WIKIPEDIA

Integer programming

An **integer programming** problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. In many settings the term refers to **integer linear programming** (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.

Integer programming is NP-complete. In particular, the special case of 0-1 integer linear programming, in which unknowns are binary, and only the restrictions must be satisfied, is one of Karp's 21 NP-complete problems.

If some decision variables are not discrete the problem is known as a **mixed-integer programming**.[1]

Contents

Canonical and standard form for ILPs

An integer linear program in canonical form is expressed as:[2]

$$
\begin{array}{ll}
\text{maximize} & \mathbf{c}^{\mathbf{T}}\mathbf{x} \\
\text{subject to} & A\mathbf{x} \le \mathbf{b}, \\
& \mathbf{x} \ge \mathbf{0}, \\
\text{and} & \mathbf{x} \in \mathbb{Z}^{n},
\end{array}
$$

and an ILP in standard form is expressed as

$$\begin{aligned}
\text{maximize} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\
\text{subject to} \quad & A\mathbf{x} + \mathbf{s} = \mathbf{b}, \\
& \mathbf{s} \geq \mathbf{0}, \\
& \mathbf{x} \geq \mathbf{0}, \\
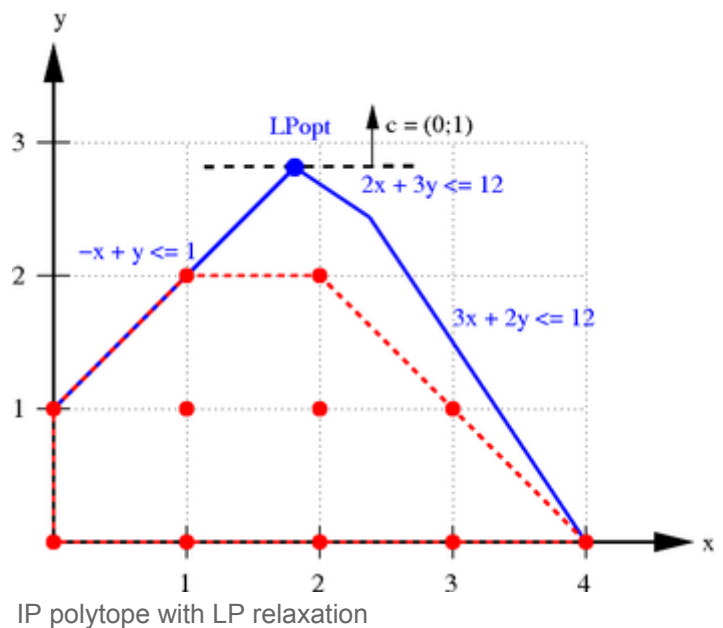\text{and} \quad & \mathbf{x} \in \mathbb{Z}^n,
\end{aligned}$$

where $\mathbf{c}, \mathbf{b}$ are vectors and A is a matrix, where all entries are integers. As with linear programs, ILPs not in standard form can be converted to standard form by eliminating inequalities, introducing slack variables ($\mathbf{s}$) and replacing variables that are not sign-constrained with the difference of two sign-constrained variables

Example

The graph on the right shows the following problem.

$$\begin{aligned}
\max \; & y \\
-x + y & \leq 1 \\
3x + 2y & \leq 12 \\
2x + 3y & \leq 12 \\
x, y & \geq 0 \\
x, y & \in \mathbb{Z}
\end{aligned}$$

The feasible integer points are shown in red, and the red dashed lines indicate their convex hull, which is the smallest polyhedron that contains all of these points. The blue lines together with the coordinate axes define the polyhedron of the LP relaxation, which is given by the inequalities without the integrality constraint. The goal of the optimization is to move the black dotted line as far upward while

IP polytope with LP relaxation

still touching the polyhedron. The optimal solutions of the integer problem are the points $(1, 2)$ and $(2, 2)$ which both have an objective value of 2. The unique optimum of the relaxation is $(1.8, 2.8)$ with objective value of 2.8. Note that if the solution of the relaxation is rounded to the nearest integers, it is not feasible for the ILP.

Proof of NP-hardness

The following is a reduction from minimum vertex cover to integer programming that will serve as the proof of NP-hardness.

Let $G = (V, E)$ be an undirected graph. Define a linear program as follows:

$$\begin{aligned}
\min \sum_{v \in V} & y_v \\
y_v + y_u \geq 1 \quad & \forall uv \in E \\
y_v \geq 0 \quad & \forall v \in V \\
y_v \in \mathbb{Z} \quad & \forall v \in V
\end{aligned}$$

Given that the constraints limit y_v to either 0 or 1, any feasible solution to the integer program is a subset of vertices. The first constraint implies that at least one end point of every edge is included in this subset. Therefore, the solution describes a vertex cover. Additionally given some vertex cover C, y_v can be set to 1 for any $v \in C$ and to 0 for any

$v \notin C$ thus giving us a feasible solution to the integer program. Thus we can conclude that if we minimize the sum of y_v we have also found the minimum vertex cover.[3]

Variants

Mixed integer linear programming (**MILP**) involves problems in which only some of the variables, x_i, are constrained to be integers, while other variables are allowed to be non-integers.

Zero-one linear programming involves problems in which the variables are restricted to be either 0 or 1. Note that any bounded integer variable can be expressed as a combination of binary variables.[4] For example, given an integer variable, $0 \le x \le U$, the variable can be expressed using $\lfloor \log_2 U \rfloor + 1$ binary variables:

$$x = x_1 + 2x_2 + 4x_3 + \cdots + 2^{\lfloor \log_2 U \rfloor} x_{\lfloor \log_2 U \rfloor + 1}.$$

Applications

There are two main reasons for using integer variables when modeling problems as a linear program:

1. The integer variables represent quantities that can only be integer. For example, it is not possible to build 3.7 cars.
2. The integer variables represent decisions and so should only take on the value 0 or 1 .

These considerations occur frequently in practice and so integer linear programming can be used in many applications areas, some of which are briefly described below.

Production planning

Mixed integer programming has many applications in industrial production, including job-shop modelling. One important example happens In agricultural production planning involves determining production yield for several crops that can share resources (e.g. Land, labor, capital, seeds, fertilizer, etc.). A possible objective is to maximize the total production, without exceeding the available resources. In some cases, this can be expressed in terms of a linear program, but variables must be constrained to be integer.

Scheduling

These problems involve service and vehicle scheduling in transportation networks. For example, a problem may involve assigning buses or subways to individual routes so that a timetable can be met, and also to equip them with drivers. Here binary decision variables indicate whether a bus or subway is assigned to a route and whether a driver is assigned to a particular train or subway.

Telecommunications networks

The goal of these problems is to design a network of lines to install so that a predefined set of communication requirements are met and the total cost of the network is minimal.[5] This requires optimizing both the topology of the network along with the setting the capacities of the various lines. In many cases, the capacities are constrained to be integer quantities. Usually there are, depending on the technology used, additional restrictions that can be modeled as linear inequalities with integer or binary variables.

Cellular networks

The task of frequency planning in GSM mobile networks involves distributing available frequencies across the antennas so that users can be served and interference is minimized between the antennas.[6] This problem can be formulated as an integer linear program in which binary variables indicate whether a frequency is assigned to an antenna.

Algorithms

The naive way to solve an ILP is to simply remove the constraint that **x** is integer, solve the corresponding LP (called the LP relaxation of the ILP), and then round the entries of the solution to the LP relaxation. But, not only may this solution not be optimal, it may not even be feasible; that is, it may violate some constraint.

Using total unimodularity

While in general the solution to LP relaxation will not be guaranteed to be integral, if the ILP has the form $\max \mathbf{c}^{\mathbf{T}}\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$ where $A, \mathbf{b},$ and $\mathbf{c}$ have all integer entries and A is totally unimodular, then every basic feasible solution is integral. Consequently, the solution returned by the simplex algorithm is guaranteed to be integral. To show that every basic feasible solution is integral, let **x** be an arbitrary basic feasible solution . Since **x** is feasible, we know that $A\mathbf{x} = \mathbf{b}$. Let $\mathbf{x}_0 = [x_{n_1}, x_{n_2}, \cdots, x_{n_j}]$ be the elements corresponding to the basis columns for the basic solution **x**. By definition of a basis, there is some square submatrix B of A with linearly independent columns such that $B\mathbf{x}_0 = \mathbf{b}$.

Since the columns of B are linearly independent and B is square, B is nonsingular, and therefore by assumption, B is unimodular and so $\det(B) = \pm 1$. Also, since B is nonsingular, it is invertible and therefore $\mathbf{x}_0 = B^{-1}\mathbf{b}$. By definition, $B^{-1} = \dfrac{B^{adj}}{\det(B)} = \pm B^{adj}$. Note that B^{adj} denotes the adjugate of B and is integral because B is integral. Therefore,

$$\Rightarrow B^{-1} = \pm B^{adj} \text{ is integral.}$$
$$\Rightarrow x_0 = B^{-1}b \text{ is integral.}$$
$$\Rightarrow \text{Every basic feasible solution is integral.}$$

Thus, if the matrix A of an ILP is totally unimodular, rather than use an ILP algorithm, the simplex method can be used to solve the LP relaxation and the solution will be integer.

Exact algorithms

When the matrix A is not totally unimodular, there are a variety of algorithms that can be used to solve integer linear programs exactly. One class of algorithms are cutting plane methods which work by solving the LP relaxation and then adding linear constraints that drive the solution towards being integer without excluding any integer feasible points.

Another class of algorithms are variants of the branch and bound method. For example, the branch and cut method that combines both branch and bound and cutting plane methods. Branch and bound algorithms have a number of advantages over algorithms that only use cutting planes. One advantage is that the algorithms can be terminated early and as long as at least one integral solution has been found, a feasible, although not necessarily optimal, solution can be returned. Further, the solutions of the LP relaxations can be used to provide a worst-case estimate of how far from optimality the returned solution is. Finally, branch and bound methods can be used to return multiple optimal solutions.

Lenstra in 1983 showed [7] that, when the number of variables is fixed, the feasibility integer programming problem can be solved in polynomial time.

Heuristic methods

Since integer linear programming is NP-hard, many problem instances are intractable and so heuristic methods must be used instead. For example, tabu search can be used to search for solutions to ILPs.[8] To use tabu search to solve ILPs, moves can be defined as incrementing or decrementing an integer constrained variable of a feasible solution, while keeping all other integer-constrained variables constant. The unrestricted variables are then solved for. Short term memory can consist of previous tried solutions while medium term memory can consist of values for the integer constrained variables that have resulted in high objective values (assuming the ILP is a maximization problem). Finally, long term memory can guide the search towards integer values that have not previously been tried.

Other heuristic methods that can be applied to ILPs include

- Hill climbing
- Simulated annealing
- Reactive search optimization
- Ant colony optimization
- Hopfield neural networks

There are also a variety of other problem-specific heuristics, such as the k-opt heuristic for the travelling salesman problem. Note that a disadvantage of heuristic methods is that if they fail to find a solution, it cannot be determined whether it is because there is no feasible solution or whether the algorithm simply was unable to find one. Further, it is usually impossible to quantify how close to optimal a solution returned by these methods is.

References

1. "Mixed-Integer Linear Programming (MILP): Model Formulation" (http://macc.mcmaster.ca/maccfiles/chachuatnotes/07-MILP-I_handout.pdf) (PDF). Retrieved 16 April 2018.
2. Papadimitriou, C. H.; Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Mineola, NY: Dover. ISBN 0486402584.
3. Erickson, J. (2015). "Integer Programming Reduction" (https://web.archive.org/web/20150518072946/https://courses.engr.illinois.edu/cs498dl1/sp2015/solutions/hw10sol.pdf) (PDF). Archived from the original (https://courses.engr.illinois.edu/cs498dl1/sp2015/solutions/hw10sol.pdf) (PDF) on 18 May 2015.
4. Williams, H.P. (2009). *Logic and integer programming*. International Series in Operations Research & Management Science. **130**. ISBN 978-0-387-92280-5.
5. Borndörfer, R.; Grötschel, M. (2012). "Designing telecommunication networks by integer programming" (http://www.zib.de/groetschel/teaching/SS2012/120503Vorlesung-DesigningTelcomNetworks-reduced.pdf) (PDF).
6. Sharma, Deepak (2010). "Frequency Planning" (http://www.slideshare.net/deepakecrbs/gsm-frequency-planning).
7. H.W. Lenstra, "Integer programming with a fixed number of variables", Mathematics of operations research, Vol 8, No 8, November 1983
8. Glover, F. (1989). "Tabu search-Part II". *ORSA Journal on computing*. **1** (3): 4–32. doi:10.1287/ijoc.2.1.4 (https://doi.org/10.1287%2Fijoc.2.1.4).

Further reading

- George L. Nemhauser; Laurence A. Wolsey (1988). *Integer and combinatorial optimization*. Wiley. ISBN 978-0-471-82819-8.
- Alexander Schrijver (1998). *Theory of linear and integer programming*. John Wiley and Sons. ISBN 978-0-471-98232-6.
- Laurence A. Wolsey (1998). *Integer programming*. Wiley. ISBN 978-0-471-28366-9.
- Dimitris Bertsimas; Robert Weismantel (2005). *Optimization over integers*. Dynamic Ideas. ISBN 978-0-9759146-2-5.
- John K. Karlof (2006). *Integer programming: theory and practice.* CRC Press. ISBN 978-0-8493-1914-3.
- H. Paul Williams (2009). *Logic and Integer Programming*. Springer. ISBN 978-0-387-92279-9.

- Michael Jünger; Thomas M. Liebling; Denis Naddef; George Nemhauser; William R. Pulleyblank; Gerhard Reinelt; Giovanni Rinaldi; Laurence A. Wolsey, eds. (2009). *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer. ISBN 978-3-540-68274-5.
- Der-San Chen; Robert G. Batson; Yu Dang (2010). *Applied Integer Programming: Modeling and Solution*. John Wiley and Sons. ISBN 978-0-470-37306-4.

External links

- A Tutorial on Integer Programming (http://mat.gsia.cmu.edu/orclass/integer/integer.html)
- Conference Integer Programming and Combinatorial Optimization, IPCO (http://www.mathopt.org/?nav=ipco)
- The Aussois Combinatorial Optimization Workshop (http://www.iasi.cnr.it/aussois)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Integer_programming&oldid=836805197"

This page was last edited on 16 April 2018, at 23:31.