of results in less than a second. For example, a search for the term 'Groningen' yields about 43 million results in just 0.5 seconds.[1] Moreover, it makes an ordering of the websites in such a way that the most relevant one is presented first. How can Google operate so quickly?

Other popular internet applications are route planners. Suppose that a person uses public transportation to travel from location A to location B, for example from the train station to a university building. The travel schedule can be determined on the internet, for example, on `http://www.9292ov.nl` in the

[1] Search on May 9, 2006, 11:37 AM

Netherlands. Given a starting location A, a destination B and an arrival or a departure time, the computer determines the fastest schedule, and it does so within a second. However, the system contains a huge database of addresses and public transport connections. So how does the system retrieve the fastest trajectory so quickly?

In this thesis we study a category of problems that occur in a wide range of applications, the so-called *optimization problems*. In these problems, we search through a set of candidate solutions to find a solution that satisfies an *a priori* objective. One of the most well-known problems is the *shortest path problem*, where one wants to find a shortest path between two given points. The *candidate solutions* are the paths between the two locations and the a priori objective is 'determine a shortest path'. This problem is an example of a so-called Combinatorial Optimization Problem, or COP, because one wants to find a *combination* of such segments with minimal length. A formal definition of a general COP is given in Section 1.2.

For the so-called *easy* COPs, quick solution methods exist. For example, route planners are able to determine a shortest path between two locations almost instantaneously, even when the number of possible routes is huge. On the other hand, there is still a large class of *hard* problems for which efficient and quick methods that solve all instances to optimality are not available yet. Exhaustive search is needed for such problems, meaning that, in principle, all possible options need to be checked. Not only scientists, also decision makers in industry and in governments face such difficult problems frequently. In modern decision making, large quantities of data are available to support the decision. Effective processing of the information may help, for example, to serve customers with tailored offers, or to implement more effective schedules in factories. Improving solution methods for these problems is of great practical importance, even when seemingly small improvements are made.

The contribution of this dissertation lies in the improvement of solution methods of difficult COPs to optimality. First of all, a definition of COPs is provided. After that, are view of *complexity theory* is given, i.e., the field of research that studies the complexity of COPs. Next, a short overview is given of methods for solving difficult problems to optimality.

This dissertation consists of four separate papers about a wide range of sub-

jects in *Branch and Bound*, a class of methodes explained in Section 1.5. Chapter 2, 3, and 4 discuss potential improvements to the BnB techniques themselves. Chapter 5 considers an application of these techniques in the field of marketing.

1.2 Combinatorial Optimization Problems

In this section, a formal definition of a *Combinatorial Optimization Problem (COP)* is given. According to Wikipedia (2006), "combinatorial optimization is a branch of optimization in applied mathematics and computer science, related to operations research, algorithm theory and computational complexity theory that sits at the intersection of several fields, including artificial intelligence, mathematics and software engineering." In Goldengorin (2002), a COP is defined by the following quadruple $(\mathcal{E}, \mathcal{D}, C, f_C)$. The set of elements of a COP is called the *ground set* and it is denoted by $\mathcal{E}$. The set $\mathcal{D}$ contains the feasible solutions that can be constructed from the elements in $\mathcal{E}$; $\mathcal{D}$ should be finite (Schrijver, 2003). The feasible solutions are evaluated with the *objective function* f_C. The *additive* cost function, used throughout most of this dissertation, adds the costs of all elements in a chosen combination. By 'costs' we do not only mean monetary costs, but also distances, times, weights, or any other measure reflecting our objective function. Finally, C denotes the specific *instance* of the problem. For instance, a shortest path instance is characterized by the distances between each pair of locations.

Many COPs are graph problems, occurring in, for example, transportation or communication networks. Since terms from graph theory are used throughout this dissertation, a basic introduction into graph theory is given here. A *graph* $G(V, E)$ consists of a set of vertices V and a set of edges E between the vertices. If the edges are directed, the graph $G(V, A)$ is called a *digraph* with the set of vertices V and the set of *arcs* A. A *walk* or a *path* is alternating sequence of arcs or edges and vertices between a pair of vertices $v, w \in V$. A *tour* or a *cycle* is a closed walk, meaning that the starting point and the end point are the same. A *tree* on a subset $V' \subseteq V$ of the vertices is a set of edges such that V' is connected but there are no cycles in V'. Figure 1.1 shows examples of a tree and a tour. An extensive introduction in graph theory is given in, among others, Jungnickel (2005) and Bang-Jensen and Gutin (2001).

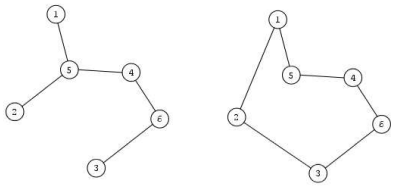

Figure 1.1. Examples of a tree (left) and a tour (right)

Take for example the shortest path problem solved by route planners. The ground set is formed by the arcs between each pair of locations, and the set of solutions contains all possible paths between the origin and the destination. The instance is defined by the costs of all the arcs, and the objective is minimization of the path length.

1.3 Complexity theory

Why can the easy problems, such as those in Google and route planners, be solved quickly, whereas other problems are very difficult to solve? The field of research that studies this question is *complexity theory*.

The most well-known example of a difficult COP is the *Traveling Salesman Problem (TSP)* (Punnen, 2002), which is defined as follows. Given n locations and given the matrix C containing the distances between each pair of locations, find the shortest tour through all locations such that each location is visited exactly once. An extensive list of applications of the TSP is given in Punnen (2002). The TSP is easy to formulate intuitively, and at first sight, it may appear straightforward to solve. However, the appearance is deceptive, since in the worst case, all solutions of the problem must be enumerated. The TSP is the most widely known hard problem, but there are many other ones.

The number of possible combinations in a COP usually increases very rapidly with the number of elements from which the combinations are chosen, the *input size*; see Garey and Johnson (1979). For example, there may be as many as $(n-1)!$ possible TSP solutions through n cities. So when $n = 20$, there are

approximately $19! \approx 1.216 \times 10^{17}$ possible tours. In a complete network with n intermediate locations, there are approximately $(n-2)!$ possible paths from which a shortest one should be chosen. If $n = 20$, there are potentially as many as 6.40×10^{15} paths. In the examples at the beginning of this introduction, the number of websites and the number of possible routes between a pair of locations are even much larger. For most COPs, the number of solutions increases exponentially with the size of the input n, meaning that the number of solutions for input size $n+1$ is typically a multiple of the number of solutions for input size n (Garey and Johnson, 1979).

Is it necessary to investigate all, or a large part of these possible solutions, or does it suffice to consider only a small fraction? In other words, when is a problem easy or difficult to solve? The hardness varies tremendously between COPs: some instances of easy problems of size 100,000 are solved within seconds, whereas it may take a long time to solve some hard problem instances of size 100, if they can be solved at all. For example, the notoriously hard Quadratic Assignment Problem (QAP) is the problem of assigning n facilities to n locations, in such a way that the sum of the weights of the flow between each pair of facilities times the distance between them is minimized. The largest non-trivial QAP instances solved to optimality are of size 32 (Loiola *et al.*, 2007).

The difficulty of a COP is usually measured with the *worst-case time complexity* of the fastest algorithm solving it to optimality (Aho *et al.*, 1974). An *algorithm* is defined in Knuth (1997) as a procedure with four characteristics: finite running times, precisely defined steps, input and output.

The worst-case complexity of a problem is written as a function $f(n)$ of the input size n of the problem instance; see for example (Cook *et al.*, 1998). The input size contains the number of elements in the ground set. The function $f(n)$ contains the number of basic operations, such as adding, subtracting or comparing numbers, that have to be carried out to find an optimal solution of a problem with problem size n. The expression $O(f(n))$, which should be read 'of order $f(n)$', is used to denote the complexity of a COP. Let N denote the number of operations, then $f(n)$ is the smallest function such that $N \leq Cf(n)$ with C being a constant. For example, if the fastest algorithm for a COP needs at most $3n^2 + 15n$ operations, then the COP is $O(n^2)$, since $3n^2 + 15n < 4n^2$ for $n > 15$. *Polynomial algorithms* are of polynomial order, for example $O(n^2)$

or $O(n^3 \log(n))$, whereas *exponential algorithms* are of higher order, for example $O(2^n)$ or $O(n!)$. Table 1.1 shows that the solution times of exponential algorithms increase much more rapidly than the solution times of polynomial algorithms. The reported times are hypothetical; it is assumed that a computer carries out one million elementary operations per second. Since the number of operations of polynomial algorithms increases relatively slowly with the input size n, these algorithms are called *efficient* (Schrijver, 2003). For an *easy* COP, efficient algorithms are available, and the worst-case complexity is polynomial. On the other hand, the worst-case time complexity of a problem for which only exponential algorithms are available, is exponential, which can make the problem difficult to solve. An extensive account on complexity theory is given in Garey and Johnson (1979).

Table 1.1. Speed of polynomial and exponential time algorithms (Garey and Johnson, 1979)

Time complexity function	Size n 10	30	50
n	0.00001 sec.	0.00003 sec.	0.00005 sec.
n^2	0.0001 sec.	0.0009 sec.	0.0025 sec.
n^5	0.1 sec.	24.3 sec.	5.2 min.
2^n	0.001 sec.	17.9 min	35.7 years
3^n	0.59 sec.	6.5 years	2×10^8 centuries

The previous part considered worst-case complexities to measure the difficulty of a COP. However, the fact that an algorithm is exponential does not automatically imply that it is always slow: an $O(2^n)$ algorithm may be able to solve most problems in polynomial time, requiring exponential time only for a small subset of instances. A well-known example of such an algorithm is the *Simplex-method* from Dantzig *et al.* (1955). This popular algorithm needs a polynomial number of operations for many Linear Programming instances (Todd, 2002; Wolfe and Cutler, 1963), but there are some instances for which exponential time is needed (Klee and Minty, 1972). The performance of the algorithms in this dissertation is therefore mainly measured with *average case analysis* instead of worst-case analysis. This means that solution methods are tested on a wide range of typical instances and average solution times are measured.

Time complexity of COPs is a widely studied field of research, the most no-

table work being Cook (1971); Karp (1972, 1975); Garey and Johnson (1979). The main result from Cook (1971) deals with decision problems, i.e., problems with yes or no as answers. For example, a TSP can be expressed in the decision form as follows: is there a complete tour through n locations with cost at most c, given a cost matrix C? Cook (1971) classifies all such problems which have a yes answer verifiable in polynomial time as the class $\mathcal{NP}$. This class $\mathcal{NP}$ consists of two broad classes. The first class is called polynomial ($\mathcal{P}$). Problems in these class are those, which, in addition to being in the class $\mathcal{NP}$, are also easy to solve, i.e., there exist polynomial algorithms for problems in this class. Examples of such problems are the shortest path problems, minimum spanning tree problem and the assignment problem. The second class of problems in $\mathcal{NP}$ are said to belong to the class $\mathcal{NP}$-Complete. A problem is said to belong to the class $\mathcal{NP}$-Complete, if a polynomial time algorithm to solve the problem would lead to a polynomial time algorithm to solve every problem in the class $\mathcal{NP}$. Examples of problems in this class are the TSP and the QAP. Unfortunately, we do not have polynomial time algorithms to any problem in the $\mathcal{NP}$-Complete class. The question as to whether $\mathcal{P}$= $\mathcal{NP}$ remains one of the most well-known problems in Computer Science to date. In fact, it belongs to the seven so-called *Millennium Prize Problems*[2], which have a reward of $1,000,000 (Clay Mathematics Institute, 2006).

The $\mathcal{NP}$-Complete class of problems are defined for decision problems. In our study we deal with optimization versions of problems. For example, the optimization version of a TSP is the following. Find a shortest complete tour through n locations, given a cost matrix C. The optimization version of $\mathcal{NP}$-Complete problems are said to be $\mathcal{NP}$-hard problems.

Frequently encountered $\mathcal{NP}$-hard problems are the Traveling Salesman Problems, Facility Location Problems (Goldengorin *et al.*, 2004), and Integer Programming Problems (Cook *et al.*, 1998). The clustering problems from Chapter 5 are also $\mathcal{NP}$-hard; see Garey and Johnson (1979). An up-to-date list of $\mathcal{NP}$-hard problems is maintained in Kann and Crescenzi (2006).

[2]One of the seven problems, the Poincaré Conjecture, has been solved by Grigori Perelman.

1.4 Solving $\mathcal{NP}$-hard problems

We have seen that it can be very time-consuming to solve some large $\mathcal{NP}$-hard problem instances to optimality. Should we focus on improving quick methods that may return suboptimal solutions, or should we try to improve exact methods?

Heuristics are algorithms which return non-optimal solutions for some instances of a problem. Much recent scientific work is done on heuristics; see, for example, Affenzeller and Mayrhofer (2002). An important cause of the recent focus on heuristics is the emerging of many *online applications*, in which a decision must be taken almost instantaneously (González *et al.*, 2001). Since heuristics typically have short solution times, they are employed in online optimization.

A promising class of heuristics is called *meta-heuristics* (Glover and Kochenberger, 2003): solution methods that orchestrate an interaction between local improvement procedures and higher strategies, such as random construction of new solutions, to create a process capable of escaping from local optima. The problem with improvement heuristics is that they may become trapped in a local optimum; see Figure 1.2. A meta-heuristic uses strategies to escape from such a local optimum. The simulated annealing algorithm presented in Chapter 5 is a meta-heuristic. Other prominent classes of meta-heuristics are genetic algorithms, tabu search and variable neighborhood search (Glover and Kochenberger, 2003). Meta-heuristics are successfully applied on a wide range of problems, such as the TSP (Buriol *et al.*, 2004) and clustering problem (DeSarbo and Grisaffe, 1998). Solutions with at most 1% higher cost than the optimal solution are often obtained.

Research on exact methods certainly remains worthwhile. There are many problems in which the difference of 1% in solution value leads to millions of euros or dollars of extra costs compared to an optimal solution. This holds in particular for *strategic decision making*, where resources are allocated for long periods of time, such as the segmentation decisions from Chapter 5. Clearly, it is more important to support the decision accurately than to obtain a solution quickly. Improvements in exact methods enable us to solve problems larger and more difficult problem instances to optimality.

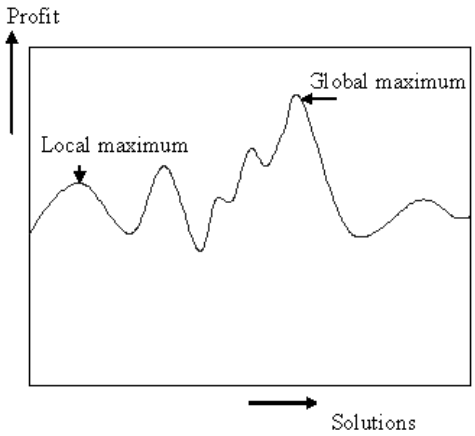

Figure 1.2. Local and global optima

Research on heuristics and exact algorithms can reinforce each other. In some applications, (meta-)heuristics and exact methods are combined into effective new algorithms; see for example Cotta and Troya (2003); Nwana *et al.* (2005) and Chapter 2.

The most frequently used type of exact methods for solving $\mathcal{NP}$-hard COPs is Branch and Bound (BnB); the next section is devoted this type of methods. Many variants of BnB exist, such as Branch and Cut (Naddef, 2002), Branch and Price, Branch and Peg (Goldengorin *et al.*, 2004), Branch and Win (Pastor and Corominas, 2004), and Cut and Solve (Climer and Zhang, 2006). Other exacts methods are Dynamic Programming, cutting plane methods, complete enumeration, and Data Correcting (Goldengorin, 2002).

1.5 The Branch and Bound Methodology

In this Section, the Branch and Bound (BnB) methodology is discussed. BnB methods have been applied successfully on various $\mathcal{NP}$-hard problems, such as Traveling Salesman Problems (Miller and Pekny, 1991; Carpaneto *et al.*, 1995), Mixed Integer Programming problems (Achterberg *et al.*, 2004; Sierksma, 1996), and Scheduling Problems (Baptiste *et al.*, 2004). State-of-the-art BnB methods are able to solve large instances to optimality. For example, the exact algorithm from Applegate *et al.* (2004) solved a non-trivial symmetric TSP instance con-

sisting of 33,810 locations in 2005 (Wikipedia, 2006).

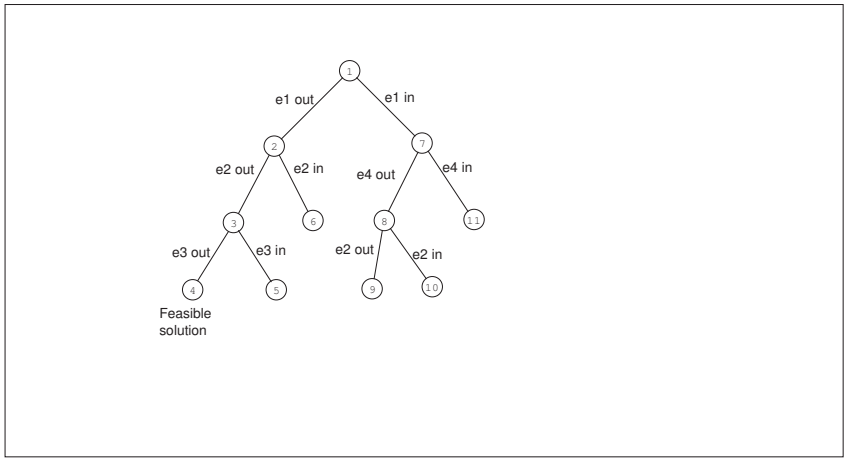

Figure 1.3. Example of a BnB search tree

The following phenomenon frequently occurs in Combinatorial Optimization: if certain constraints of the original hard problem are removed, the problem becomes easily solvable; see for example Schrijver (2003). Such a less constrained problem is called a *relaxation*. There are numerous examples of $\mathcal{NP}$-hard problems with easily solvable relaxations, the most famous example being Integer Programming and its Linear Programming relaxation.

BnB methods use easily solvable relaxations as follows. Initially, an easy relaxation is solved. If the solution obtained is feasible for the original hard problem, then this solution is optimal for the original problem as well and the BnB method for the partivular problem terminates; otherwise, the problem is divided into smaller and more restricted problems: the *subproblems*. The process continues until all subproblems are either solved or *discarded*, meaning that the subproblem is not relevant for determining an optimal solution. Another term for discarding is *fathoming*. BnB methods list all solutions in a search tree; see Figure 1.3. We refer to Ibaraki (1987) for a detailed description of the BnB process.

A BnB method for a minimization problem is characterized by the following four building blocks; see Miller and Pekny (1991):

Upper bound. Usually, the upper bound is the value of the best solution obtained

so far.

Lower bound. The lower bound of a subproblem should be smaller than or equal to the smallest value of any feasible solution of that subproblem. If a lower bound exceeds the value of the upper bound, we discard the corresponding subproblems.

Branching rule. At each node of a BnB search tree, a branching rule prescribes how the current problem should be partitioned into new subproblems.

Search strategy. The search strategy prescribes how the BnB algorithm should proceed through the search tree. The two most common methods are *Depth First Search*, which solves the most recently generated subproblem first, and *Best First Search*, which solves the most promising subproblem first.

A BnB algorithm for a maximization problem is similar, but the roles of upper bounds and lower bounds are exchanged.

1.6 Improvements to Branch and Bound methods

It is often thought that improvements on solution techniques are mainly due to increasing computing power, but this is not quite true. For example, Bixby (1994) finds that in the period 1980-1990, algorithmic improvements are the main cause of solution time reductions in Linear Programming. One cannot rely solely on increasing computing power; improvements in BnB algorithms are needed to solve large instances of $\mathcal{NP}$-hard problems to optimality.

This dissertation aims to improve the building blocks of BnB algorithms as follows:

- A subproblem is fathomed if the value of its lower bound exceeds the value of the best solution obtained so far, the *upper bound*. The more subproblems are fathomed, the smaller the search trees. Can we improve the upper bound, and if yes, does the time invested in such an improvement decrease total solution times? Chapter 2 describes the concept of *iterative patching*, a procedure to construct effective upper bounds for the Asymmetric Traveling Salesman Problem.

- Search tree reductions are achieved by decreasing the upper bounds, but also by increasing the lower bounds of subproblems. To this end, we increase the lower bound of each subproblem with so-called *upper tolerances* in Chapter 3 and 4. Again, we face the trade-off between the improvement in lower bounds and the time it takes to determine the bounds.

- At each node of a BnB search tree, the *branching rule* prescribes how the current problem should be partitioned into subproblems. A branching rule is effective if it preserves elements or structures which are also in an optimal solution of the hard problem and deletes the other ones. We introduce new *tolerance-based branching rules* in Chapter 3. We try to improve these branching rules in Chapter 4.

1.7 Clustering and market segmentation

In marketing, a company can distinguish itself from its competitors by offering each relevant group of customers a tailored marketing mix (Wedel and Kamakura, 1998). A group of consumers is only targeted well if the consumers in the group achieve similar scores on the relevant attributes, such as price sensitivity or income. They should respond more or less the same on the marketing efforts of the company, such as a price increase.

The grouping of subjects in similar groups is the so-called *Clustering Problem*. Clustering is, according to the definition from Mirkin and Muchnik (1998), "a mathematical technique for revealing classifications in the data collected on real world phenomena". Clustering Problems are encountered in a wide range of disciplines, such as computer science (Abrantes and Marques, 1996) and pattern recognition (Pawitan and Huang, 2003), but we concentrate on an application in market research. The clustering of consumers in marketing is called *market segmentation*.

In Steenkamp and Ter Hofstede (2002), it is noted that in many international segmentations studies, logistics costs are so high that organization resort to the segmentation strategy in which countries are taken as segments. The resulting marketing offers of the company fit consumer preferences poorly. In Chapter 5, the trade-off between logistics costs and the fit of consumer preferences is

explicitly made. To obtain such segmentations, we use meta-heuristics and BnB algorithms.