PART II: ANALYZING INFEASIBILITY

As mathematical models grow larger and more complex, infeasibility happens more often during the process of model formulation, and is harder to diagnose than ever before. A linear program may have hundreds of thousands or even millions of constraints: which of these are causing the infeasibility and how should the problem be repaired? In nonlinear programs the issue is even more vexed: the model may be truly infeasible, or the solver may just have been given a poor starting point from which it is unable to reach feasibility.

Some form of automated or semi-automated assistance in diagnosing and repairing infeasibility is necessary in the face of the scale and complexity of modern optimization models. Fortunately, algorithmic tools have been developed in recent years. There are three main approaches. The first is the identification of an *Irreducible Infeasible Subset (IIS)* of constraints within the larger set of constraints defining the model. An IIS has the property that it is infeasible, but becomes feasible if any one or more of its constraints are removed; it is irreducible in that sense. Identifying an IIS allows the modeler to focus attention on a small set of conflicts within the larger model. Further refinements of the base algorithms try to return IISs that are of small cardinality, or that are easier for humans to understand. Other issues include trade-offs between the speed of identifying an IIS and the cardinality of the IIS that is returned.

The second main approach to analyzing infeasibility is to identify a *Maximum Feasible Subset* (MAX FS) of constraints within the larger set of constraints defining the model. This naturally focuses the analysis on the constraints that do not appear in this subset, i.e. the minimum cardinality set of constraints that must be removed so that the remainder constitutes a feasible set. Identifying a maximum feasible subset is an NP-hard problem, so the methods for doing so are clever heuristics.

Both of these approaches to analyzing infeasibility focus attention on a small part of a large model so that the modeler can determine how to repair it using human understanding of the meanings of the constraints. However the third approach seeks to suggest the best repair for the model, where "best" can be defined in various ways that can be handled algorithmically, e.g. the fewest changes to constraint right hand side values. The suggested repair can of course be accepted, modified, or rejected by the human modeler.

Many of the methods for analyzing infeasibility that are described in Part II depend on the ability of a solver to determine the feasibility status (feasible or infeasible) of an arbitrary set of constraints with very high accuracy. This ability is by and large available for linear programs, but it is much more problematic for nonlinear programs and for mixed-integer programs. In those cases, we may have to settle for the identification of an infeasible subset that is not irreducible (but is significantly smaller than the original set of constraints one hopes), or a *minimal intractable subset (MIS)* which is a minimal subset of constraints that causes a solver to report infeasibility under stated solver parameter settings (initial point, tolerances, etc.). For this reason, there are differing expectations for the success of the general analysis algorithms depending on the type of optimization model. However there are methods that are special to each type of model.

As you will see in Part II, effective algorithms for the analysis of infeasibility in linear programs and linear networks exist, and have been implemented in most commercial LP solvers. The situation is not yet so positive for nonlinear and mixed-integer programs, but research in this area is active and ongoing, with frequent new developments. Significant breakthroughs are likely in the relatively near future, especially as improved algorithms for reaching feasibility quickly reach maturity (see Part I).

Some of the algorithmic tools are best integrated directly into the solvers rather than into separate analysis software. This is the case for many of the infeasibility analysis algorithms since they make use of data that is available during the solution or re-solution of the problem. This is certainly the case for several of the algorithms for analyzing infeasible linear programs that use information from the final phase 1 basis, and thereafter from bases produced by repeated solutions of slightly differing versions of the model. Algorithms in this class benefit from the use of hot-starts based on the immediately previous solution.

Infeasibility analysis is part of larger efforts in *computer-assisted analysis* of complex optimization models originated by Greenberg (1981a, 1981b, 1983). He developed software such as ANALYZE (Greenberg 1983) which provides tools for the manipulation and analysis of linear programs. PERUSE (Kurator and O'Neill 1980) was another early system which permitted interactive query of the LP matrix and solution. MProbe (Chinneck 2001) is a more recent system that provides various tools for general probing of optimization models, particularly nonlinear programs. Practical approaches to infeasibility analysis in particular date to the 1970s, notably in the *Refinery and Petrochemical Modeling System* by Bonner & Moore (1979). Other model-specific approaches for infeasibility analysis were developed by Harvey Greenberg as part of his *Intelligent Mathematical Programming System* (IMPS) project originating in the early 1980s (see Greenberg (1987c, 1991) for a description of the IMPS project and Greenberg (1996b) for a summary of relevant literature). In contrast, the IIS isolation approach developed in Chap. 6 is independent of the particular model and has been developed for general solvers applicable to any LP.

The eventual goal in computer-assisted analysis is the development of a complete environment supporting optimization modeling, similar to the environments enjoyed by general software developers that include debuggers, profilers and other useful tools. This is an essential part of the verification and validation of optimization models. There has been significant progress in the development of techniques and tools supporting the debugging of complex optimization models. Most modern LP solvers now include routines for isolating IISs for instance, and model debugging is now included as a topic in modern textbooks on optimization. For example, see the textbook by Pannell (1997) for an excellent discussion of how to debug a linear program, including infeasibility analysis.