NHỮNG PHƯƠNG PHÁP DỰA TRÊN NHÁNH-VÀ-CẬN CHO NHỮNG BÀI TOÁN QUI HOẠCH NGUYÊN

BRANCH-AND-BOUND BASED METHODS FOR INTEGER PROGRAMS

Trần Văn Hoài

Khoa Công Nghệ Thông Tin, Đại học Bách khoa, Tp. Hồ Chí Minh, Việt nam
hoai@dit.hcmut.edu.vn

BẢN TÓM TẮT

Bài báo giới thiệu một lãnh vực nghiên cứu trong tối ưu tổ hợp, gọi là quy hoạch nguyên. Không giống những phương pháp tính toán mềm, hướng nghiên cứu này tập trung nghiên cứu những lý thuyết và phương pháp hiệu quả trong việc tìm kiếm lời giải tối ưu toàn cục. Trong đó, branch-and-bound kết hợp với việc tạo những mặt cắt là một phương pháp khung nổi tiếng. Nhiều khía cạnh từ ứng dụng, mô hình đến phương pháp giải sẽ được khảo sát và trình bày tóm tắt. Lãnh vực này cũng là một hướng ứng dụng của tính toán song song đang là một xu thế phát triển của tính toán khoa học hiệu năng cao. Việc giải các bài toán tối ưu có khối lượng tính toán lớn trong thời gian hợp lý là rất khả thi.

ABSTRACT

In the paper, we present an interesting research area in combinatorial optimization, called integer programming. Not as non-exact methods, this direction primarily focuses on finding optimal solutions efficiently. Branch-and-bound with cutting plane generation is a well-known framework to solve integer programs. All its aspects ranged from applications, models to solution methods will be briefly surveyed. Moreover, the area is also considered under a view of parallel computing, which is an increasingly popular trend in high performance scientific computing. It is promising to solve large scale optimization problems to optimality within a reasonable computation time.

1. Integer programming and applications

Variables of optimization problems fall into two categories: *continuous* variables, and *discrete* variables. We call those problems with discrete variables *combinatorial optimization problems* which look for an object from a *finite*, or possibly countably infinite, set. More specifically, we are given a finite set $N=\{1,\cdots,n\}$, weight for each $j \in N$, and a set F of feasible subsets of N. The problem of finding a minimum weight feasible subset is a combinatorial optimization problem

$$\text{(COP)} \quad \min_{S \subseteq N} \left\{ \sum\nolimits_{j \in S} c_j : S \in F \right\} \qquad (1)$$

It is known that a COP can be formulated as an *integer program* (IP), which is a special form of a *linear mixed integer program*

$$\text{(MIP)} \quad \begin{array}{ll} \min & c^{T}x + h^{T}y \\ s.t. & Ax + Gy = b \\ & x \in Z_{+}^{n},\, y \in R_{+}^{n} \end{array} \qquad (2)$$

In an IP, there is no continuous variable. If we restrict the variables x as $x \in \{0,1\}^n$, we have a *binary problem* (BIP).

A wide variety of practical applications can be modeled and solved using integer programming. Some of them are:

- Airline crew scheduling: Given a flight set of an aircraft type operating in a schedule horizon. The problem is to assign crew to serve those flights with a minimum operational cost. Moreover, the solution also has to satisfy a complicated set of constraints coming from labor laws, contractual agreements, regulations.

- Electricity generation planning: The problem is to develop an hourly schedule spanning a day or a week so as to decide which generators will be producing and at what levels. We also keep the solution constrained within a peak demand, satisfying estimated hourly or half-hourly demand. Many more constraints should be included in practical applications.
- Network designing: Computer networking and telecommunication are two demanding areas nowadays. One of the problems in these areas is how to install a new capacity to satisfy a predicted demand and minimize the installation cost. The possibility of failure of a line of center due to breakdown or accident should be taken into account.
- Timetabling: Scheduling classes and examinations to avoid conflicts, without violating resources available, such as number of class rooms, room capacity, and other hard and soft constraints are computationally difficult problems. One of a simple model for the problem is set partitioning model, which is a BIP.

2. Sequential solution methods

2.1. Cutting plane algorithms

Solving large scale integer programs is still challenging problem involving not only general algorithms but also problem specific techniques. General algorithms can be classified into 2 types: *non-exact* algorithms and *exact* ones. Non-exact methods are mainly based on *heuristics* or *approximation algorithms*. Greedy and local search algorithms could be the first choice of researchers to attack a combinatorial optimization problem. Unfortunately, under the local search mechanism, we can get trapped in a local minimum. In order to escape from that, we need an improved heuristic. Well-known techniques which can help us in such a case are *tabu search*, *simulated annealing*, and *genetic algorithms*. They are called *meta-heuristics*.

Tabu search was started by Glover (1986) and strongly discussed in two following texts (Glover, 1989, 1990). The idea behind the method is to avoid cycling in the process of moving among feasible solutions. Simulated annealing approaches in a different way by choosing randomly a neighbor to move to. Precisely, the neighbor will be chosen with the probability of 1 if it has a better cost value, otherwise, with some probability strictly between 0 and 1. The method is described in Metropolis et al. (1953); Kirkpatrick et al. (1983). Genetic algorithms originated with the work of Holland (1975); Goldberg (1989). Not working with an individual solution, at each iteration, the method considers a finite set of solutions (called *population*) and this set will change randomly from one generation to the next.

If one prefers an exact approach, *cutting plane algorithms* and *branch-and-bound algorithms* are widely used methods. Furthermore, they can also be used in non-exact heuristics. Given an IP: $\min\{c^{\text{Î¤}}x : Ax=b, x\in Z^n\}$, a cutting plane method tries to find out the description of the convex hull of the set of feasible solutions. The method iteratively cuts off fractional points of the polyhedron $P=\{x\in R^n : Ax=b\}$ by valid inequalities. In order to understand the algorithm, we must perceive the following definition.

Definition: *The separation problem* associated with an integer program $\min\{c^{\text{Î¤}}x : x\in X\subseteq R^n\}$ is the problem: given $x^{\text{¿}}\in R^n$, is $x\in conv(X)$? If not, find a valid inequality $\text{Ï€}^{\text{Î¤}}x\le \text{Ï€}_0$ for X, but violated by the point.

The equivalence of optimization and separation is shown by Grötschel et al. (1981).

In Algorithm **Error! Reference source not found.** which describes a basic cutting plane algorithm, F in the algorithm denotes a family of valid inequalities for X. Certainly, the algorithm can terminate without finding an integral solution. Moreover, although several cutting plane algorithms (e.g., the cutting plane algorithm with Gomory cuts) were proved, under some circumstances, to be of finite convergence, it is still not practical to continue the loop until no violated valid inequality is found. The exit from the loop gives an improved formulation which can be the input of a branch-and-bound method to be discussed in the next section.

Algorithm 2.1: A cutting plane algorithm

```
  $t \leftarrow 0, P^0 = P$
While( stopping condition not reached) do
  Solve the linear program $x^t = \text{argmin}_x \{c^{\text{Î¤}} x : x \in P^t\}$
    If $x^t \in Z^n$ then
      $x^t$ is an optimal solution
      Break
    Else
      Solve the separation problem for $x^t$ and the family $F$
      If $\exists(\text{Ï€}, \text{Ï€}_0) \in F$ such that $\text{Ï€}^{\text{Î¤}} x > \text{Ï€}_0$ then
        $P^{t+1} \leftarrow P^t \cap \{\text{Ï€}^{\text{Î¤}} x \le \text{Ï€}_0\}$
        $t \leftarrow t+1$
      Else
        Break
      End if
    End if
End while
```

Links to integer programming can be found in many textbooks on combinatorial optimization or integer programming, such as Nemhauser and Wolsey (1988); Papadimitriou and Steiglitz (1998); Wolsey (1998). A book of Kallrath and Wilson (1997) which shows the integer programming problem under the application's point of view could be helpful to those who want to solve practical problems.

2.2. Branch-and-bound

The so-called *branch-and-bound* method is based on two following conclusions:

Proposition 2.2 Consider the problem $z = \min\{c^{\text{Î¤}} x : x \in X\}$. Let $X = X_1 \cup \cdots \cup X_K$ be a decomposition of X into smaller sets, and let $z^k = \min\{c^{\text{Î¤}} x : x \in X^k\}$ for $k = 1, \cdots, K$. Then $z = \min_k z^k$.

Proposition 2.3 Use the same notations as in Proposition **Error! Reference source not found.**. Let $\underline{z}^k$ be a lower bound on z^k, and $\overline{z}^k$ be an upper bound on z^k. Then $\underline{z} = \min_k \underline{z}^k$ is a lower bound on z and $\overline{z} = \min_k \overline{z}^k$ is an upper bound on z.

With the help of Proposition **Error! Reference source not found.**, many smaller problems finding z^k do not need to be solved or be stopped quickly because they violate some bounds. The idea is called *implicit enumeration* which contrasts to *explicit enumeration* which explores totally the feasible region. It is not necessary to solve smaller problems to optimality. Instead, upper or lower bounds are enough for the branch-and-bound method (otherwise, we can decompose those problems to smaller parts). Hence, a *relaxation* of an IP is needed, being defined as an optimization problem (RP) with the feasible set $X_R \supseteq X$, and the cost function $z_R(x) \le c^T x$ for $x \in X$. Algorithm **Error! Reference source not found.** gives a general branch-and-bound algorithm for solving IP: $\min\{c^{\text{Î¤}} x : x \in X\}$.

Algorithm 2.2: A general branch-and-bound algorithm

```
$L \leftarrow \{(\mathrm{IP})\}$ , $X^0 \leftarrow X$ , $\underline{z}^0 \leftarrow -\infty$ , $\bar{z}_{\mathrm{IP}} \leftarrow \infty$
**While**( $L$ not empty) **do**
    Select and delete a problem $(\mathrm{IP})^i$ from $L$
    Solve its relaxation $(\mathrm{RP})^i$ with optimal value $z_R^i$ and optimal solution $x_R^i$ (if they exist) {*Problem relaxation*}
    **If** $z_R^i < \bar{z}_{\mathrm{IP}}$ **then** {*bounding*}
        **If** $x_R^i \in X^i$ and $c^{Î¤} x_R^i < \bar{z}_{\mathrm{IP}}$ **then**
            $\bar{z}_{\mathrm{IP}} \leftarrow c^{Î¤} x_R^i$
            Delete from $L$ all problems with $\underline{z}^i \le \bar{z}_{\mathrm{IP}}$ . {*fathoming*}
            **If** $c^{Î¤} x_R^i = z_R^i$ **then**
                **Continue**
            **End if**
        **End if**
        Decompose $X^i$ into $\{X^{ij}\}_{j=1}^k$ and add associated problems $\{(\mathrm{IP})^i\}_{j=1}^k$ to $L$ , where $\underline{z}^{ij} = z_R^i$ for $j = 1, \cdots, k$ . {*branching*}
    **End if**
**End while**
```

In the algorithm, there are several main interesting questions as follows.

- How to select the next open subproblem $(\mathrm{IP})^i$? A good selection can help finding optimal solutions quickly. However, the selection strategy should also consider a trade-off between keeping the number of explored nodes in the search tree and staying within the memory capacity of the computer in use.
- How to decompose X^i ? Generally speaking, the search space is subdivided by addition of constraints to normally disjoint subproblems.
- How to make a “good” relaxation ? Linear relaxation is often a choice in many branch-and-bound codes partly due to the linear presentation of the remaining problem if we drop the integral constraints of an integer problem. Cutting plane generation is also a main approach to tighten the feasible region of a subproblem.
- How to improve the upper bound by non-exact methods ? Reducing the gap between lower bound and upper is quite crucial in solving large scale integer programs.

Obviously, exploring a branch-and-bound tree of an instance of an optimization problem is a time consuming task, especially with *NP*-hard problems. Parallel computing which utilize computing resources of parallel machines is one of methods to explore branch-and-bound trees efficiently.

3. Parallel branch-and-bound

In order to parallelize a sequential branch-and-bound solver, we have to investigate many aspects of parallel architecture and parallel programming. In this paper, we only focus on distributed memory systems because of their popularity and high performance. Message-passing is a well-known parallel programming model for such systems. These systems is also scalable up to hundreds or thousands processors. Additionally, there exist good implementations, such as MPI (message passing interface), PVM (parallel virtual machine), to support developers. A combination of integer programming and parallel computing is increasingly pervasive.

As mentioned in the previous section, the tree search with bounding is the main point of branch-and-bound. Therefore, the parallelism is normally performed on the tree level. This means that open subproblems are smallest computational units processed by processors and transferred among them.

In order to implement an efficient parallel branch-and-bound solver, we should consider main interesting issues below.

- Design scheme: In the sequential design of branch-and-bound, a pool of open subproblems exists in a unique main memory. However, a parallel solver can use a centralized pool or several distributed pools. In the centralized scheme, we can have a complete knowledge on tree search (bounds, cutting planes, etc.), but the master possibly becomes a bottleneck. Otherwise, synchronizing several pools distributed in memory of different processors is a hard problem. But, the distributed scheme is more scalable.

- Portable and easy-to-use design: Scientists in combinatorial optimization area can have little knowledge on parallel computing. But, in many cases, they really want to speedup the process of finding optimal solution or search a good solution for their problems. If we want to provide them with a parallel branch-and-bound framework, we should consider the ability to transform a sequential code to a corresponding parallel code easily. The library will be more portable with MPI or PVM as communication library.
- Load balancing: Besides general load balancing aspects, a parallel branch-and-bound should take into account aspects of node selection strategy and branching strategy. They are the sources of creating open subproblems and controlling the search direction.

There are still many theoretical and practical aspects which cannot be presented in this paper. We can find them in Clausen (1997); Ralphs et al. (2003).

4. Conclusion

Integer programs are used to model many practical applications. However, it is classified as an *NP*-complete problem which cannot be solved by an efficient algorithms. Therefore, carefully investigating practical properties of an application can help us to present a good model and suggest an efficient problem-specific solution method. This is also one of our research direction into real world applications. Another area in our future plan is to apply parallel and distributed computing to integer programming in order to speedup the solution process.

References

J. Clausen. *Parallel Computing in Optimization*, chapter Parallel Branch-and-Bound - Principles and Personal Experiences, pages 239–267. Kluwer Academic Publishers, 1997.

F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13:533–549, 1986.

F. Glover. Tabu Search – Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

F. Glover. Tabu Search – Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, pages 1–24. Addison-Wesley, 1989.

M. Grötschel, L. Lovász, and A. Schrijver. The Ellipsoid Method and its Consequences in Combinatorial Optimization. *Combinatorica*, 1(2):169–197, 1981.

J. H. Holland. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

J. Kallrath and J. M. Wilson. *Business Optimization*. MacMillan Business, London, 1997.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimisation by Simulated Annealing. *Science*, 220:671–680, 1983.

N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.

C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization (Algorithms and Complexity)*. Prentice-Hall, New Jersey, 1998.

T. K. Ralphs, L. Ladányi, and M. J. Saltzman. Parallel Branch, Cut, and Price for Large Scale Discrete Optimization. Technical report, Department of Mathematical Sciences, IBM, 2003.

L. A. Wolsey. *Integer Programming*. John Wiley, New York, 1998.