Handbook of Combinatorial Optimization
D.-Z. Du and P.M. Pardalos (Eds.) pp. – – –
©1998 Kluwer Academic Publishers

Interior Point Methods for Combinatorial Optimization

John E. Mitchell
Mathematical Sciences
Rensselaer Polytechnic Institute, Troy, NY 12180 USA
E-mail: `mitchj@rpi.edu`

Panos M. Pardalos
Center for Applied Optimization, ISE Department
University of Florida, Gainesville, FL 32611 USA
E-mail: `pardalos@ufl.edu`

Mauricio G. C. Resende
Information Sciences Research
AT&T Labs Research, Florham Park, NJ 07932 USA
E-mail: `mgcr@research.att.com`

Contents

1 Introduction **2**

2 Combinatorial optimization **3**
2.1 Examples of combinatorial optimization problems 3
2.2 Scope and computational efficiency 7

3 Solution techniques **13**
3.1 Combinatorial approach 13
3.2 Continuous approach 14
3.2.1 Examples of embedding 14
3.2.2 Global approximation 15
3.2.3 Continuous trajectories 15
3.2.4 Topological properties 17

4 Interior point methods for linear and network programming **18**
4.1 Linear programming 18
4.2 Network programming 21
4.3 Components of interior point network flow methods 25

4.3.1 The dual affine scaling algorithm 26
4.3.2 Computing the direction 27
4.3.3 Network preconditioners for conjugate gradient method 29
4.3.4 Identifying the optimal partition 32
4.3.5 Recovering the optimal flow 36

5 Branch and bound methods **39**
5.1 General concepts 40
5.2 An example: The QAP 42

6 Branch and cut methods **49**
6.1 Interior point cutting plane methods 52
6.2 Solving the relaxations approximately 54
6.3 Restarting 55
6.4 Primal heuristics and termination 57
6.5 Separation routines 57
6.6 Fixing variables 57
6.7 Dropping constraints 58
6.8 The complete algorithm 58
6.9 Some computational results 60
6.10 Combining interior point and simplex cutting plane algorithms 60
6.11 Interior point column generation methods for other problems 61
6.12 Theoretical issues and future directions 62

7 Nonconvex potential function minimization **63**
7.1 Computing the descent direction 68
7.2 Some computational considerations 71
7.3 Application to combinatorial optimization 73

8 A lower bounding technique **74**

9 Semidefinite Programming Relaxations **78**

10 Concluding Remarks **81**

References

1 Introduction

Interior-point methods, originally invented in the context of linear programming, have found a much broader range of applications, including *discrete* problems that arise in computer science and operations research as well as continuous computational problems arising in the natural sciences and engineering. This chapter describes the conceptual basis and applications of interior-point methods for discrete problems in computing.

The chapter is organized as follows. Section 2 explains the nature and scope of combinatorial optimization problems and illustrates the use of interior point approaches for these problems. Section 3 contrasts the combinatorial and continuous approaches for solving discrete problems and elaborates on the main ideas underlying the latter approach. The continuous approach constitutes the conceptual foundation of interior-point methods. Section 4 is dedicated to interior point algorithms for linear and network optimization. Sections 5 and 6 discuss branch-and-bound and branch-and-cut methods based on interior point approaches. Sections 7 and 8 discuss the application of interior point techniques to minimize nonconvex potential functions to find good feasible solutions to combinatorial optimization problems as well as good lower bounds. In Section 9, a brief introduction to semidefinite programming techniques and their application to combinatorial optimization is presented. We conclude the paper in Section 10 by observing the central role played by optimization in both natural and man-made sciences. We provide selected pointers to web sites constaining up-to-date information on interior point methods and their applications to combinatorial optimaization.

2 Combinatorial optimization

In this section, we discuss several examples of combinatorial optimization problems and illustrate the application of interior point techniques to the development of algorithms for these problems.

2.1 Examples of combinatorial optimization problems

As a typical real-life example of combinatorial optimization, consider the problem of operating a flight schedule of an airline at minimum cost. A flight schedule consists of many flights connecting many cities, with specified arrival and departure times. There are several operating constraints. Each plane must fly a round trip route. Each pilot must also fly a round trip route, but not necessarily the same route taken by the plane, since at each airport the pilot can change planes. There are obvious timing constraints interlocking the schedules of pilots, planes and flights. There must be adequate rest built into the pilot schedule and periodic maintenance built into the plane schedule. Only certain crews are qualified to operate certain types of planes. The operating cost consists of many components, some of them more subtle than others. For example, in an imperfect schedule, a pilot may have to fly as a passenger on some flight. This results in lost revenue not only because a passenger seat is taken up but also because the pilot has to be paid even when riding as a passenger. How does one make an operating plan for an airline that minimizes the total cost while meeting all the constraints? A problem of this type is called a *combinatorial optimization* problem, since there are only a finite number of combinations possible, and in principle, one can enumerate all of them, eliminate the ones that do not meet the conditions and among those that do, select the one that has the least operating cost. Needless to say, one needs to be more clever than simple enumeration, due to the vast number of combinations involved.

As another example, consider a communication network consisting of switches interconnected by trunks (e.g. terrestrial, oceanic, satellite) in a particular topology. A telephone call originating in one switch can take many different paths (of switches) to terminate in another switch, using up trunks along the path. The problem is to design a minimum cost network that can carry the expected traffic. After a network is designed and implemented, operating the network involves various other combinatorial optimization problems, e.g. dynamic routing of calls.

As a third example, consider inductive inference, a central problem in artificial intelligence and machine learning. Inductive inference is the process of hypothesizing a general rule from examples. Inductive inference involves the following steps: (i) Inferring rules from examples, finding compact abstract models of data or hidden patterns in the data; (ii) Making predictions based on abstractions; (iii) Learning, i.e. modifying the abstraction based on comparing predictions with actual results; (iv) Designing questions to generate new examples. Consider the first step of the above process, i.e. discovering patterns in data. For example, given the sequence $2, 4, 6, 8, \ldots$, we may ask, "What comes next?" One could pick any number and justify it by fitting a fourth degree polynomial through the 5 points. However, the answer "10" is considered the most "intelligent." That is so because it is based on the first-order polynomial $2n$, which is linear and hence *simpler* than a fourth degree polynomial. The answer to an inductive inference problem is not unique. In inductive inference, one wants a *simple* explanation that fits a given set of observations. Simpler answers are considered better answers. One therefore needs a way to measure simplicity. For example, in finite automaton inference, the number of states could be a measure of simplicity. In logic circuit inference, the measure could be the number of gates and wires. Inductive inference, in fact leads to a discrete optimization problem, where one wants to maximize simplicity, or find a model, or set of rules, no more complex than some specified measure, consistent with already known data.

As a further example, consider the linear ordering problem, an important problem in economics, the social sciences, and also archaeology. In this problem, we are given several objects that we wish to place in order. There is a cost associated with placing object i before object j and a cost for placing object j before object i. The objective is to order the objects to minimize the total cost. There are methods for ranking sports teams that can be formulated as linear ordering problems: if team A beats team B then team A should go ahead of team B in the ranking, but it may be that team B beat team C, who in turn beat team A, so the determination of the "best" ordering is a non-trivial task, usually depending on the margin of victory.

Even though the four examples given above come from four different facets of life and look superficially to be quite different, they all have a common mathematical structure and can be described in a common mathematical notation called *integer programming*. In integer programming, the unknowns are represented by variables that take on a finite or discrete set of values. The various constraints or conditions on the problem are captured by algebraic expressions of these variables. For example, in the airline crew assignment problem discussed above, let us denote by variable x_{ij} the decision quantity that assigns crew i to flight j. Let there be m crew and n flights. If the variable x_{ij} takes on a value 1, then we say that crew i is assigned to flight j, and the cost of that assignment is c_{ij}. If the value is 0, then crew i is *not* assigned to flight j. Thus, the total crew-scheduling cost for the airline is given by the expression

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

that must be minimized. The condition that every flight should have exactly one crew is expressed by the equations

$$\sum_{i=1}^{m} x_{ij} = 1, \text{ for every flight } j = 1,\ldots,n. \tag{2}$$

We should also stipulate that the variables should take on only values 0 or 1. This condition is denoted by the notation

$$x_{ij} \in \{0,1\},\ 1 \leq i \leq m;\ 1 \leq j \leq n. \tag{3}$$

Other conditions on the crew can be expressed in a similar fashion. Thus, an integer programming formulation of the airline crew assignment problem is to minimize the operating cost given by (1) subject to various conditions given by other algebraic equations and inequalities. The formulations of the network design problem, the inductive inference problem, as well as the linear operdering problem, look mathematically similar to the above problem.

Linear programming is a special and simpler type of combinatorial optimization problem in which the integrality constraints of the type (3) are absent and we are given a linear objective function to be minimized subject to linear inequalities and equalities. A standard form of linear program is stated as follows:

$$\min_{x\in \mathsf{R}^n} \{c^\top x | Ax \leq b;\ l \leq x \leq u\}, \tag{4}$$

where $c, u, l, x \in \mathsf{R}^n$, $b \in \mathsf{R}^m$ and $A \in \mathsf{R}^{m\times n}$. In (4), x is the vector of decision variables, $Ax \leq b$ and $l \leq x \leq u$ represent constraints on the decision variables, and $c^\top x$ is the linear objective function to be minimized. Figure 1 shows a geometric interpretation of a linear program on the Euclidean plane. Each

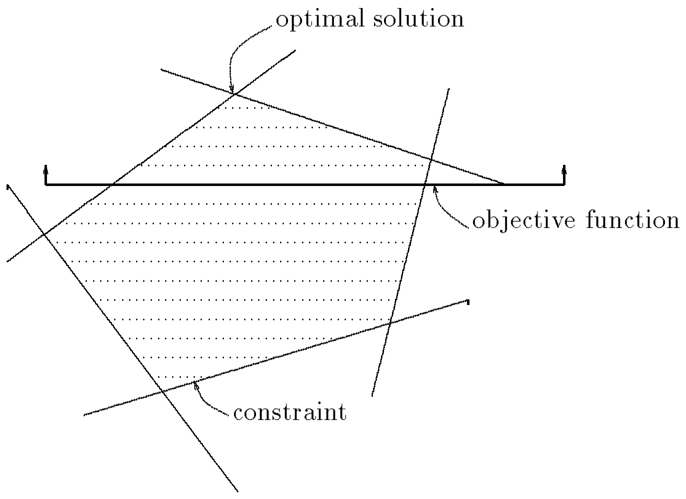

Figure 1: Geometric view of linear programming

linear inequality is represented by a line that partitions the plane into two *half-spaces*. Each inequality requires that for a solution to be feasible, it must lie in one of the half-spaces. The feasible region is the intersection of the half-spaces and is represented in the figure by the hashed area. The objective function, that must be minimized over the feasible region, is represented by a sliding line. This line intersects the feasible region in a set of points, all having the same objective value. As the line is swept across the feasible region in the direction of improvement, its objective value decreases. The set of points determined by the intersection of the sliding line with the feasible region that attains the best objective function value is called the *optimal* solution. In the example of the figure there is a unique optimal solution. In fact, a fundamental theorem in linear programming states that the optimal solution of a linear program occurs at a vertex of the polytope defined by the constraints of the linear program. This result gives linear programming its combinatorial nature. Even though the linear programming decision variables are continuous in nature, this result states that only a discrete and finite number of points in the solution space need to be examined.

Linear programming has a wide range of applications, including personnel assignment, production planning and distribution, refinery planning, target assignment, medical imaging, control systems, circuit simulation, weather forecasting, signal processing and financial engineering. Many polynomial-time solvable combinatorial problems are special cases of linear programming (e.g. matching and maximum flow). Linear programming has also been the source of many theoretical developments, in fields as diverse such as economics and queueing theory.

Combinatorial problems occur in diverse areas. These include graph theory (e.g. graph partitioning, network flows, graph coloring), linear inequalities (e.g. linear and integer programming), number theory (e.g. factoring, primality testing, discrete logarithm), group theory (e.g. graph isomorphism, group intersection), lattice theory (e.g. basis reduction), and logic and artificial intelligence (e.g. satisfiability, inductive and deductive inference boolean function minimization). All these problems, when abstracted mathematically have a commonality of discreteness. The solution approaches for solving these problems also have a great deal in common. In fact, attempts to come up with solution techniques revealed more commonality of the problems than was revealed from just the problem formulation. The solution of combinatorial problems has been the subject of much research. There is a continuously evolving body of knowledge, both theoretical and practical, for solving these problems.

2.2 Scope and computational efficiency

We illustrate with some examples the broad scope of applications of the interior-point techniques and their computational effectiveness. Since the most widely applied combinatorial optimization problem is linear programming, we begin with this problem. Each step of the interior-point method as applied to linear programming involves the solution of a linear system of equations. While a straightforward implementation of solving these linear systems can still outperform the Simplex Method, more sophisticated implementations have achieved orders of magnitude improvement over the Simplex Method. These advanced implementations make use of techniques from many disciplines such as linear algebra, numerical analysis, computer architecture, advanced data structures, and differential geometry. Tables 1–2 show the performance comparison between implementations of the Simplex (CPLEX) and interior-point (ADP [79]) methods on a class of linear programming relaxations of the quadratic assignment problems [127, 122]. Similar relative performances have been observed in problems drawn from disciplines such as operations research, electrical engineering, computer science, and statistics [79]. As the table shows, the relative superiority of interior-point method over the Simplex Method grows as the problem size grows and the speed-up factor can exceed 1000. Larger problems in the table could only be solved by the interior-point method because of impracticality of running the Simplex Method. In fact, the main practical contribution of the interior-point method has been to enable the solution of many large-scale real-life problems in fields such as telecommunication, transportation and defense, that could not be solved earlier by the Simplex Method.

From the point of view of efficient implementation, interior-point methods have another important property: they can exploit parallelism rather well [60, 131]. A parallel architecture based on multi-dimensional finite projective ge-

Table 1: LP relaxations of QAP integer programming formulation

	LP relaxation		simplex		int. pt.		
name	rows	vars	itr	time	itr	time	time ratio
nug05	210	225	103	0.2s	14	1.6s	0.1
nug06	372	486	551	2.3s	17	2.6s	0.9
nug07	602	931	2813	22.0s	19	6.2s	3.5
nug08	912	1632	5960	91.3s	18	9.5s	9.6
nug12	3192	8856	57524	9959.1s	29	754.1s	13.2
nug15	6330	22275	239918	192895.2s	36	5203.8s	37.1
nug20	15240	72600	est. time: $>$ 2 months		31	6745.5s	-
nug30	52260	379350	did not run		36	35058.0s	-

Table 2: CPLEX 3.0 and ADP runs on selected QAPLIB instances

	LP relaxation		primal simplex		dual simplex		ADP	
prob	rows	vars	itr	time	itr	time	itr	time
nug05	1410	825	265	1.7s	370	1.1s	48	3.2s
nug06	3972	2886	7222	604.3s	1872	22.2s	55	12.2s
nug07	9422	8281	39830	47970.3s	6057	720.3s	59	43.3s
nug08	19728	20448	did not run		16034	37577.1s	63	139.1s
nug12	177432	299256	did not run		did not run		91	6504.2s

ometries, particularly well suited for interior-point methods, has been proposed [75].

We now illustrate computational experience with an interior point based heuristic for integer programming [74, 77]. Here again, the main computational task at each iteration, is the solution of one or more systems of linear equations. These systems have a structure similar to the system solved in each iteration of interior point algorithms for linear programming and therefore software developed for linear programming can be reused in integer programming implementations.

Consider, as a first example, the Satisfiability (SAT) Problem in propositional calculus, a central problem in mathematical logic. During the last decade, a variety of heuristics have been proposed for this problem [61, 30]. A Boolean variable x can assume only values 0 or 1. Boolean variables can be combined by the logical connectives **or** ($\vee$), **and** ($\wedge$) and **not** ($\bar{x}$) to form Boolean formulae (e.g. $x_1 \wedge \bar{x}_2 \vee x_3$). A variable or a single negation of the variable is called a *literal.* A Boolean formula consisting of only literals combined by the $\vee$ operator is called a *clause.* SAT can be stated as follows: Given m clauses $\mathcal{C}_1, \ldots, \mathcal{C}_m$ involving n variables $x_1, \ldots, x_n$, does the formula $\mathcal{C}_1 \wedge \cdots \wedge \mathcal{C}_m$ evaluate to 1 for some Boolean input vector $[x_1, \ldots, x_n]$? If so, the formula is said to be satisfiable. Otherwise it is unsatisfiable.

Table 3: SAT: Comparison of Simplex and interior point methods

SAT Problem Size			Speed
Variables	Clauses ($\|\mathcal{C}\|$)	Avg Lits/Clause	Up
50	100	5	5
100	200	5	22
200	400	7	66
400	800	10	319

SAT can be formulated as the integer programming feasibility problem

$$\sum_{j \in \mathcal{I}_C} x_j - \sum_{j \in \mathcal{J}_C} x_j \geq 1 - |\mathcal{J}_c|, \ \mathcal{C} = \mathcal{C}_1, \ldots, \mathcal{C}_m, \tag{5}$$

where

$$\begin{aligned} \mathcal{I}_\mathcal{C} &= \{j \mid \text{literal } x_j \text{ appears in clause } \mathcal{C}\} \\ \mathcal{J}_\mathcal{C} &= \{j \mid \text{literal } \bar{x}_j \text{ appears in clause } \mathcal{C}\} \ . \end{aligned}$$

If an integer vector $x \in \{0,1\}^n$ is produced satisfying (5), the corresponding SAT problem is said to be satisfiable.

An interior point implementation was compared with an approach based on the Simplex Method to prove satisfiability of randomly generated instances of SAT [65]. Instances with up to 1000 variables and 32,000 clauses were solved. Compared with the Simplex Method approach on small problems (Table 3), speedups of over two orders of magnitude were observed. Furthermore, the interior point approach was successful in proving satisfiability in over 250 instances that the Simplex Method approach failed.

As a second example, consider inductive inference. The interior point approach was applied to a basic model of inductive inference [68]. In this model there is a black box (Figure 2) with n Boolean input variables $x_1, \ldots, x_n$ and a single Boolean output variable y. The black box contains a hidden Boolean function $\mathcal{F} : \{0,1\}^n \rightarrow \{0,1\}$ that maps inputs to outputs. Given a limited number of inputs and corresponding outputs, we ask: Does there exist an algebraic sum-of-products expression with no more than K product terms that matches this behavior? If so, what is it? It turns out that this problem can be formulated as a SAT problem.

Consider the hidden logic described by the 32-input, 1-output Boolean expression $y = x_4 x_{11} x_{15} \bar{x}_{22} + x_2 x_{12} \bar{x}_{15} \bar{x}_{29} + \bar{x}_3 x_9 x_{20} + \bar{x}_{10} x_{11} \bar{x}_{29} x_{32}$. This function has $2^{32} \simeq 4.3 \times 10^9$ distinct input-output combinations. Table 4 summarizes the computational results for this instance, where subsets of input-output examples of size 50, 100 and 400 were considered and the number of terms in the expression to be synthesized was fixed at $K = 4$. In all instances, the interior

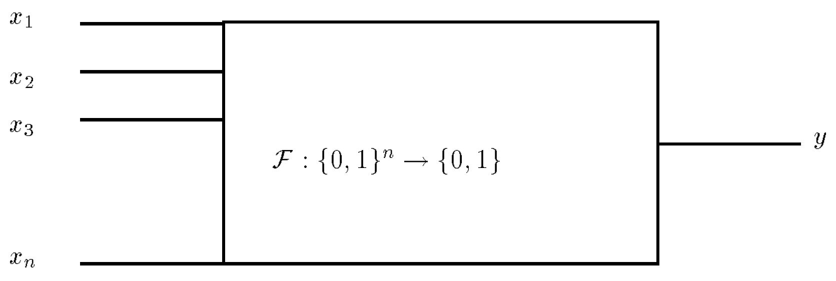

Figure 2: Black box with hidden logic

Table 4: Inductive inference SAT problems: 32-variable hidden logic

I/O Samples	SAT Size: Vars	SAT Size: $\lvert\mathcal{C}\rvert$	itr	CPU time	Inferred Logic	Prediction Accuracy
50	332	2703	49	66s	$y = \bar{x}_{22} x_{28} \bar{x}_{29} + x_{12} \bar{x}_{17} \bar{x}_{25} x_{27} + \bar{x}_3 x_9 x_{20} + x_{11} x_{12} \bar{x}_{16} \bar{x}_{32}$	.74
100	404	5153	78	178s	$y = x_9 x_{11} \bar{x}_{22} \bar{x}_{29} + x_4 x_{11} \bar{x}_{22} + \bar{x}_3 x_9 x_{20} + x_{12} \bar{x}_{15} \bar{x}_{16} \bar{x}_{29}$	.91
400	824	19478	147	1227s	$y = x_4 x_{11} \bar{x}_{22} + \bar{x}_{10} x_{11} \bar{x}_{29} x_{32} + \bar{x}_3 x_9 x_{20} + x_2 x_{12} \bar{x}_{15} \bar{x}_{29}$	exact

Table 5: Efficiency on inductive inference problems: interior point and combinatorial approaches

Variables	SAT Problem		Interior Method		Combinatorial
Hidden Logic	vars	$\|\mathcal{C}\|$	itr	time	Method (time)
8	396	2798	1	9.33s	43.05s
8	930	6547	13	45.72s	11.78s
8	1068	8214	33	122.62s	9.48s
16	532	7825	89	375.83s	20449.20s
16	924	13803	98	520.60s	*
16	1602	23281	78	607.80s	*
32	228	1374	1	5.02s	159.68s
32	249	2182	1	9.38s	176.32s
32	267	2746	1	9.76s	144.40s
32	450	9380	71	390.22s	*
32	759	20862	1	154.62s	*

$*$ Did not find satisfiable assignment in 43200s.

point algorithm synthesized a function that described completely the behavior of the sample. With a sample of only 400 input-output patterns the approach succeeded in exactly describing the hidden logic. The prediction accuracy given in the table was computed with Monte Carlo simulation, where 10,000 random vectors were input to the black box and to the inferred logic and their outputs compared. Table 5 illustrates the efficiency of the interior-point method compared to the combinatorial Davis-Putnam Method [24].

As another example of an application of the continuous approach to combinatorial problems, consider the wire routing problem for gate arrays, an important subproblem arising in VLSI design. As shown in Figure 3, a gate array can be abstracted mathematically as a grid graph. Input to the wire routing problem consists of a list of wires specified by end points on a rectangular grid. Each edge of the graph, also known as a channel, has a prespecified capacity representing the maximum number of wires it can carry. The combinatorial problem is to find a wiring pattern without exceeding capacity of horizontal and vertical channels. This problem can be formulated as an integer programming problem. The interior-point approach has successfully obtained provably optimal global solutions to large-scale problems of this type having more than 20,000 wires [109]. On the other hand, combinatorial heuristics, such as simulated annealing are not comparable either in the quality of the solution they can find or in terms of the computational cost.

A further example of the successful application of interior point methods to solve combinatorial optimization problems comes from statistical physics. The problem of finding the ground state of an Ising spin glass is related to the magnetism of materials. Finding the ground state can be modelled as the

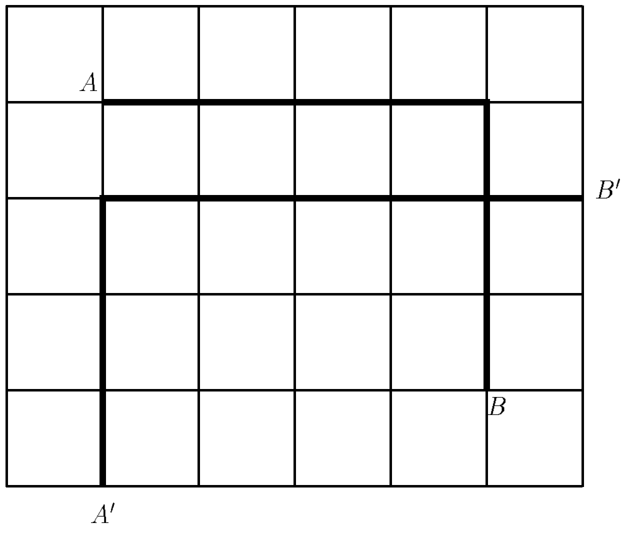

Figure 3: Wire routing

problem of finding the maximum cut in a graph whose vertices and edges are those of a grid on a torus. It can be formulated as an integer programming problem and solved using a cutting plane approach. If the weights on the edges are ± 1 then the linear programs suffer from degeneracy, which limits the size of problems that can be solved efficiently using the Simplex Method. The use of an interior point algorithm to solve the relaxations allows the solution of far larger problems. For example, solving problems on a 70×70 toroidal grid using simplex required up to a day on a Sun SPARCstation 10 [26], whereas problems on a 100×100 grid could be solved in an average of about 3 hours 20 minutes on a Sun SPARC 20/71 when using an interior point code [98].

In the case of many other combinatorial problems, numerous heuristic approaches have been developed. Many times, the heuristic merely encodes the prior knowledge or anticipation of the structure of solution to a specific class of practical applications into the working of the algorithm. This may make a limited improvement in efficiency without really coming to grips with the problem of exponential growth that plagues the combinatorial approach. Besides, one needs to develop a wide variety of heuristics to deal with different situations. Interior-point methods have provided a unified approach to create efficient algorithms for many different combinatorial problems.

3 Solution techniques

Solution techniques for combinatorial problems can be classified into two groups: combinatorial and continuous approaches. In this section, we contrast these approaches.

3.1 Combinatorial approach

The combinatorial approach creates a sequence of states drawn from a discrete and finite set. Each state represents a suboptimal solution or a partial solution to the original problem. It may be a graph, a vertex of a polytope, a collection of subsets of a finite set or some other combinatorial object. At each major step of the algorithm, the next state is chosen in an attempt to improve the current state. The improvement may be in the quality of the solution measured in terms of the objective function, or it may be in making the partial solution more feasible. In any case, the improvement is guided by *local* search. By local search we mean that the solution procedure only examines a neighboring set of configurations and greedily selects one that improves the current solution. Thus, local search is quite myopic, with no consideration given to evaluate whether this move may make any sense globally. Indeed, a combinatorial approach often lacks the information needed for making such an evaluation. In many cases, the greedy local improvement may trap the solution in a local minimum that is qualitatively much worse than a true global minimum. To escape from a local minimum, the combinatorial approach needs to resort to techniques such as backtracking or abandoning the sequences of states created so far altogether and restarting with a different initial state. Most combinatorial problems suffer from the property of having a large number of local minima when the search space is confined to a discrete set. For a majority of combinatorial optimization problems, the phenomenon of multiple local minima may create a problem for the combinatorial approach.

On the other hand, for a limited class of problems, one can rule out the possibility of local minima and show that local improvement also leads to global improvement. For many problems in this class, polynomial-time algorithms (i.e. algorithms whose running time can be proven to be bounded from above by polynomial functions of the lengths of the problems) have been known for a long time. Examples of problems in this class are bipartite matching and network flows. It turns out that many of these problems are special cases of linear programming, which is also a polynomial-time problem. However, the Simplex Method, which employs a combinatorial approach to solving linear programs, has been shown to be an exponential-time algorithm. In contrast, all polynomial-time algorithms for solving the general linear programming problem employ a continuous approach. These algorithms use either the Ellipsoid Method [81] or one of the variants of the Karmarkar Method.

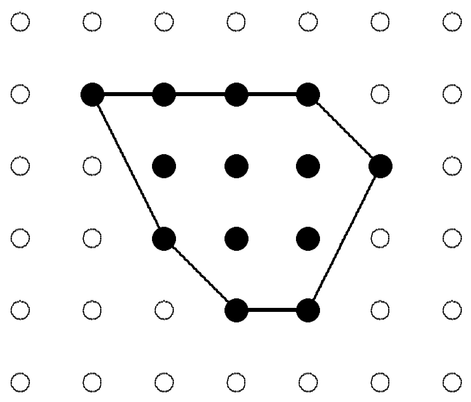

Figure 4: Discrete set embedded into continuous set

3.2 Continuous approach

In the continuous approach to solving discrete problems, the set of candidate solutions to a given combinatorial problem is embedded in a larger continuous space. The topological and geometric properties of the continuous space play an essential role in the construction of the algorithm as well as in the analysis of its efficiency. The algorithm involves the creation of a sequence of points in the enlarged space that converges to the solution of the original combinatorial problem. At each major step of the algorithm, the next point in the sequence is obtained from the current point by making a good *global* approximation to the entire set of relevant solutions and solving it. Usually it is also possible to associate a continuous trajectory or a set of trajectories with the limiting case of the discrete algorithm obtained by taking infinitesimal steps. Topological properties of the underlying continuous space such as connectivity of the level sets of the function being optimized are used for bounding the number of local minima and choosing an effective formulation of the continuous optimization problem. The geometric properties such as distance, volume and curvature of trajectories are used for analyzing the rate of convergence of an algorithm, whereas the topological properties help determine if a proposed algorithm will converge at all. We now elaborate further on each of the main concepts involved in the continuous approach.

3.2.1 Examples of embedding

Suppose the candidate solutions to a discrete problem are represented as points in the n-dimensional real space R^n. This solution set can be embedded into a larger continuous space by forming the convex hull of these points (Figure 4).

This is the most common form of continuous embedding and is used for solving linear and integer programming problems. As another example, consider a discrete problem whose candidate solution set is a finite cyclic group. This can be embedded in a continuous Lie group $\{e^{i\theta} | 0 \leq \theta < 2\pi\}$ [20]. A Lie group embedding is useful for the problem of graph isomorphism or automorphism. In this problem, let A denote the adjacency matrix of the graph. Then the discrete solution set of the automorphism problem is the permutation group given by $\{P | AP = PA;\ P \text{ is a permutation matrix }\}$. This can be embedded in a larger continuous group given by $\{U | AU = UA;\ U \text{ is a complex unitary matrix}\}$.

3.2.2 Global approximation

At each major step of the algorithm, a subproblem is solved to obtain the next point in the sequence. The subproblem should satisfy two properties: (i) it should be a *global* approximation to the original problem; and (ii) should be efficiently solvable. In the context of linear programming, the Karmarkar Method contains a way of making a global approximation having both of the above desirable properties and is based on the following theorem.

Theorem 3.1 *Given any polytope P and an interior point $\mathbf{x} \in P$, there exists a projective transformation T, that transforms P to P' and $\mathbf{x}$ to $\mathbf{x}' \in P'$ so that it is possible to find in the transformed space a circumscribing ball $B(\mathbf{x}', R) \supseteq P'$, of radius R, center $\mathbf{x}'$, containing P' and a inscribing ball $B(\mathbf{x}', r) \subseteq P'$ of radius r, center $\mathbf{x}'$ contained in P' such that the ratio R/r is at most n.*

The inverse image (under T) of the inscribed ball is used as the optimization space for the subproblem and satisfies the two properties stated above, leading to a polynomial-time algorithm for linear programming. The effectiveness of this global approximation is also borne out in practice since Karmarkar's Method and its variants take very few approximation steps to find the global optimum of the original problem. Extension of this global approximation step to integer programming have led to new algorithms for solving NP-complete problems, as we shall see later.

3.2.3 Continuous trajectories

Suppose an interior-point method produces an iteration of the following type,

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha \mathbf{f}^{(k)} + \mathcal{O}(\alpha^2), \tag{6}$$

where $\mathbf{x}^{(k)}$ is the k-th iterate, $\mathbf{f}^{(k)}$ is the k-th direction of improvement, and α is the step-length parameter. Then by taking the limit as $\alpha \rightarrow 0$, we get the *infinitesimal version* of the algorithm whose continuous trajectories are given by the differential equation

$$\frac{d\mathbf{x}}{d\alpha} = \mathbf{f}(\mathbf{x}), \tag{7}$$

where $\mathbf{f}(\mathbf{x})$ defines a vector field. Thus, the infinitesimal version of the algorithm can be thought of as a nonlinear dynamical system. For the projective method for linear programming, the differential equation is given by

$$\frac{d\mathbf{x}}{dt} = -[D - \mathbf{x}\mathbf{x}^{\top}]P_{AD} \cdot D\mathbf{c},$$

where

$$P_{AD} = I - DA^{\top}(AD^2A^{\top})^{-1}AD,$$
$$D = \mathrm{diag}\{x_1, x_2, \cdots, x_n\}.$$

Similarly, continuous trajectories and the corresponding differential equations can be derived for other interior-point methods. These trajectories have a rich mathematical structure in them. Many times they also have algebraic descriptions and alternative interpretations. The continuous trajectory given above for the linear programming problem converges to an optimal solution of the problem corresponding to the objective function vector $\mathbf{c}$. Note that the vector field depends on $\mathbf{c}$ in a smooth way and as the vector $\mathbf{c}$ is varied one can get to each vertex of the polytope as limit of some continuous trajectory. If one were to attempt a direct combinatorial description of the discrete solution set of a linear programming problem, it would become enormously complex since the number of solutions can be exponential with respect to the size of the problem. In contrast, the simple differential equation given above implicitly encodes the complex structure of the solution set. Another important fact to be noticed is that the differential equation is written in terms of the *original* input matrix A defining the problem. Viewing combinatorial objects as limiting cases of continuous objects often makes them more accessible to mathematical reasoning and also permits construction of more efficient algorithms.

The power of the language of differential equations in describing complex phenomena is rather well-known in the natural sciences. For example, if one were to attempt a direct description of the trajectories involved in planetary motion, it would be enormously complex. However, a small set of differential equations written in terms of the *original* parameters of the problem is able to describe the same motion. One of the most important accomplishments of Newtonian mechanics was finding a *simple* description of the apparently complex phenomena of planetary motion, in the form of differential equations.

In the context of combinatorial optimization, the structure of the solution set of a discrete problem is often rather complex. As a result, a straightforward combinatorial approach to solving these problems has not succeeded in many cases and has led to a belief that these problems are intractable. Even for linear programming, which is one of the simplest combinatorial optimization problems, the best known method, in both theory and practice, is based on the continuous approach rather than the combinatorial approach. Underlying this continuous approach is a small set of differential equations, capable of encoding the complicated combinatorial structure of the solution set. As this approach is extended and generalized, one hopes to find new and efficient algorithms for many other combinatorial problems.

3.2.4 Topological properties

There are many ways to formulate a given discrete problem as a continuous optimization problem, and it is rather easy to make a formulation that would be difficult to solve even by means of continuous trajectories. How does one make a formulation that is solvable? The most well-known class of continuous solvable problems is the class of convex minimization problems. This leads to a natural question: Is convexity the characteristic property that separates the class of efficiently solvable minimization problems from the rest? To explore this question we need to look at topological properties. Topology is the study of properties invariant under any continuous, one-to-one transformation of space having a continuous inverse.

Suppose we have a continuous optimization problem that is solvable by means of continuous trajectories. It may be a convex problem, for example. Suppose we apply a nonlinear transformation to the space that is a diffeomorphism. The transformed problem need not be convex, but it will continue to be solvable by means of continuous trajectories. In fact, the image of the continuous trajectories in the original space, obtained by applying the *same* diffeomorphism gives us a way of solving the transformed problem. Conversely, if the original problem was unsolvable, it could not be converted into a solvable problem by any such transformation. Hence any diffeomorphism maps solvable problems onto solvable problems and unsolvable problems onto unsolvable problems. This argument suggests that the property characterizing the solvable class may be a topological property and not simply a geometric property such as convexity.

The simplest topological property relevant to the performance of interior-point methods is connectivity of the level sets of the function being optimized. Intuitively, a subset of continuous space is connected if any two points of the subset can be joined by a continuous path lying entirely in the subset. In the context of function minimization, the significance of connectivity lies in the fact that functions having connected level sets do not have spurious local minima. In other words every local minimum is necessarily a global minimum. A continuous formulation of NP-complete problems having such desirable topological properties is given in [74]. The approach described there provides a theoretical foundation for constructing efficient algorithms for discrete problems on the basis of a common principle. Algorithms for many practical problems can now be developed which differ mainly in the way the combinatorial structure of he problem is exploited to gain additional computational efficiency.

4 Interior point methods for linear and network programming

4.1 Linear programming

Interior point methods were first described by Dikin [29] in the mid 1960s, and current interest in them started with Karmarkar's algorithm in the mid 1980s [72]. As the name suggests, these algorithms generate a sequence of iterates which moves through the relative interior of the feasible region, in marked contrast to the simplex method [22], where each iterate is an extreme point. Like the ellipsoid method [81], many interior point methods have polynomial complexity, whereas every known variant of the simplex method can take an exponential number of iterations in the worst case. Computationally, interior point methods usually require far less time than their worst-case bounds, and they appear to be superior to the simplex method, at least for problems with a large number of constraints and variables (say, more than one thousand). Recent books discussing interior point methods for linear programming include [132, 140, 146, 148].

The dual affine scaling method is similar to Dikin's original method and is discussed in section 4.3.1. In this section, we consider a slightly more complicated interior point method, namely the primal-dual predictor-corrector method PDPCM [92, 90]. This is perhaps the most popular and widely implemented interior point method. The basic idea with an interior point method is to enable the method to take long steps, by choosing directions that do not immediately run into the boundary. With the PDPCM this is achieved by considering a modification of the original problem, with a penalty term for approaching the boundary. Thus, for the standard form linear programming problem

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

where c and x are n-vectors, b is an m-vector, and A is dimensioned appropriately, the barrier function subproblem is constructed:

$$\begin{array}{lrcl} \min & \multicolumn{3}{l}{c^T x - \mu \sum_{i=1}^{n} \log(x_i)} \\ \text{subject to} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

Here, log denotes the natural logarithm, and μ is a positive constant. Note that if x_i approaches zero for any component, then the objective function value approaches ∞.

If the original linear program has a compact set of optimal solutions then each barrier subproblem will have a unique optimal solution. The set of all these optimal solutions is known as the *central trajectory*. The limit point of the central trajectory as μ tends to zero is an optimal point for the original linear program. If the original linear program has more than one optimal solution, then the limit point is in the relative interior of the optimal face.

Fiacco and McCormick [34] suggested following the central trajectory to the optimal solution. This requires solving a barrier subproblem for a particular choice of μ, decreasing μ, and repeating. The hope is that knowing the solution to one subproblem will make it easy to solve the next one. It also suffices to only solve the subproblems approximately, both theoretically and practically. Monteiro and Adler [102] showed that if μ is decreased by a sufficiently small amount then an approximate solution to one subproblem can be used to obtain an approximate solution to the next one in just one iteration, leading to an algorithm that requires $O(n^{1/2})$ iterations. With a more aggressive reduction in μ (for example, μ is halved at each iteration), more iterations are required to obtain an approximate solution to the new subproblem, and the best complexity result that has been proved for such algorithms is that they require $O(n)$ iterations.

There are several issues that need to be resolved in order to specify the algorithm, including the choice of values of μ, methods for solving the subproblems, and the desired accuracy of the solutions of the subproblems. Many different choices have been investigated, at least theoretically. Much of the successful computational work has focussed on the PDPCM, and so we now describe that algorithm.

The optimality conditions for the subproblems can be written:

$$\begin{aligned} Ax &= b && (8) \\ A^T y + z &= c && (9) \\ XZe &= \mu e && (10) \end{aligned}$$

where e denotes the vector of all ones of appropriate dimension, y is an m-vector, z is a nonnegative n-vector, and X and Z are $n \times n$ diagonal matrices with $X_{ii} = x_i$ and $Z_{ii} = z_i$ for $i = 1, \ldots, n$. Equation (9) together with the nonnegativity restriction on z corresponds to dual feasibility. Notice that the last condition is equivalent to saying that $x_i z_i = \mu$ for each component i. Complementary slackness for the original linear program would require $x_i z_i = 0$ for each component i. The duality gap is $x^T z$, so if a point satisfies the optimality conditions (8–10) then the duality gap will be $n\mu$.

We assume we have a strictly positive feasible solution x and a dual feasible solution (y, z) satisfying $A^T y + z = c$, $z > 0$. If these assumptions are not satisfied, the algorithm can be modified appropriately; see, for example, [90] or Zhang [150].

An iteration of PDPCM consists of three parts:

- A Newton step to move towards the solution of the linear program is calculated (but not taken). This is known as the *predictor step*.

- This predictor step is used to update μ.
- A *corrector step* is taken, which combines the decrease of the predictor step with a step that tries to stay close to the central trajectory.

The predictor step gives an immediate return on the value of the objective function, and the corrector step brings the iterate back towards the central trajectory, making it easier to obtain a good decrease on future steps.

The calculation of the predictor step requires solving the following system of equations to obtain the search directions $\Delta^p x$, $\Delta^p y$, and $\Delta^p z$:

$$\begin{aligned} A\Delta^p x &= 0 & (11)\\ A^T\Delta^p y + \Delta^p z &= 0 & (12)\\ Z\Delta^p x + X\Delta^p z &= -XZe. & (13) \end{aligned}$$

One method to solve this system is to notice that it requires

$$AZ^{-1}XA^T\Delta^p y = -AXe. \qquad (14)$$

The Cholesky factorization of the matrix $AZ^{-1}XA^T$ can be calculated, and this can be used to obtain $\Delta^p y$. The vectors $\Delta^p z$ and $\Delta^p x$ can then be calculated from the equations (12) and (13).

It can be shown that if we choose a step of length α then the duality is reduced by a factor of α, so if we took a step of length one then the duality gap would be reduced to zero. However, it is not usually possible to take such a large step and still remain feasible. Thus, we calculate α_P^p and α_D^p, the maximum possible step lengths to maintain primal and dual feasibility.

We use these steplengths to aid in the adaptive update of μ. If the steplengths are close to one, then the duality gap can be decreased dramatically, and the new μ should be considerably smaller than the old value. Conversely, if the steplengths are short then the iterates are close to the boundary, so μ should only be decreased slightly and the iterates should be pushed back towards the central trajectory. This leads to one possible update of μ as

$$\mu^+ = (g_p/x^Tz)^2 g_p/n \qquad (15)$$

where g_p is the duality gap that would result if primal and dual steps of lengths α_P^p and α_D^p, respectively, were taken.

A corrector step $(\Delta x, \Delta y, \Delta z)$ is used to bring the iterates back towards the central trajectory. This involves solving the system of equations

$$\begin{aligned} A\Delta x &= 0 & (16)\\ A^T\Delta y + \Delta z &= 0 & (17)\\ Z\Delta x + X\Delta z &= \mu^+ e - XZe - v^p & (18) \end{aligned}$$

where v^p is an n-vector, with components $v_i^p = \Delta^p x_i \Delta^p z_i$. This system can be solved using the Cholesky factors of the matrix $AZ^{-1}XA^T$, which were already formed when calculating the predictor step.

Once the direction has been calculated, the primal and dual step lengths α_P and α_D are chosen to ensure that the next iterate has $x^+ > 0$ and $z^+ > 0$. The iterates are updated as:

$$
\begin{aligned}
x^+ &= x + \alpha_P \Delta x &(19)\\
y^+ &= y + \alpha_D \Delta y &(20)\\
z^+ &= z + \alpha_D \Delta z. &(21)
\end{aligned}
$$

Typically, α_P and α_D are chosen to move the iterates as much as 99.95% of the way to the boundary.

The predictor-corrector method [103, 92] can be thought of as finding a direction by using a second order approximation to the central trajectory. A second order solution to (8–10) would require that the direction satisfy

$$Z\Delta x + X\Delta z + v = \mu^+ e - XZe$$

where v is a vector to be determined with $v_i = \Delta x_i \Delta z_i$. It is not possible to easily solve this equation together with (16–17), so it is approximated by the system of equations (16–18).

The method makes just one iteration for each reduction in μ, but typically μ is decreased very quickly, by perhaps at least a constant factor at each iteration. The duality gap usually close to $n\mu$, so the algorithm is terminated when the duality gap drops below some tolerance. The algorithm typically takes only 40 or so iterations even for problems with thousands of constraints and/or variables.

The computational work in an iteration is dominated by the factorization of the matrix $AZ^{-1}XA^T$. The first step in the factorization is usually to permute the rows of A to reduce the number of nonzeroes in the Cholesky factors — this step need only be performed once in the algorithm. Once the ordering is set, a numerical factorization of the matrix is performed at each iteration. These factors are then used to calculate the directions by means of backward and forward substitutions. The use of the correction direction was found to decrease the number of iterations required to solve a linear program by enough to justify the extra work at each iteration of calculating an extra direction. Higher order approximations [13, 14, 3, 76] have proven useful for some problems where the cost of the factorization is far greater than the cost of backward and forward substitutions — see [148]. For a discussion of the computational issues involed in implementing an interior point algorithm, see, for example, Adler et al. [3] and Andersen *et al.* [7].

4.2 Network programming

A large number of problems in transportation, communications, and manufacturing can be modeled as network flow problems. In these problems one seeks to find the most efficient, or optimal, way to move flow (e.g. materials, information, buses, electrical currents) on a network (e.g. postal network, computer network, transportation grid, power grid). Among these optimization problems, many are special classes of linear programming problems, with combinatorial properties that enable development of efficient solution techniques. In this section, we limit our discussion to these linear network flow problems. For a treatment of classes of nonlinear network flow problems, the reader is referred to [31, 50, 51, 114] and references therein.

Given a directed graph $G = (\mathcal{N}, \mathcal{A})$, where $\mathcal{N}$ is a set of m nodes and $\mathcal{A}$ a set of n arcs, let (i, j) denote a directed arc from node i to node j. Every node is classified in one of the following three categories. *Source* nodes produce more flow than they consume. *Sink* nodes consume more flow than they produce. *Transshipment* nodes produce as much flow as they consume. Without loss of generality, one can assume that the total flow produced in the network equals the total flow consumed. Each arc has associated with it an origination node and a destination node, implying a direction for flow to follow. Arcs have limitations (often called capacities or bounds) on how much flow can move through them. The flow on arc (i, j) must be no less than l_{ij} and can be no greater than u_{ij}. To set up the problem in the framework of an optimization problem, a unit flow cost c_{ij}, incurred by each unit of flow moving through arc (i, j), must be defined. Besides being restricted by lower and upper bounds at each arc, flows must satisfy another important condition, known as Kirchhoff's Law (conservation of flow), which states that for every node in the network, the sum of all incoming flow together with the flow produced at the node must equal the sum of all outgoing flow and the flow consumed at the node. The objective of the *minimum cost network flow problem* is to determine the flow on each arc of the network, such that all of the flow produced in the network is moved from the source nodes to the sink nodes in the most cost-effective way, while not violating Kirchhoff's Law and flow limitations on the arcs. The minimum cost network flow problem can be formulated as the following linear program:

$$\min \sum_{(i,j)\in\mathcal{A}} c_{ij} x_{ij} \tag{22}$$

subject to:

$$\sum_{(j,k)\in\mathcal{A}} x_{jk} - \sum_{(k,j)\in\mathcal{A}} x_{kj} = b_j, \quad j \in \mathcal{N} \tag{23}$$

$$l_{ij} \le x_{ij} \le u_{ij}, \quad (i,j) \in \mathcal{A}. \tag{24}$$

In this formulation, x_{ij} denotes the flow on arc (i, j) and c_{ij} is the cost of transporting one unit of flow on arc (i, j). For each node $j \in \mathcal{N}$, let b_j denote a quantity associated with node j that indicates how much flow is produced or consumed at the node. If $b_j > 0$, node j is a source. If $b_j < 0$, node j is a sink. Otherwise ($b_j = 0$), node j is a transshipment node. For each arc $(i, j) \in \mathcal{A}$, as before, let l_{ij} and u_{ij} denote, respectively, the lower and upper bounds on flow on arc (i,j). The case where $u_{ij} = \infty$, for all $(i,j) \in \mathcal{A}$, gives rise to the *uncapacitated* network flow problem. Without loss of generality, l_{ij} can be set to zero. Most often, the problem data (i.e. c_{ij}, u_{ij}, l_{ij}, for $(i,j) \in \mathcal{A}$ and b_j, for $j \in \mathcal{N}$) are assumed to be integer, and many codes adopt this assumption. However, there can exist applications where the data are real numbers, and algorithms should be capable of handling problems with real data.

Constraints of type (23) are referred to as the flow conservation equations, while constraints of type (24) are called the flow capacity constraints. In matrix notation, the above network flow problem can be formulated as a linear program of the special form

$$\min \{c^\top x \mid Ax = b,\ l \le x \le u\},$$

where A is the $m \times n$ *node-arc incidence matrix* of the graph $G = (\mathcal{N}, \mathcal{A})$, i.e. for each arc (i,j) in $\mathcal{A}$ there is an associated column in matrix A with exactly two nonzero entries: an entry 1 in row i and an entry -1 in row j. Note that from the mn entries of A, only $2n$ are nonzero and because of this, the node-arc incidence matrix is not a space-efficient representation of the network. There are many other ways to represent a network. A popular representation is the *node-node adjacency* matrix B. This is an $m \times m$ matrix with an entry 1 in position (i,j) if arc $(i,j) \in \mathcal{A}$ and 0 otherwise. Such a representation is efficient for dense networks, but is inefficient for sparse networks. A more efficient representation of sparse networks is the *adjacency list*, where for each node $i \in \mathcal{N}$ there exists a list of arcs emanating from node i, i.e. a list of nodes j such that $(i,j) \in \mathcal{A}$. The *forward star* representation is a multi-array implementation of the adjacency list data structure. The adjacency list enables easy access to the arcs emanating from a given node, but not the incoming arcs. The *reverse star* representation enables easy access to the list of arcs incoming into i. Another representation that is much used in interior point network flow implementations is a simple *arc list*, where the arcs are stored in a linear array. The complexity of an algorithm for solving network flow problems depends greatly on the network representation and the data structures used for maintaining and updating the intermediate computations.

We denote the i-th column of A by A_i, the i-th row of A by $A_{.i}$ and a submatrix of A formed by columns with indices in set S by A_S. If graph G is disconnected and has p connected components, there are exactly p redundant flow conservation constraints, which are sometimes removed from the problem formulation. We rule out a trivially infeasible problem by assuming

$$\sum_{j \in \mathcal{N}^k} b_j = 0, \quad k = 1, \ldots, p, \tag{25}$$

where $\mathcal{N}^k$ is the set of nodes for the k-th component of G.

Often it is further required that the flow x_{ij} be integer, i.e. we replace (24) with

$$l_{ij} \leq x_{ij} \leq u_{ij}, \ x_{ij} \text{ integer}, \ (i,j) \in \mathcal{A}. \tag{26}$$

Since the node-arc incidence matrix A is totally unimodular, when the data is integer all vertex solutions of the linear program are integer. An algorithm that finds a vertex solution, such as the simplex method, will necessarily produce an integer optimal flow. In certain types of network flow problems, such as the assignment problem, one may be only interested in solutions having integer flows, since fractional flows do not have a logical interpretation.

In the remainder of this section we assume, without loss of generality, that $l_{ij} = 0$ for all $(i,j) \in \mathcal{A}$ and that $c \neq 0$. A simple change of variables can transform the original problem into an equivalent one with $l_{ij} = 0$ for all $(i,j) \in \mathcal{A}$. The case where $c = 0$ is a simple feasibility problem, and can be handled by solving a maximum flow problem [4].

Many important combinatorial optimization problems are special cases of the minimum cost network flow problem. Such problems include the linear assignment and transportation problems, and the maximum flow and shortest path problems. In the transportation problem, the underlying graph is bipartite, i.e. there exist two sets $\mathcal{S}$ and $\mathcal{T}$ such that $\mathcal{S} \cup \mathcal{T} = \mathcal{N}$ and $\mathcal{S} \cap \mathcal{T} = \emptyset$ and arcs occur only from nodes of $\mathcal{S}$ to nodes of $\mathcal{T}$. Set $\mathcal{S}$ is usually called the set of source nodes and set $\mathcal{T}$ is the set of sink nodes. For the transportation problem, the right hand side vector in (23) is given by

$$b_j = \begin{cases} s_j & \text{if } j \in \mathcal{S} \\ -t_j & \text{if } j \in \mathcal{T}, \end{cases}$$

where s_j is the supply at node $j \in \mathcal{S}$ and t_j is the demand at node $j \in \mathcal{T}$. The assignment problem is a special case of the transportation problem, in which $s_j = 1$ for all $j \in \mathcal{S}$ and $t_j = 1$ for all $j \in \mathcal{T}$.

The computation of the maximum flow from node s to node t in $G = (\mathcal{N}, \mathcal{A})$ can be done by computing a minimum cost flow in $G' = (\mathcal{N}', \mathcal{A}')$, where $\mathcal{N}' = \mathcal{N}$ and $\mathcal{A}' = \mathcal{A} \cup (t,s)$, where

$$c_{ij} = \begin{cases} 0 & \text{if } (i,j) \in \mathcal{A} \\ -1 & \text{if } (i,j) = (t,s), \end{cases}$$

and

$$u_{ij} = \begin{cases} \text{cap}(i,j) & \text{if } (i,j) \in \mathcal{A} \\ \infty & \text{if } (i,j) = (t,s), \end{cases}$$

where $\text{cap}(i,j)$ is the capacity of arc (i,j) in the maximum flow problem.

The shortest paths from node s to all nodes in $\mathcal{N} \setminus \{s\}$ can be computed by solving an uncapacitated minimum cost network flow problem in which c_{ij} is the length of arc (i,j) and the right hand side vector in (23) is given by

$$b_j = \begin{cases} m-1 & \text{if } j = s \\ -1 & \text{if } j \in \mathcal{N} \setminus \{s\}. \end{cases}$$

Although all of the above combinatorial optimization problems are formulated as minimum cost network flow problems, several specialized algorithms have been devised for solving them efficiently.

In many practical applications, flows in networks with more than one commodity need to be optimized. In the multicommodity network flow problem, k commodities are to be moved in the network. The set of commodities is denoted by $\mathcal{K}$. Let x^k_{ij} denote the flow of commodity k in arc (i,j). The multicommodity network flow problem can be formulated as the following linear program:

$$\min \sum_{k\in\mathcal{K}} \sum_{(i,j)\in\mathcal{A}} c^k_{ij} x^k_{ij} \tag{27}$$

subject to:

$$\sum_{(j,l)\in\mathcal{A}} x^k_{jl} - \sum_{(l,j)\in\mathcal{A}} x^k_{lj} = b^k_j, \quad j\in\mathcal{N}, \quad k\in\mathcal{K} \tag{28}$$

$$\sum_{k\in\mathcal{K}} x^k_{ij} \le u_{ij}, \quad (i,j)\in\mathcal{A}, \tag{29}$$

$$x^k_{ij} \ge 0, \quad (i,j)\in\mathcal{A}, \quad k\in\mathcal{K}. \tag{30}$$

The minimum cost network flow problem is a special case of the multicommodity network flow problem, in which there is only one commodity.

In the 1940s, Hitchcock [58] proposed an algorithm for solving the transportation problem and later Dantzig [23] developed the Simplex Method for linear programming problems. In the 1950s, Kruskal [83] developed a minimum spanning tree algorithm and Prim [118] devised an algorithm for the shortest path problem. During that decade, commercial digital computers were introduced widely. The first book on network flows was published by Ford and Fulkerson [37] in 1962. Since then, active research produced a variety of algorithms, data structures, and software for solving network flow problems. For an introduction to network flow problems and applications, see the books [4, 15, 31, 37, 80, 84, 133, 137].

4.3 Components of interior point network flow methods

Since Karmarkar's breakthrough in 1984, many variants of his algorithm, including the dual affine scaling, with and without centering, reduced dual affine scaling, primal path following (method of centers), primal-dual path following, predictor-corrector primal-dual path following, and the infeasible primal-dual path following, have been used to solve network flow problems. Though these algorithms are, in some sense, different, they share many of the same computational requirements. The key ingredients for efficient implementation of these algorithms are:

1. The solution of the linear system $ADA^\top u = t$, where D is a diagonal $n \times n$ scaling matrix, and u and t are m-vectors. This requires an iterative algorithm for computing approximate directions, preconditioners, stopping criteria for the iterative algorithm, etc.

2. The recovery of the desired optimal solution. This may depend on how the problem is presented (integer data or real data), and what type of solution is required (fractional or integer solution, ϵ-optimal or exact solution, primal optimal or primal-dual optimal solution, etc.).

In this subsection, we present in detail these components, illustrating their implementation in the dual affine scaling network flow algorithm DLNET of Resende and Veiga [130].

4.3.1 The dual affine scaling algorithm

The dual affine scaling (DAS) algorithm [12, 29, 135, 141] was one of the first interior point methods to be shown to be competive computationally with the simplex method [2, 3]. As before, let A be an $m \times n$ matrix, c, u, and x be n-dimensional vectors and b an m-dimensional vector. The DAS algorithm solves the linear program

$$\min \{c^\top x \mid Ax = b,\ 0 \leq x \leq u\}$$

indirectly, by solving its dual

$$\max \{b^\top y - u^\top z \mid A^\top y - z + s = c,\ z \geq 0, s \geq 0\}, \tag{31}$$

where z and s are an n-dimensional vectors and y is an m-dimensional vector. The algorithm starts with an initial interior solution $\{y^0, z^0, s^0\}$ such that

$$A^\top y^0 - z^0 + s^0 = c,\ z^0 > 0,\ s^0 > 0,$$

and iterates according to

$$\{y^{k+1}, z^{k+1}, s^{k+1}\} = \{y^k, z^k, s^k\} + \alpha \{\Delta y, \Delta z, \Delta s\},$$

where the search directions $\Delta y, \Delta z$, and Δs satisfy

$$\begin{aligned} A(Z_k^2 + S_k^2)^{-1} A^\top \Delta y &= b - AZ_k^2(Z_k^2 + S_k^2)^{-1} u, \\ \Delta z &= Z_k^2(Z_k^2 + S_k^2)^{-1}(A^\top \Delta y - S_k^2 u), \\ \Delta s &= \Delta z - A^\top \Delta y, \end{aligned}$$

where

$$Z_k = \operatorname{diag}(z_1^k, \ldots, z_n^k) \text{ and } S_k = \operatorname{diag}(s_1^k, \ldots, s_n^k)$$

and α is such that $z^{k+1} > 0$ and $s^{k+1} > 0$, i.e. $\alpha = \gamma \times \min\{\alpha_z, \alpha_s\}$, where $0 < \gamma < 1$ and

$$\alpha_z = \min\{-z_i^k/(\Delta z)_i \mid (\Delta z)_i < 0,\ i = 1, \ldots, n\}$$

$$\alpha_s = \min\{-s_i^k/(\Delta s)_i \mid (\Delta s)_i < 0,\ i = 1,\dots,n\}.$$

The dual problem (31) has a readily available initial interior point solution:

$$\begin{aligned} y_i^0 &= 0,\ i = 1,\dots,n \\ s_i^0 &= c_i + \lambda,\ i = 1,\dots,n \\ z_i^0 &= \lambda,\ i = 1,\dots,n, \end{aligned}$$

where λ is a scalar such that $\lambda > 0$ and $\lambda > -c_i,\ i = 1,\dots,n$. The algorithm described above has two important parameters, γ and λ. For example, in DLNET, $\gamma = 0.95$ and $\lambda = 2\,\|c\|_2$.

4.3.2 Computing the direction

The computational efficiency of interior point network flow methods relies heavily on a preconditioned conjugate gradient algorithm to solve the direction finding system at each iteration. The preconditioned conjugate gradient algorithm is used to solve

$$M^{-1}(AD_kA^\top)\Delta y = M^{-1}\bar{b} \tag{32}$$

where M is a positive definite matrix and, in the case of the DAS algorithm, $\bar{b} = b - AZ_k^2D_ku$, and $D_k = (Z_k^2 + S_k^2)^{-1}$ is a diagonal matrix of positive elements. The objective is to make the preconditioned matrix

$$M^{-1}(AD_kA^\top) \tag{33}$$

less ill-conditioned than $AD_kA^\top$, and improve the convergence of the conjugate gradient algorithm.

The pseudo-code for the preconditioned conjugate gradient algorithm is presented in Figure 5. The computationally intensive steps in the preconditioned conjugate gradient algorithm are lines 3, 7 and 11 of the pseudo-code. These lines correspond to a matrix-vector multiplication (7) and solving linear systems of equations (3 and 11). Line 3 is computed once and lines 7 and 11 are computed once every conjugate gradient iteration. The matrix-vector multiplications are of the form $AD_kA^\top\ p_i$, carried out without forming $AD_kA^\top$ explicitly. One way to compute the above matrix-vector multiplication is to decompose it into three sparse matrix-vector multiplications. Let

$$\zeta' = A^\top p_i \quad \text{and} \quad \zeta'' = D_k\zeta'.$$

Then

$$(A\ (D_k\ (A^\top p_i))) = A\zeta''.$$

The complexity of this matrix-vector multiplication is $O(n)$, involving n additions, $2n$ subtractions and n floating point multiplications.

```
procedure pcg($A, D_k, \bar{b}, \epsilon_{cg}, \Delta y$)
1   $\Delta y_0 := 0$;
2   $r_0 := \bar{b}$;
3   $z_0 := M^{-1} r_0$;
4   $p_0 := z_0$;
5   $i := 0$;
6   do stopping criterion not satisfied $\rightarrow$
7       $q_i := A D_k A^\top p_i$;
8       $\alpha_i := z_i^\top r_i / p_i^\top q_i$;
9       $\Delta y_{i+1} := \Delta y_i + \alpha_i p_i$;
10      $r_{i+1} := r_i - \alpha_i q_i$;
11      $z_{i+1} := M^{-1} r_{i+1}$;
12      $\beta_i := z_{i+1}^\top r_{i+1} / z_i^\top r_i$;
13      $p_{i+1} := z_{i+1} + \beta_i p_i$;
14      $i := i + 1$
15  od;
16  $\Delta y := \Delta y_i$
end pcg;
```

Figure 5: The preconditioned conjugate gradient algorithm

The preconditioned residual is computed in lines 3 and 11 when the system of linear equations

$$M z_{i+1} = r_{i+1}, \tag{34}$$

is solved, where M is a positive definite matrix. An efficient implementation requires a preconditioner that can make (34) easy to solve. On the other hand, one needs a preconditioner that makes (33) well conditioned. In the next subsection, we show several preconditioners that satisfy, to some extent, these two criteria.

To determine when the approximate direction Δy_i produced by the conjugate gradient algorithm is satisfactory, one can compute the angle θ between $(AD_kA^\top)\Delta y_i$ and $\bar{b}$ and stop when $|1 - \cos\theta| < \epsilon_{cos}$, where ϵ_{cos} is some small tolerance. In practice, one can initially use $\epsilon_{cos} = 10^{-3}$ and tighten the tolerance as the interior point iterations proceed, as $\epsilon_{cos} = \epsilon_{cos} \times 0.95$. The exact computation of

$$\cos\theta = \frac{|\bar{b}^\top (AD_kA^\top)\Delta y_i|}{\|\bar{b}\|_2 \cdot \|(AD_kA^\top)\Delta y_i\|_2}$$

has the complexity of one conjugate gradient iteration and is therefore expensive if computed at each conjugate gradient iteration. One way to proceed is to compute the cosine every l_{cos} conjugate gradient iterations. A more efficient procedure [116] follows from the observation that $(AD_kA^\top)\Delta y_i$ is approximately equal to $\bar{b} - r_i$, where r_i is the estimate of the residual at the i-th conjugate gradient iteration. Using this approximation, the cosine can be estimated by

$$\cos\theta = \frac{|\bar{b}^\top(\bar{b} - r_i)|}{\|\bar{b}\|_2 \cdot \|(\bar{b} - r_i)\|_2}.$$

Since, in practice, the conjugate gradient method finds good directions in few iterations, this estimate has been shown to be effective and can be computed at each conjugate gradient iteration.

4.3.3 Network preconditioners for conjugate gradient method

A useful preconditioner for the conjugate gradient algorithm must be one that allows the efficient solution of (34), while at the same time causing the number of conjugate gradient iterations to be small. Five preconditioners have been found useful in conjugate gradient based interior point network flow methods: diagonal, maximum weighted spanning tree, incomplete QR decomposition, the Karmarkar-Ramakrishnan preconditioner for general linear programming, and the approximate Cholesky decomposition preconditioner [93] .

A diagonal matrix constitutes the most straightforward preconditioner used in conjunction with the conjugate gradient algorithm [45]. They are simple to compute, taking $O(n)$ double precision operations, and can be effective [129, 131, 149]. In the diagonal preconditioner, $M = \text{diag}\ (AD_kA^\top)$, and the preconditioned residue systems of lines 3 and 11 of the conjugate gradient pseudo-code in Figure 5 can each be solved in $O(m)$ double precision divisions.

A preconditioner that is observed to improve with the number of interior point iterations is the maximum weighted spanning tree preconditioner. Since the underlying graph G is not necessarily connected, one can identify a maximal forest using as weights the diagonal elements of the current scaling matrix,

$$w = D_k e, \tag{35}$$

where e is a unit n-vector. In practice, Kruskal's and Prim's algorithm have been used to compute the maximal forest. Kruskal's algorithm, implemented with the data structures in [137] has been applied to arcs, ordered approximately with a bucket sort [62, 130], or exactly using a hybrid QuickSort [64]. Prim's algorithm is implemented in [116] using the data structures presented in [4].

At the k-th interior point iteration, let $\mathcal{S}_k$ be the submatrix of A with columns corresponding to arcs in the maximal forest, $t_1, \ldots, t_q$. The preconditioner can be written as

$$M = \mathcal{S}_k \mathcal{D}_k \mathcal{S}_k^\top,$$

where, for example, in the DAS algorithm

$$\mathcal{D}_k = \text{diag}(1/z_{t_1}^2 + 1/s_{t_1}^2, \ldots, 1/z_{t_q}^2 + 1/s_{t_q}^2).$$

For simplicity of notation, we include in $\mathcal{S}_k$ the linear dependent rows corresponding to the redundant flow conservation constraints. At each conjugate gradient iteration, the preconditioned residue system

$$(\mathcal{S}_k \mathcal{D}_k \mathcal{S}_k^\top) z_{i+1} = r_{i+1} \tag{36}$$

is solved with the variables corresponding to redundant constraints set to zero. As with the diagonal preconditioner, (36) can be solved in $O(m)$ time, as the system coefficient matrix can be ordered into a block triangular form.

Portugal et al. [116] introduced a preconditioner based on an incomplete QR decomposition (IQRD) for use in interior point methods to solve transportation problems. They showed empirically, for that class of problems, that this preconditioner mimics the diagonal preconditioner during the initial iterations of the interior point method, and the spanning tree preconditioner in the final interior point method iterations, while causing the conjugate gradient method to take fewer iterations than either method during the intermediate iterations. In [117], the use of this preconditioner is extended to general minimum cost network flow problems. In the following discussion, we omit the iteration index k from notation for the sake of simplicity. Let $\bar{\mathcal{T}} = \{1, 2, \ldots, n\} \setminus \mathcal{T}$ be the index set of the arcs not in the computed maximal spanning tree, and let

$$D = \begin{bmatrix} D_{\mathcal{T}} & \\ & D_{\bar{\mathcal{T}}} \end{bmatrix},$$

where $D_{\mathcal{T}} \in \mathrm{R}^{q \times q}$ is the diagonal matrix with the arc weights of the maximal spanning tree and $D_{\bar{\mathcal{T}}} \in \mathrm{R}^{(n-q)\times(n-q)}$ is the diagonal matrix with weights of the arcs not in the maximal spanning tree. Then

$$\begin{aligned} ADA^\top &= \begin{bmatrix} A_{\mathcal{T}} & A_{\bar{\mathcal{T}}} \end{bmatrix} \begin{bmatrix} D_{\mathcal{T}} & \\ & D_{\bar{\mathcal{T}}} \end{bmatrix} \begin{bmatrix} A_{\mathcal{T}}^\top \\ A_{\bar{\mathcal{T}}}^\top \end{bmatrix} \\ &= \begin{bmatrix} A_{\mathcal{T}} D_{\mathcal{T}}^{\frac{1}{2}} & A_{\bar{\mathcal{T}}} D_{\bar{\mathcal{T}}}^{\frac{1}{2}} \end{bmatrix} \begin{bmatrix} D_{\mathcal{T}}^{\frac{1}{2}} A_{\mathcal{T}}^\top \\ D_{\bar{\mathcal{T}}}^{\frac{1}{2}} A_{\bar{\mathcal{T}}}^\top \end{bmatrix}. \end{aligned}$$

The Cholesky factorization of $ADA^\top$ can be found by simply computing the QR factorization of

$$\bar{A} = \begin{bmatrix} D_{\mathcal{T}}^{\frac{1}{2}} A_{\mathcal{T}}^\top \\ D_{\bar{\mathcal{T}}}^{\frac{1}{2}} A_{\bar{\mathcal{T}}}^\top \end{bmatrix}. \tag{37}$$

In fact, if $Q\bar{A} = R$, then

$$ADA^\top = \bar{A}^\top \bar{A} = R^\top Q^\top Q R = R^\top R.$$

```
procedure iqrd($\mathcal{T}, \bar{\mathcal{T}}, \mathcal{N}, D, \mathcal{D}, F$)
    do $i \in \mathcal{N} \setminus \{r\} \rightarrow$
        $j = \rho_i$;
        if $(i, j) \in \mathcal{A}_{\mathcal{T}} \rightarrow \mathcal{D}_{ii} = D_{ij}$ fi;
        if $(j, i) \in \mathcal{A}_{\mathcal{T}} \rightarrow \mathcal{D}_{ii} = D_{ji}$ fi;
    od;
    do $(i, j) \in \mathcal{A}_{\bar{\mathcal{T}}} \rightarrow$
        if $i \in \mathcal{N} \setminus \{r\} \rightarrow \mathcal{D}_{ii} = \mathcal{D}_{ii} + D_{ij}$ fi;
        if $j \in \mathcal{N} \setminus \{r\} \rightarrow \mathcal{D}_{jj} = \mathcal{D}_{jj} + D_{ij}$ fi;
    od;
    do $i \in \mathcal{N} \setminus \{r\} \rightarrow$
        $j = \rho_i$;
        if $j \in \mathcal{N} \setminus \{r\} \rightarrow$
            if $(i, j) \in \mathcal{A}_{\mathcal{T}} \rightarrow F_{ij} = D_{ij} / \mathcal{D}_{ii}$ fi;
            if $(j, i) \in \mathcal{A}_{\mathcal{T}} \rightarrow F_{ji} = D_{ji} / \mathcal{D}_{ii}$ fi;
        fi;
    od;
end iqrd;
```

Figure 6: Computing the F and $\mathcal{D}$ matrices in IQRD

The computation of the QR factorization is not recommended here, since besides being more expensive than a Cholesky factorization, it also destroys the sparsity of the matrix $\bar{A}$. Instead, Portugal et al. [116] propose an *incomplete QR* decomposition of $\bar{A}$. Applying Givens rotations [39] to $\bar{A}$, using the diagonal elements of $D_{\mathcal{T}}^{\frac{1}{2}} A_{\mathcal{T}}^{\top}$, the elements of $D_{\bar{\mathcal{T}}}^{\frac{1}{2}} A_{\bar{\mathcal{T}}}^{\top}$ become null. No fill-in is incurred in this factorization. See [116] for an example illustrating this procedure. After the factorization, we have the preconditioner

$$M = F \mathcal{D} F^{\top},$$

where F is a matrix with a diagonal of ones that can be reordered to triangular form, and $\mathcal{D}$ is a diagonal matrix with positive elements.

To avoid square root operations, $\mathcal{D}$ and F are obtained without explicitly computing $\mathcal{D}^{\frac{1}{2}} F^{\top}$. Suppose that the maximum spanning tree is rooted at node r, corresponding to the flow conservation equation that has been removed from the formulation. Furthermore, let $\mathcal{A}_{\mathcal{T}}$ denote the subset of arcs belonging to the tree and let ρ_i represent the predecessor of node i in the tree. The procedure used to compute the nonzero elements of $\mathcal{D}$ and the off-diagonal nonzero elements of F is presented in the pseudo-code in Figure 6.

The computation of the preconditioned residual with $F \mathcal{D} F^{\top}$ requires $O(m)$ divisions, multiplications, and subtractions, since $\mathcal{D}$ is a diagonal matrix and F

can be permuted into a triangular matrix with diagonal elements equal to one. The construction of F and $\mathcal{D}$, that constitute the preconditioner, requires $O(n)$ additions and $O(m)$ divisions.

In practice, the diagonal preconditioner is effective during the initial iterations of the DAS algorithm. As the DAS iterations progress, the spanning tree preconditioner is more effective as it becomes a better approximation of matrix $AD_kA^\top$. Arguments as to why this preconditioner is effective are given in [62, 116]. The DLNET implementation begins with the diagonal preconditioner and monitors the number of iterations required by the conjugate gradient algorithm. When the conjugate gradient takes more than $\beta\sqrt{m}$ iterations, where $\beta > 0$, DLNET switches to the spanning tree preconditioner. Upper and lower limits to the number of DAS iterations using a diagonal preconditioned conjugate gradient are specified.

In [93], is proposed a Cholesky decomposition of an approximation of the matrix $A\Theta A^\top$ (CDAM) as preconditioner. This preconditioner has the form

$$M = LL^\top, \tag{38}$$

whith L the lower triangular Cholesky factor of the matrix

$$B\Theta_B B^\top + \rho \times \text{diag}(N\Theta_N N^\top), \tag{39}$$

where B and N are such that $A = [B \quad N]$ with B a basis matrix, Θ_B and Θ_N are the diagonal submatrices of Θ corresponding to B and N, respectively, and ρ is a parameter.

Another preconditioner used in an interior point implementation is the one for general linear programming, developed by Karmarkar and Ramakrishnan and used in [79, 120]. This preconditioner is based on a dynamic scheme to drop elements of the original scaled constraint matrix DA, as well as the from the factors of the matrix $ADA^\top$ of the linear system, and use the incomplete Cholesky factors as the preconditioner. Because of the way elements are dropped, this preconditioner mimics the diagonal preconditioner in the initial iterations and the tree preconditioner in the final iterations of the interior point algorithm.

4.3.4 Identifying the optimal partition

One way to stop an interior point algorithm before the (fractional) interior point iterates converge is to estimate (or guess) the optimal partition of arcs at each iteration, attempt to recover the flow from the partition and, if a feasible flow is produced, test if that flow is optimal. In the discussion that follows we describe a strategy to partition the set of arcs in the dual affine scaling algorithm. The discussion follows [128] closely, using a dual affine scaling method for uncapacitated networks to illustrate the procedure.

Let $A \in \mathsf{R}^{m\times n}$, $c, x, s \in \mathsf{R}^n$ and $b, y \in \mathsf{R}^m$. Consider the linear programming problem

$$\begin{array}{ll} \text{minimize} & c^\top x \\ \text{subject to} & Ax = b, \quad x \geq 0 \end{array} \tag{40}$$

and its dual

$$\begin{array}{ll} \text{maximize} & b^\top y \\ \text{subject to} & A^\top y + s = c, \quad s \geq 0. \end{array} \tag{41}$$

The dual affine scaling algorithm starts with an initial dual solution $y^0 \in \{y : s = c - A^\top y > 0\}$ and obtains iterate y^{k+1} from y^k according to $y^{k+1} = y^k + \alpha^k d_y^k$, where the search direction d_y is $d_y^k = (AD_k^{-2}A^\top)^{-1}b$ and $D_k = \mathrm{diag}(s_1^k, ..., s_n^k)$. A step moving a fraction γ of the way to the boundary of the feasible region is taken at each iteration, namely,

$$\alpha^k = \gamma \times \min\{-s_i^k/(d_s^k)_i \ : \ (d_s^k)_i < 0, \ i = 1, ..., n\}, \tag{42}$$

where $d_s^k = -A^\top d_y^k$ is a unit displacement vector in the space of slack variables. At each iteration, a tentative primal solution is computed by $x^k = D_k^{-2}A^\top(AD_k^{-2}A^\top)^{-1}b$. The set of optimal solutions is referred to as the *optimal face*. We use the index set N_* for the always-active index set on the optimal face of the primal, and B_* for its complement. It is well-known that B_* is the always-active index set on the optimal face of the dual, and N_* is its complement. An *indicator* is a quantity to detect whether an index belongs to N_* or B_*. We next describe three indicators that can be implemented in the DAS algorithm. For pointers to other indicators, see [33].

Under a very weak condition, the iterative sequence of the DAS algorithm converges to a relative interior point of a face on which the objective function is constant, i.e. the sequence $\{y^k\}$ converges to an interior point of a face on which the objective function is constant. Let B be the always-active index set on the face and N be its complement, and let b^∞ be the limiting objective function value. There exists a constant $C_0 > 0$ such that

$$\limsup_{k\to\infty} \frac{s_i^k}{b^\infty - b^\top y^k} \leq C_0 \tag{43}$$

for all $i \in B$, while

$$\frac{s_i^k}{b^\infty - b^\top y^k} \tag{44}$$

diverges to infinity for all $i \in N$. Denote by s^∞ the limiting slack vector. Then $s_N^\infty > 0$ and $s_B^\infty = 0$. The vector

$$u^k \equiv \frac{(D^k)^{-1}d_s^k}{b^\infty - b^\top y^k} = \frac{D^k x^k}{b^\infty - b^\top y^k} \tag{45}$$

plays an important role, since

$$\lim_{k\to\infty}(u^k)^\top e = \lim_{k\to\infty}\frac{(s^k)^\top x^k}{b^\infty - b^\top y^k} = 1. \tag{46}$$

Consequently, in the limit $b^\infty - b^\top y^k$ can be estimated by $(s^k)^\top x^k$ asymptotically, and (43) can be stated as

$$\lim_{k\to\infty}\sup\frac{s_i^k}{(s^k)^\top x^k} \leq C_0.$$

Then, if $i \in B$, for any β such that $0 < \beta < 1$,

$$\lim_{k\to\infty}\sup\frac{s_i^k}{((s^k)^\top x^k)^\beta} = 0,$$

since $((s^k)^\top x^k)^\beta$ converges to zero at a slower rate than $((s^k)^\top x^k)$ for any β such that $0 < \beta < 1$. Therefore, if $\beta = 1/2$, the following indicator has the property that $\lim_{k\to\infty} N^k = N_*$.

Indicator 1: Let $C_1 > 0$ be any constant, and define

$$N^k = \{i \in E \ : \ s_i^k \leq C_1\sqrt{(s^k)^\top x^k}\}. \tag{47}$$

This indicator is available under very weak assumptions, so it can be used to detect B_* and N_* without any substantial restriction on step-size. On the other hand, it gives the correct partition only if the limit point y^∞ happens to be a relative interior point of the optimal face of the dual and thus lacks a firm theoretical justification. However, since we know by experience that y^∞ usually lies in the relative interior of the optimal face, we may expect that it should work well in practice. Another potential problem with this indicator is that it is not scaling invariant, so that it will behave differently if the scaling of the problem is changed.

Now we assume that the step-size γ is asymptotically less than or equal to $2/3$. Then the limiting point exists in the interior of the optimal face and b^∞ is the optimal value. Specifically, $\{y^k\}$ converges to an interior point of the optimal face of the dual problem, $\{x^k\}$ converges to the analytic center of the optimal face of the primal problem, and $\{b^\top y^k\}$ converges linearly to the optimal value b^∞ asymptotically, where the (asymptotic) reduction rate is exactly $1-\gamma$. Furthermore, one can show that

$$\lim_{k\to\infty} u_i^k = 1/|B_*| \quad \text{for } i \in B_* \tag{48}$$

$$\lim_{k\to\infty} u_i^k = 0 \quad \text{otherwise.} \tag{49}$$

The vector u^k is not available because the exact optimal value is unknown a priori, but $b^\infty - b^\top y^k$ can be estimated by $(s^k)^\top x^k$ to obtain

$$\lim_{k\to\infty} \frac{s_i^k x_i^k}{(s^k)^\top x^k} = 1/|B_*| \quad \text{for } i \in B_* \tag{50}$$

$$\lim_{k\to\infty} \frac{s_i^k x_i^k}{(s^k)^\top x^k} = 0 \quad \text{otherwise.} \tag{51}$$

On the basis of this fact, the following procedure to construct N^k, which asymptotically coincides with N_*:

Indicator 2: Let δ be a constant between 0 and 1. We obtain N^k according to the following procedure:

- Step 1: Sort $g_i^k = s_i^k x_i^k/(s^k)^\top x^k$ according to its order of magnitude. Denote i_l the index for the l-th largest component.
- Step 2: For $p := 1, 2, \ldots$ compare g_{i_p} and δ/p, and let p^* be the first number such that $g_{i_{p^*}} \leq \delta/p^*$. Then set

$$N^k = \{i_1, i_2, \ldots, i_{p^*-1}\}. \tag{52}$$

To state the third, and most practical indicator, let us turn our attention to the asymptotic behavior of s_i^{k+1}/s_i^k. If $i \in N_*$, then s_i^k converges to a positive value, and hence

$$\lim_{k\to\infty} \frac{s_i^{k+1}}{s_i^k} = 1. \tag{53}$$

If $i \in B_*$, s_i^k converges to zero. Since

$$\lim_{k\to\infty} \frac{s_i^k x_i^k}{b^\infty - b^\top y^k} = \frac{1}{|B_*|}, \tag{54}$$

x_i^k converges to a positive number, and the objective function reduces with a rate of $1-\gamma$, then

$$\lim_{k\to\infty} \frac{s_i^{k+1}}{s_i^k} = 1 - \gamma, \tag{55}$$

which leads to the following indicator:

Indicator 3: Take a constant η such that $1 - \gamma < \eta < 1$. Then let

$$N^k = \{i \ : \ \frac{s_i^{k+1}}{s_i^k} \geq \eta\} \tag{56}$$

be defined as the index set. Then $N^k = N_*$ holds asymptotically.

Of the three indicators described here, Indicators 2 and 3 stand on the firmest theoretical basis. Furthermore, unlike Indicator 1, both are scaling invariant. The above discussion can be easily extended for the case of capacitated network flow problems. DLNET uses Indicator 3 to identify the set of active arcs defining the optimal face by examining the ratio between subsequent iterates of each dual slack. At the optimum, the flow on each arc can be classified as being at its upper bound, lower bound, or as active. From the discussion above, if the flow on arc i converges to its upper bound,

$$\lim_{k\to\infty} s_i^k/s_i^{k-1} = 1-\gamma \quad \text{and} \quad \lim_{k\to\infty} z_i^k/z_i^{k-1} = 1.$$

If the flow on arc i converges to its lower bound,

$$\lim_{k\to\infty} s_i^k/s_i^{k-1} = 1 \quad \text{and} \quad \lim_{k\to\infty} z_i^k/z_i^{k-1} = 1-\gamma.$$

If the flow on arc i is active,

$$\lim_{k\to\infty} s_i^k/s_i^{k-1} = 1-\gamma \quad \text{and} \quad \lim_{k\to\infty} z_i^k/z_i^{k-1} = 1-\gamma.$$

From a practical point of view, scale invariance is the most interesting feature of this indicator. An implementable version can use constants which depend only on the step size factor γ. Let $\kappa_0 = .7$ and $\kappa_1 = .9$. At each iteration of DLNET, the arcs are classified as follows:

- If $s_i^k/s_i^{k-1} < \kappa_0$ and $z_i^k/z_i^{k-1} > \kappa_1$, arc i is set to its upper bound.
- If $s_i^k/s_i^{k-1} > \kappa_1$ and $z_i^k/z_i^{k-1} < \kappa_0$, arc i is set to its lower bound.
- Otherwise, arc i is set active, defining the tentative optimal face.

4.3.5 Recovering the optimal flow

The simplex method restricts the sequence of solutions it generates to nodes of the linear programming polytope. Since the matrix A of the network linear program is totally unimodular, when a simplex variant is applied to a network flow problem with integer data, the optimal solution is also integer. On the other hand, an interior point algorithm generates a sequence of interior point (fractional) solutions. Unless the primal optimal solution is unique, the primal solution that an interior point algorithm converges to is not guaranteed to be integer. In an implementation of an interior point network flow method, one would like to be capable of recovering an integer flow even when the problem has multiple optima. We discuss below the stopping strategies implemented in DLNET and used to recover an integer optimal solution.

Besides the indicator described in subsection 4.3.4, DLNET uses the arcs of the spanning forest of the tree preconditioner as an indicator. If there exists a unique optimal flow, this indicator correctly identifies an optimal primal basic sequence, and an integer flow can be easily recovered by solving a triangular system of linear equations. In general, however, the arc indices do not converge to a basic sequence. Let $\mathcal{T} = \{t_1, \ldots, t_q\}$ denote the set of arc indices in the spanning forest. To obtain a tentative primal basic solution, first set flow on arcs not in the forest to either their upper or lower bound, i.e. for all $i \in \mathcal{A} \setminus \mathcal{T}$:

$$x_i^* = \begin{cases} 0 & \text{if } s_i^k > z_i^k \\ u_i & \text{otherwise,} \end{cases}$$

where s^k and z^k are the current iterates of the dual slack vectors as defined in (31). The remaining basic arcs have flows that satisfy the linear system

$$A_{\mathcal{T}} x_{\mathcal{T}}^* = b - \sum_{i \in \Omega^-} u_i A_i, \tag{57}$$

where $\Omega^- = \{i \in \mathcal{A} \setminus \mathcal{T} : s_i^k \leq z_i^k\}$. Because $A_{\mathcal{T}}$ can be reordered in a triangular form, (57) can be solved in $\mathcal{O}(m)$ operations. If $u_{\mathcal{T}} \geq x_{\mathcal{T}}^* \geq 0$ then the primal solution is feasible and optimality can be tested.

Optimality can be verified by producing a dual feasible solution (y^*, s^*, z^*) that is either complementary or that implies a duality gap less than 1. The first step to build a tentative optimal dual solution is identify the set of dual constraints defining the supporting affine space of the dual face complementary to x^*,

$$\mathcal{F} = \{i \in \mathcal{T} \ : \ 0 < x_i^* < u_i\},$$

i.e. the set of arcs with zero dual slacks. Since, in general, x^* is not feasible, $\mathcal{F}$ is usually determined by the indicators of subsection 4.3.4, as the index-set of active arcs. To ensure a complementary primal-dual pair, the current dual interior vector y^k is projected orthogonally onto this affine space. The solution y^* of the least squares problem

$$\min_{y^* \in \mathbf{R}^m} \{\|y^* - y^k\|_2 \ : \ A_{\mathcal{F}}^\top y^* = c_{\mathcal{F}}\} \tag{58}$$

is the projected dual iterate.

Let $G_{\mathcal{F}} = (\mathcal{N}, \mathcal{F})$ be the subgraph of G with $\mathcal{F}$ as its set of arcs. Since this subgraph is a forest, its incidence matrix, $A_{\mathcal{F}}$, can be reordered into a block triangular form, with each block corresponding to a tree in the forest. Assume $G_{\mathcal{F}}$ has p components, with $T_1, \ldots, T_p$ as the sets of arcs in each component tree. After reordering, the incidence matrix can be represented as

$$A_{\mathcal{F}} = \begin{bmatrix} A_{T_1} & & \\ & \ddots & \\ & & A_{T_p} \end{bmatrix}.$$

The supporting affine space of the dual face can be expressed as the sum of orthogonal one-dimensional subspaces. The operation in (58) can be performed by computing the orthogonal projections onto each individual subspace independently, and therefore can be completed in $\mathcal{O}(m)$ time. For $i = 1, \ldots, p$, denote the number of arcs in T_i by m_i, and the set of nodes spanned by those arcs by $\mathcal{N}_i$. A_{T_i} is an $(m_i + 1) \times m_i$ matrix and each subspace

$$\Psi_i = \{y_{\mathcal{N}_i} \in \mathsf{R}^{m_i+1} \ : \ A_{T_i}^\top y_{\mathcal{N}_i} = c_{T_i}\}$$

has dimension one. For all $y_{\mathcal{N}_i} \in \Psi_i$,

$$y_{\mathcal{N}_i} = y_{\mathcal{N}_i}^0 + \alpha_i y_{\mathcal{N}_i}^h, \tag{59}$$

where $y_{\mathcal{N}_i}^0$ is a given solution in Ψ_i and $y_{\mathcal{N}_i}^h$ is a solution of the homogeneous system $A_{T_i}^\top y_{\mathcal{N}_i} = 0$. Since A_{T_i} is the incidence matrix of a tree, the unit vector is a homogeneous solution. The given solution $y_{\mathcal{N}_i}^0$ can be computed by selecting $v \in \mathcal{N}_i$, setting $y_v^0 = 0$, removing the row corresponding to node v from matrix A_{T_i} and solving the resulting triangular system

$$\tilde{A}_{T_i}^\top y_{\mathcal{N}_i \setminus \{v\}} = c_{T_i}.$$

With the representation in (59), the orthogonal projection of $y_{\mathcal{N}_i}$ onto subspace Ψ_i is

$$y_{\mathcal{N}_i}^* = y_{\mathcal{N}_i}^0 + \frac{e_{\mathcal{N}_i}^\top (y_{\mathcal{N}_i} - y_{\mathcal{N}_i}^0)}{(m_i + 1)} e_{\mathcal{N}_i}$$

where e is the unit vector. The orthogonal projection, as indicated in (58), is obtained by combining the projections onto each subspace,

$$y^* = (y_{\mathcal{N}_1}^*, \ldots, y_{\mathcal{N}_q}^*).$$

A feasible dual solution is built by computing the slacks as

$$z_i^* = \begin{cases} -\delta_i & \text{if } \delta_i < 0 \\ 0 & \text{otherwise,} \end{cases} \qquad s_i^* = \begin{cases} 0 & \text{if } \delta_i < 0 \\ \delta_i & \text{otherwise,} \end{cases}$$

where $\delta_i = c_i - A^\top y^*$.

If the solution of (57) is feasible, optimality can be checked at this point, using the projected dual solution as a lower bound on the optimal flow. The primal and dual solutions, x^* and (y^*, s^*, z^*), are optimal if complementary slackness is satisfied, i.e. if for all $i \in \mathcal{A} \setminus \mathcal{T}$ either $s_i^* > 0$ and $x_i^* = 0$ or $z_i^* > 0$ and $x_i^* = u_i$. Otherwise, the primal solution, x^*, is still optimal if the duality gap is less than 1, i.e. if $c^\top x^* - b^\top y^* + u^\top z^* < 1$.

However, in general, the method proceeds attempting to find a feasible flow x^* that is complementary to the projected dual solution y^*. Based on the projected dual solution y^*, a refined tentative optimal face is selected by redefining the set of active arcs as

$$\tilde{\mathcal{F}} = \{i \in \mathcal{A} \ : \ |c_i - A_{\cdot i}^\top y^*| < \epsilon\}.$$

Next, the method attempts to build a primal feasible solution, x^*, complementary to the tentative dual optimal solution by setting the inactive arcs to lower or upper bounds, i.e., for $i \in \mathcal{A} \setminus \tilde{\mathcal{F}}$,

$$x_i^* = \begin{cases} 0 & \text{if } i \in \Omega^+ = \{i \in \mathcal{A} \setminus \tilde{\mathcal{F}} \ : \ c_i - A_{\cdot i}^\top y^* > 0\} \\ u_i & \text{if } i \in \Omega^- = \{i \in \mathcal{A} \setminus \tilde{\mathcal{F}} \ : \ c_i - A_{\cdot i}^\top y^* < 0\}. \end{cases}$$

By considering only the active arcs, a *restricted network* is built, represented by the constraint set

$$A_{\tilde{\mathcal{F}}} x_{\tilde{\mathcal{F}}} = \tilde{b} = b - \sum_{i \in \Omega^-} u_i A_i, \tag{60}$$

$$0 \le x_i \le u_i, \ \ i \in \tilde{\mathcal{F}}. \tag{61}$$

Clearly, from the flow balance constraints (60), if a feasible flow $x_{\tilde{\mathcal{F}}}^*$ for the restricted network exists, it defines, along with $x_{\Omega^+}^*$ and $x_{\Omega^-}^*$, a primal feasible solution complementary to y^*. A feasible flow for the restricted network can be determined by solving a maximum flow problem on the *augmented network* defined by underlying graph $\tilde{G} = (\tilde{\mathcal{N}}, \tilde{\mathcal{A}})$, where

$$\tilde{\mathcal{N}} = \{\sigma\} \cup \{\theta\} \cup \mathcal{N}$$

and

$$\tilde{\mathcal{A}} = \Sigma \cup \Theta \cup \tilde{\mathcal{F}}.$$

In addition, for each arc $(i,j) \in \tilde{\mathcal{F}}$ there is an associated capacity u_{ij}. The additional arcs are such that

$$\Sigma = \{(\sigma, i) \ : \ i \in \mathcal{N}^+\},$$

with associated capacity $\tilde{b}_i$ for each arc (σ, i), and

$$\Theta = \{(i, \theta) \ : \ i \in \mathcal{N}^-\},$$

with associated capacity $-\tilde{b}_i$ for each arc (i,θ), where $\mathcal{N}^+ = \{i \in \mathcal{N} \ : \ \tilde{b}_i > 0\}$ and $\mathcal{N}^- = \{i \in \mathcal{N} \ : \ \tilde{b}_i < 0\}$. It can be shown that if $\mathcal{M}_{\sigma,\theta}$ is the maximum flow value from σ to θ, and $\tilde{x}$ is a maximal flow on the augmented network, then $\mathcal{M}_{\sigma,\theta} = \sum_{i \in \mathcal{N}^+} \tilde{b}_i$ if and only if $\tilde{x}_{\tilde{\mathcal{F}}}$ is a feasible flow for the restricted network. Therefore, finding a feasible flow for the restricted network involves the solution a maximum flow problem. Furthermore, this feasible flow is integer, as we can select a maximum flow algorithm [4] that provides an integer solution.

5 Branch and bound methods

Branch and bound methods are exact algorithms for integer programming problems — given enough time, they are guaranteed to find an optimal solution. If there is not enough time available to solve a given problem exactly, a branch and bound algorithm can still be used to provide a bound on the optimal value. These methods can be used in conjunction with a heuristic algorithm such as local search, tabu search, simulated annealing, GRASP, genetic algorithms, or more specialized algorithms, to give a good solution to a problem, with a guarantee on the maximum possible improvement available over this good solution. Branch and bound algorithms work by solving relaxations of the integer programming problem, and selectively partitioning the feasible region to eventually find the optimal solution.

5.1 General concepts

Consider an integer programming problem of the form

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & \leq & b \\ & x & \geq & 0, \text{ integer,} \end{array}$$

where A is an $m \times n$ matrix, c and x are n-vectors, and b is an m-vector. The linear programming relaxation (*LP relaxation*) of this problem is

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{subject to} & Ax & \leq & b \\ & x & \geq & 0. \end{array}$$

If the optimal solution x^* to the LP relaxation is integral then it solves the integer programming problem also. Generally, the optimal solution to the LP relaxation will not be an integral point. In this case, the value of the LP relaxation provides a lower bound on the optimal value of the integer program, and we attempt to improve the relaxation.

In a branch and bound method, the relaxation is improved by dividing the relaxation into two subproblems, where one of the variables is restricted to take certain values. For example, if $x_i^* = 0.4$, we may set up one subproblem where x_i must be zero and another subproblem where x_i is restricted to take a value of at least one. We think of the subproblems as forming a tree, rooted at the initial relaxation.

If the solution to the relaxation of one of the subproblems in the tree is integral then it provides an upper bound on the optimal value of the complete integer program. If the solution to the relaxation of another subproblem has value larger than this upper bound, then that subproblem can be pruned, as no feasible solution for it can be optimal for the complete problem. If the relaxation of the subproblem is infeasible then the subproblem itself is infeasible and can be pruned. The only other possibility at a node of the tree is that the solution to the relaxation is fractional, with value less than that of the best known integral solution. In this case, we further subdivide the subproblem. There are many techniques available for choosing the branching variable and for choosing the next subproblem to examine; for more details, see, for example, Parker and Rardin [115].

Interior point methods are good for linear programming problems with a large number of variables, so they should also be useful for large integer programming problems. Unfortunately, large integer programming problems are often intractable for a general purpose method like branch and bound, because the tree becomes prohibitively large. Branch and bound interior point methods have proven successful for problems such as capacitated facility location problems [19, 25], where the integer variables correspond to the decision as to whether to build a facility at a particular location, and there are a large number of continuous variables corresponding to transporting goods from the facilities to customers. For these problems, the LP relaxations can be large even for instances with only a few integer variables.

As with interior point cutting plane methods (see section 6), the most important technique for making an interior point branch and bound method competitive is early termination. There are four possible outcomes at each node of the branch and bound tree; for three of these, it suffices to solve the relaxation approximately. The first outcome is that the relaxation has value greater than the known upper bound on the optimal value, so the node can be pruned by bounds. Usually, an interior point method will get close to the optimal value quickly, so the possibility of pruning by bounds can be detected early. The second possible outcome is that the relaxation is infeasible. Even if this is not detected quickly, we can usually iterate with a dual feasible algorithm (as with interior point cutting plane algorithms), so if the dual value becomes larger than the known bound we can prune. The third possible outcome is that the optimal solution to the relaxation is fractional. In this case, there are methods (including the Tapia indicator [33] and other indicators discussed in section 4.3.4) for detecting whether a variable is converging to a fractional value, and these can be used before optimality is reached. The final possible outcome is that the optimal solution to the relaxation is integral. In this situation, we can prune the node, perhaps resulting in an improvement in the value of the best known integral solution. Thus, we are able to prune in the only situation where it is necessary to solve the relaxation to optimality.

If the optimal solution to the relaxation is fractional, then the subproblem must be subdivided. The iterate for the parent problem will be dual feasible but primal infeasible for the child problems. The solution process can be restarted at these child problems either by using an infeasible interior point method or by using methods similar to those described for interior point cuting plane methods in section 6. For very large or degenerate problems, the interior point method has proven superior to simplex even when the interior point code is started from scratch at each node.

The first interior point branch and bound code was due to Borchers and Mitchell [19]. This method was adapted by De Silva and Abramson [25] specifically for facility location problems. Ramakrishnan *et al.* [119] have developed a branch and bound algorithm for the quadratic assignment problem. The linear programming relaxations at the nodes of the tree for this problem are so large that it was necessary to use an interior point method to solve them. Lee and Mitchell have been developing a parallel interior point branch and cut algorithm for mixed integer nonlinear programming problems [85].

5.2 An example: The QAP

The quadratic assignment problem (QAP) can be stated as

$$\min_{p \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{p(i)p(j)},$$

where Π is the set of all permutations of $\{1, 2, \ldots, n\}$, $A = (a_{ij}) \in \mathsf{R}^{n \times n}$, $B = (b_{ij}) \in \mathsf{R}^{n \times n}$.

Resende, Ramakrishnan, and Drezner [127] consider the following linear program as a lower bound (see also [1]) to the optimal solution of a QAP.

$$\min \sum_{i \in I}^{(i<j)} \sum_{r \in I}^{(r \neq s)} \sum_{j \in I} \sum_{s \in I} (a_{ij} b_{rs} + a_{ji} b_{sr}) y_{irjs}$$

subject to:

$$\sum_{j \in I}^{(j>i)} y_{irjs} + \sum_{j \in I}^{(j<i)} y_{jsir} = x_{ir}, \quad i \in I, r \in I, s \in I \ (s \neq r),$$

$$\sum_{s \in I}^{(r \neq s)} y_{irjs} = x_{ir}, \quad i \in I, r \in I, j \in I \ (j > i),$$

$$\sum_{s \in I}^{(r \neq s)} y_{jsir} = x_{ir}, \quad i \in I, r \in I, j \in I \ (j < i),$$

$$\sum_{i \in I} x_{ir} = 1, \quad r \in I,$$

$$\sum_{r \in I} x_{ir} = 1, \quad i \in I,$$

$$0 \leq x_{ir} \leq 1, \quad i \in I, r \in I,$$

$$0 \leq y_{irjs} \leq 1, \quad i \in I, r \in I, j \in I, s \in I,$$

Table 6: Dimension of lower bound linear programs

n	constraints	variables
2	12	6
3	42	27
4	104	88
5	210	225
6	372	486
7	602	931
8	912	1632
9	1314	2673
10	1820	4150
11	2442	6171
12	3192	8856
13	4082	12337
14	5124	16758

where the set $I = \{1, 2, \ldots, n\}$. This linear program has $n^2(n-1)^2/2 + n^2$ variables and $2n^2(n-1) + 2n$ constraints. Table 6 shows the dimension of theses linear programs for several values of n.

The linear programs were solved with ADP [79], a dual interior point algorithm (see Subsection 2.2). The solver produces a sequence of lower bounds (dual interior solutions), each of which can be compared with the best upper bound to decide if pruning of the search tree can be done at the node on which the lower bound is computed. Figure 7 illustrates the sequence of lower bounds produced by ADP, compared to the sequence of feasible primal solutions produced by the primal simplex code of CPLEX on QAPLIB test problem `nug15`. The figure suggests that the algorithm can be stopped many iterations prior to convergence to the optimal value and still be close in value to the optimal solution. This is important in branch and bound codes, where often a lower bound needed to prune the search tree is less than the value of the best lower bound.

Pardalos, Ramakrishnan, Resende, and Li [110] describe a branch and bound algorithm used to study the effectiveness of a variance reduction based lower bound proposed by Li, Pardalos, Ramakrishnan, and Resende [87]. This branch and bound algorithm is used by Ramakrishnan, Pardalos, and Resende [121] in conjunction with the LP-based lower bound described earlier.

In the first step, an initial upper bound is computed and an initial branch-and-bound search tree is set up. The branch and bound tree is a binary tree, each node having a left and right child. For the purpose of describing the branching process, denote, at any node of the branch and bound tree, S_A to be

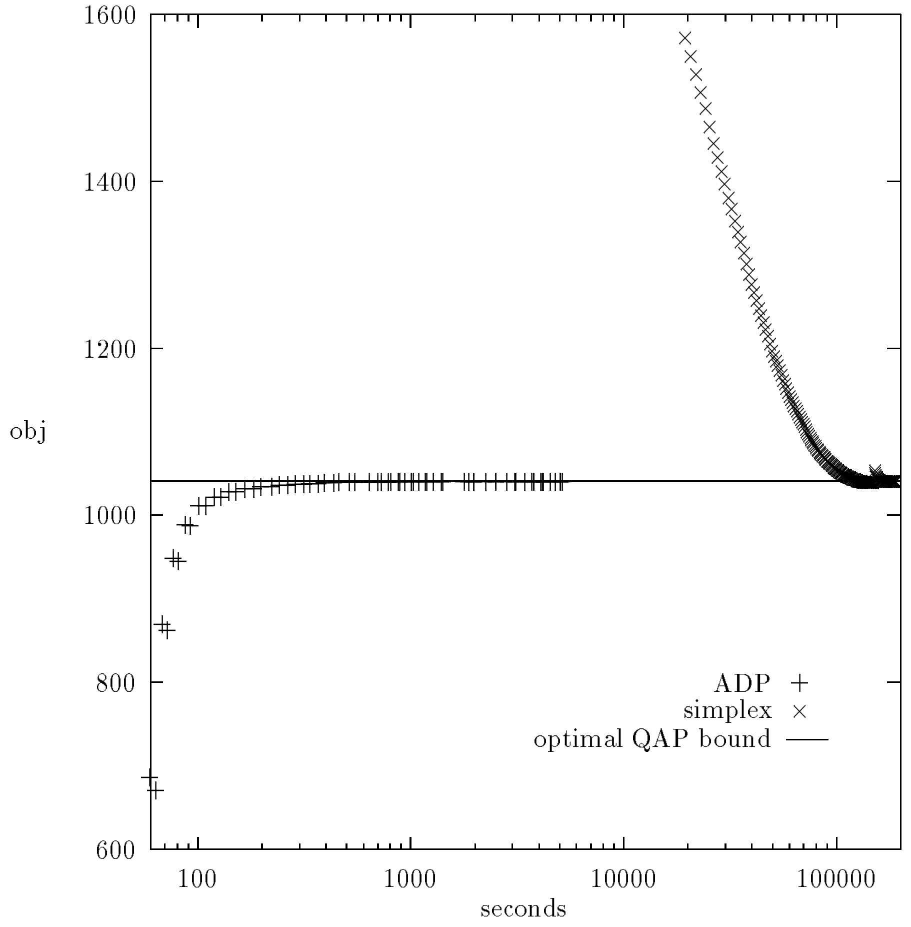

Figure 7: CPLEX simplex and ADP iterates on `nug15`

the set of already assigned facilities in the partial assignment, S_E the facilities that will never be in the partial assignment in any node of the subtree rooted at the current node. Let S_A^l, S_E^l and S_A^r, S_E^r be the corresponding sets for the left and right children of the current node. Let q denote the partial assignment at the current node. Each node of the branch and bound tree is organized as a heap with a key that is equal to the lower bound on the solution to the original QAP obtainable by any node in the subtree rooted at this node. The binary tree is organized in maximum order, i.e. the node with the largest lower bound is first.

The initial best known upper bound is computed by the GRASP heuristic described in [88, 126]. The initial search tree consists of n nodes with $S_A = \{i\}$ and $S_E = \emptyset$ for $i = 1, \ldots, n$, and $q(i) = p(i)$, where p is the permutation obtained by the GRASP and for $k \neq i$, $q(k) = 0$ and a key of 0.

In the second step, the four procedures of the branch-and-bound as described earlier are:

- *Selection*: The selection procedure simply chooses the partial permutation stored in the root of the heap, i.e. we pick the node with the maximum key.
- *Branching*: The branching procedure creates two children, the left and right children, as follows:

$$\begin{aligned} \text{pick} \quad i &\notin S_A \\ S_A^l &= S_A \\ S_E^l &= S_E \cup \{i\} \\ S_A^r &= S_A \cup \{i\} \\ S_E^r &= \emptyset \\ q^l &= q \\ q^r &= q \text{ and } q(i) = p(i), \text{where } p \text{ is the incumbent,} \end{aligned}$$

  and the key of left child is the same as the key of the current node and the key of the right child is the newly computed lower bound.
- *Elimination*: The elimination procedure compares the newly computed lower bound of the right child to the incumbent and deletes the right child if its key is greater than the incumbent, thus pruning the entire subtree rooted at the right child.
- *Termination Test*: The algorithm stops if, and only if, the heap is empty.

In the final step, a best permutation found is taken as the global optimal permutation.

As an example of the branch and bound algorithm, consider the QAPLIB instance `nug05`. The iterations of the branch and bound algorithm are summarized in Table 7. The GRASP approximation algorithm produced a solution

Table 7: Branch and bound algorithm on `nug05`

node	UB	LB	permutation				
1	52	58	1	-	-	-	-
2	52	55	2	-	-	-	-
3	52	52	5	-	-	-	-
4	52	57	3	-	-	-	-
5	52	50	4	-	-	-	-
6	52	57	4	3	-	-	-
7	52	50	4	5	-	-	-
8	52	56	4	5	3	-	-
9	52	56	4	5	2	-	-
10	52	50	4	5	1	-	-
11	52	60	4	5	1	3	-
12	52	50	4	5	1	2	-
	50	-	4	5	1	2	3
13	50	56	4	2	-	-	-
14	50	50	4	1	-	-	-

(UB) having cost 52. The branch and bound algorithm examined 14 nodes of the search tree. In the first five nodes, each facility was fixed to location 1 and the lower bounds of each branch computed. The lower bounds corresponding to branches rooted at nodes 1 through 4 were all greater than or equal to the upper bound, and thus those branches of the tree could be pruned. At node 6 a level-2 branching begins with a lower bound less than the upper bound produced at node 7. Deeper branchings are done at nodes 8, 11, and 12, at which point a new upper bound is computed having value 50. Nodes 13 and 14 complete the search. The same branch and bound algorithm using the GLB scans 44 nodes of the tree to prove optimality.

We tested the codes on several instances from the QAP library QAPLIB. Table 8 summarizes the runs on both algorithms. For each instance it displays the name and dimension of the problem, as well as the solution times and number of branch and bound search tree nodes examined by each of the algorithms. The ratio of CPU times is also displayed.

The number of GRASP iterations was set to 100,000 for all runs.

Table 9 shows statistics for the LP-based algorithm. For each run, the table lists the number of nodes examined, the number of nodes on which the lower bound obtained was greater than the best upper bound at that moment, the number of nodes on which the lower bound obtained was less than or equal to the best upper bound at that moment, and the percentage of nodes examined that were of levels 1, 2, 3, 4, and 5 or greater.

Table 8: QAP test instances: LP-based vs. GLB-based B&B algorithms

		LP-based B&B		GLB-based B&B		time	nodes
problem	dim	nodes	time	nodes	time	ratio	ratio
nug05	5	12	11.7	44	0.1	117.0	3.7
nug06	6	6	9.5	82	0.1	95.0	13.7
nug07	7	7	16.6	115	0.1	166.0	16.4
nug08	8	8	35.1	895	0.2	175.5	111.9
nug12	12	220	5238.2	49063	14.6	358.8	223.0
nug15	15	1195	87085.7	1794507	912.4	95.4	1501.7
scr10	10	19	202.1	1494	0.6	336.8	78.6
scr12	12	252	5118.7	12918	4.8	1066.4	51.3
scr15	15	228	3043.3	506360	274.7	11.1	2220.9
rou10	10	52	275.7	2683	0.8	344.6	51.6
rou12	12	152	2715.9	37982	12.3	220.8	249.9
rou15	15	991	30811.7	4846805	2240.3	13.8	4890.8
esc08a	8	8	37.4	57464	7.0	5.3	7183.0
esc08b	8	208	491.1	7352	0.7	701.6	35.3
esc08c	8	8	42.7	2552	0.3	142.3	319.0
esc08d	8	8	38.1	2216	0.3	127.0	277.0
esc08e	8	64	251.0	10376	1.0	251.0	162.1
esc08f	8	8	37.6	1520	0.3	125.3	190.0
chr12a	12	12	312.0	672	0.7	445.7	56.0
chr12b	12	12	289.4	318	0.6	482.3	26.5
chr12c	12	12	386.1	3214	1.5	257.4	267.8
chr15a	15	15	1495.9	413825	235.5	6.4	27588.3
chr15b	15	15	1831.9	396255	217.8	8.4	26417.0
chr15c	15	15	1908.5	428722	240.0	8.0	28581.5
chr18a	18	35	1600.0	$> 1.6 \times 10^9$	$> 10^6$	$< 648.0^{-1}$	$> 45 \times 10^6$

Table 9: QAP test instances: B&B tree search

	nodes of B&B tree			percentage of nodes of level				
problem	scan	good	bad	1	2	3	4	≥ 5
nug05	14	10	4	35.7	28.6	21.4	14.3	0.0
nug06	6	6	0	100.0	0.0	0.0	0.0	0.0
nug07	7	7	0	100.0	0.0	0.0	0.0	0.0
nug08	8	8	0	100.0	0.0	0.0	0.0	0.0
nug12	220	200	20	5.5	45.0	45.5	4.1	0.0
nug15	1195	1103	92	1.3	17.6	56.6	21.1	3.5
scr10	19	18	1	52.6	47.4	0.0	0.0	0.0
scr12	252	228	24	4.8	43.7	23.8	21.4	6.3
scr15	228	211	17	6.6	49.1	11.4	10.5	22.4
rou10	54	46	8	18.5	16.7	14.8	13.0	37.0
rou12	154	137	17	7.8	57.1	6.5	5.8	22.7
rou15	991	912	79	1.5	21.2	69.5	1.2	6.6
esc08a	8	8	0	100.0	0.0	0.0	0.0	0.0
esc08b	208	176	32	3.8	26.9	69.2	0.0	0.0
esc08c	8	8	0	100.0	0.0	0.0	0.0	0.0
esc08d	8	8	0	100.0	0.0	0.0	0.0	0.0
esc08e	64	56	8	12.5	87.5	0.0	0.0	0.0
esc08f	8	8	0	100.0	0.0	0.0	0.0	0.0
chr12a	12	12	0	100.0	0.0	0.0	0.0	0.0
chr12b	12	12	0	100.0	0.0	0.0	0.0	0.0
chr12c	12	12	0	100.0	0.0	0.0	0.0	0.0
chr15a	15	15	0	100.0	0.0	0.0	0.0	0.0
chr15b	15	15	0	100.0	0.0	0.0	0.0	0.0
chr15c	15	15	0	100.0	0.0	0.0	0.0	0.0
chr18a	35	17	18	51.4	48.6	0.0	0.0	0.0

6 Branch and cut methods

For some problems, branch and bound algorithms can be improved by refining the relaxations solved at each node of the tree, so that the relaxation becomes a better and better approximation to the set of integral feasible solutions. In a general branch and cut method, many linear programming relaxations are solved at each node of the tree. Like branch and bound, a branch and cut method is an exact algorithm for an integer programming problem.

In a cutting plane method, extra constraints are added to the relaxation. These extra constraints are satisfied by all feasible solutions to the integer programming problem, but they are violated by the optimal solution to the LP relaxation, so we call them *cutting planes*. As the name suggests, a branch and cut method combines a cutting plane approach with a branch and bound method, attacking the subproblems at the nodes of the tree using a cutting plane method until it appears that no further progress can be made in a reasonable amount of time.

Consider, for example, the integer programming problem

$$\begin{array}{lrcrcl}
\min & -2x_1 & - & x_2 & & \\
\text{s.t.} & x_1 & + & 2x_2 & \leq & 7 \\
 & 2x_1 & - & x_2 & \leq & 3 \\
 & & & x_1, x_2 & \geq & 0, \text{ integer.}
\end{array}$$

This problem is illustrated in figure 8. The feasible integer points are indicated. The LP relaxation is obtained by ignoring the integrality restrictions; this is given by the polyhedron contained in the solid lines. The boundary of the *convex hull* of the feasible integer points is indicated by dashed lines and can be described by the inequalities

$$\begin{array}{rcrcl}
x_1 & - & x_2 & \leq & 1 \\
x_1 & & & \leq & 2 \\
x_1 & + & x_2 & \leq & 4 \\
 & & x_2 & \leq & 3 \\
 & & x_1, x_2 & \geq & 0.
\end{array}$$

When solving this problem using a cutting plane algorithm, the linear programming relaxation is first solved, giving the point $x_1 = 2.6$, $x_2 = 2.2$, which has value -7.4. The inequalities $x_1 + x_2 \leq 4$ and $x_1 \leq 2$ are satisfied by all the feasible integer points but they are violated by the point $(2.6, 2.2)$. Thus, these two inequalities are valid *cutting planes*. Adding these two inequalities to the relaxation and solving again gives the point $x_1 = 2$, $x_2 = 2$, with value -6. Notice that this point is feasible in the original integer program, so it must actually be optimal for that problem, since it is optimal for a relaxation of the integer program.

If instead of adding both inequalities, we had just added the inequality $x_1 \leq 2$, the optimal solution to the new relaxation would have been $x_1 = 2$,

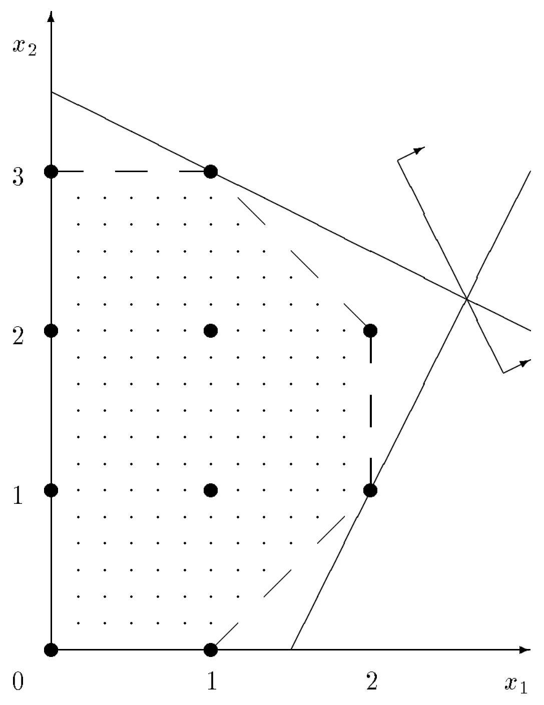

Figure 8: A cutting plane example

$x_2 = 2.5$, with value -6.5. We could then have looked for a cutting plane that separates this point from the convex hull, for example $x_1 + x_2 \leq 4$, added this to the relaxation and solved the new relaxation. This illustrates the basic structure of a cutting plane algorithm:

- Solve the linear programming relaxation.
- If the solution to the relaxation is feasible in the integer programming problem, STOP with optimality.
- Else, find one or more cutting planes that separate the optimal solution to the relaxation from the convex hull of feasible integral points, and add a subset of these constraints to the relaxation.
- Return to the first step.

Notice that the values of the relaxations provide lower bounds on the optimal value of the integer program. These lower bounds can be used to measure progress towards optimality, and to give performance guarantees on integral solutions. None of these constraints can be omitted from the description of the convex hull, and they are called *facets* of the convex hull. Cutting planes that define facets are the strongest possible cutting planes, and they should be added to the relaxation in preference to non-facet defining inequalities, if possible. Families of facet defining inequalities are known for many classes of integer programming problems (for example, the traveling salesman problem [47, 108], the matching problem [32], the linear ordering problem [48], and the maximum cut problem [27, 28]). Jünger et al. [63] contains a survey of cutting plane methods for various integer programming problems. Nemhauser and Wolsey [105] gives more background on cutting plane methods for integer programming problems.

Traditionally, Gomory cutting planes [46] were used to improve the relaxation. These cuts are formed from the optimal tableau for the LP relaxation of the integer program. Cutting plane methods fell out of favour for many years because algorithms using Gomory cuts showed slow convergence. The resurgence of interest in these methods is due to the use of specialized methods that search for facets, enabling the algorithm to converge far more rapidly. The cutting planes are determined using a *separation routine*, which is usually very problem specific. General integer programming problems have been solved by using cutting planes based on facets of the *knapsack problem* $\min\{c^T x : a^T x \leq b, x \geq 0, x \text{ integer}\}$: each constraint of the general problem can be treated as a knapsack constraint [59]. Other general cutting plane techniques include lift-and-project methods [10]. Gomory cutting planes have also been the subject of a recent investigation [11]. It appears that they are not as bad as originally thought, and they in fact work quite well if certain modifications are made, such as adding many constraints at once.

The *separation problem* for the problem $\min\{c^T x : Ax \leq b,\ x \text{ integer}\}$ can be defined as:

> Given a point $\bar{x}$, either determine that $\bar{x}$ is in the convex hull Q of the feasible integer points, or find a cutting plane that separates $\bar{x}$ from the convex hull.

Grötschel et al. [49] used the ellipsoid algorithm to show that if the separation problem can be solved in polynomial time then the problem (IP) itself can also be solved in polynomial time. It follows that the separation problem for an NP-hard problem cannot be solved in polynomial time, unless $P = NP$. Many of the separation routines in the literature are heuristics designed to find cutting planes belonging to certain families of facets; there are many undiscovered families of facets, and, for NP-hard problems, it is unlikely that a complete description of the facets of the convex hull will be discovered. Such a description would certainly contain an exponential number of facets (provided $P \neq NP$). Even small problems can have many facets. For example, the convex hull of the travelling salesman problem with only nine cities has over 42 million facets [21].

6.1 Interior point cutting plane methods

We now assume that our integer programming problem takes the form

$$\begin{array}{lrcll}
\min & c^T x & & & \\
\text{subject to} & Ax & = & b & \\
& 0 \le x & \le & u & \qquad\qquad (IP) \\
& x_i & & \text{binary for } i \text{ in I} & \\
& x & & \text{satisfies some additional conditions} &
\end{array}$$

where A is an $m \times n$ matrix of rank m, c, u, and x are n-vectors, b is an m-vector, and I is a subset of $\{1, \ldots, n\}$. We assume that these additional conditions can be represented by linear constraints, perhaps by an exponential number of such constraints. For example, the traveling salesman problem can be represented in this form, with the additional conditions being the subtour elimination constraints [47, 108], and the conditions $Ax = b$ representing the degree constraints that the tour must enter and leave each vertex exactly once. It is also possible that the problem does not need any such additional conditions. Of course, problems with inequality constraints can be written in this form by including slack variables. Note that we allow a mixture of integer and continuous variables. In this section, we describe cutting plane methods to solve (IP) where the LP relaxations are solved using *interior point methods.* Computational experience with interior point cutting plane methods is described in [99, 96, 101]. Previous surveys on interior point cutting plane methods include [94, 95].

It has been observed that interior point algorithms do not work very well when started from close to a nonoptimal extreme point. Of course, this is exactly what we will have to do if we solve the LP relaxation to optimality, since the fractional optimal solution to the relaxation will be a nonoptimal infeasible extreme point after adding a cutting plane. The principal method used to overcome this drawback is to only solve the relaxation approximately. We use this approximate solution to generate an integral feasible point that is, with luck, close to the optimal integral solution. The best integral solution found so far gives an upper bound on the optimal value of (IP) and the value of the dual solution gives a lower bound. A conceptual interior point cutting plane algorithm is given in Figure 9.

To make this algorithm practical, we have to decide how accurately to solve the relaxations. Notice that if the entries in c are integral then it is sufficient to reduce the gap between the integral solution and the lower bound to be less than one. Other refinements include methods for choosing which cuts to add, generating good integral solutions, dropping unimportant constraints, and fixing variables at their bounds. We discuss all of these issues, and conclude by presenting the complete algorithm, and some typical computational results.

In what follows, we refer several times to the linear ordering problem and to finding the ground state of an Ising spin glass with no external force, which we call the Ising spin glass problem. We now define those problems.

1. Solve the current relaxation of (IP) approximately using an interior point method.
2. Generate an integral feasible solution from the approximate primal solution.
3. If the gap between the best integral solution found so far and the best lower bound provided by a dual solution is sufficiently small, STOP with an optimal solution to the original problem.
4. Otherwise, use a separation routine to generate cutting planes, add these constraints to the LP relaxation, and return to Step 1.

Figure 9: A conceptual interior point cutting plane algorithm

The linear ordering problem:
Given p sectors with costs g_{ij} for placing sector i before sector j for each pair i and j of sectors, find a permutation of the sectors with minimum total cost.

This problem can be represented algebraically as follows:

$$\begin{array}{lrcl} \min & \sum_{1\le i\le p, 1\le j\le p, i\ne j} g_{ij}x_{ij} & & \\ \text{subject to} & x_{ij}+x_{ji} & = & 1 \text{ for all pairs } i \text{ and } j \\ & x & & \text{binary} \\ & x & & \text{satisfies the triangle inequalities,} \end{array}$$

where the triangle inequalities require that

$$x_{ij} + x_{jk} + x_{ki} \le 2$$

for each triple (i, j, k). When this problem is solved using a cutting plane approach, the triangle inequalities are used as cuts. They define facets of the convex hull of feasible solutions. Other facets are known (see Grötschel et al. [48]), but these prove to be unnecessary for many problems.

The Ising spin glass problem:
Given a grid of points on a torus, and given interaction forces c_{ij} between each point and each of its neighbours, partition the vertices into two sets to minimize the total cost, where the total cost is the sum of all interaction forces between vertices that are in different sets.

The physical interpretation of this problem is that each point possesses either a positive or a negative charge, and the interaction force will be either $+1$ or

-1 depending on the charges on the neighbours. The interactions between the points can be measured, but the charges at the points cannot and need to be determined. The Ising spin glass problem is a special case of the *maximum cut problem*:

> Given a graph $G = (V, E)$ and edge weights w_{ij}, partition the vertices into two sets to maximize the value of the cut, that is, the sum of the weights of edges where the two ends of the edge are in opposite sets of the partition.

This problem can be represented as an integer program, where x_e indicates whether e is in the cut:

$$\begin{array}{lrl} \min & c^T x & \\ \text{subject to} & x & \text{is binary} \\ & x & \text{satisfies the cycle/cut inequalities.} \end{array}$$

The cycle/cut inequalities exploit the fact that every cycle and every cut intersect in an even number of edges. They can be stated as

$$x(F) - x(C \setminus F) \leq \mid F \mid -1$$

for sets F of odd cardinality, where C is a cycle in the graph, and $x(S) := \sum_{e \in S} x_e$ for any subset S of the edges. An inequality of this form defines a facet if the cycle C is chordless.

6.2 Solving the relaxations approximately

The principle technique used to make an interior point cutting plane algorithm practical is early termination: the current relaxation is only solved approximately. Typically, the relaxations are solved more exactly as the algorithm proceeds.

There are two main, related, advantages to early termination. In the first place, iterations are saved on the current relaxation and the early solution is usually good enough to enable the efficient detection of cutting planes, so solving the relaxation to optimality would not provide any additional information but would require additional computational effort. Secondly, the approximate solution provides a better starting point for the method on the next relaxation, because it is more centered than the optimal solution.

The disadvantages result from the fact that the solution to the current relaxation may be the optimal solution to the integer program. It is possible that the approximate solution is not in the convex hull of feasible integral solutions, even though the optimal solution is in this set, and so cutting planes may be generated and the relaxation may be modified unnecessarily. On the other hand, if the approximate solution is in the convex hull, the separation routines will not find cutting planes, but time will be wasted in trying to find cuts. The effect of the first drawback can be mitigated by initializing the relaxation with a point that is not too far from the center of the convex hull, and by solving the relaxation to optimality occasionally, for example on every tenth relaxation. This last technique proved to be very useful in the experiments on Ising spin glass problems described in [96].

One way to reduce the cost of the first drawback is to control how accurately the relaxations are solved by using a dynamically adjusted tolerance for the duality gap: one searches for cutting planes once the duality gap falls below this tolerance. If many cutting planes are found, then perhaps one did not need to solve the current relaxation so accurately, so one can increase this tolerance. On the other hand, if only a few cutting planes are found then the tolerance should be decreased. In most of those experiments, the tolerance was initiliazed with a value of 0.3 on the relative duality gap and then was modified by multiplying by a power of 1.1, with the power depending on the number of cuts found and on how badly these cuts were violated by the current iterate.

Other ways to control the accuracy include requiring that the current primal solution should have better value than the best known integral solution, and that the dual solution should be better than the best known lower bound. Perhaps surprisingly, it was observed that the condition based on the dual value is in general too restrictive, forcing the algorithm to perform more iterations than necessary on the current relaxation without resulting in a reduction in the total number of relaxations solved. Mitchell has even had mixed results with the condition on the primal solution: for the linear ordering problem, this condition resulted in an increase in computational times, but it improved runtimes for Ising spin glass problems. (The Ising spin glass problems are harder than the linear ordering problems.)

A more sophisticated condition is to require that the relaxations be solved to an accuracy such that it appears that the optimal value to the relaxation would not be sufficiently good to enable the algorithm to terminate, unless it provided an optimal integral solution. For example, one can require that the average of the primal and dual values should be at least one less than the best known integral value, if all the data is integral. If one solves such a relaxation to optimality, the lower bound would not be sufficient to prove optimality with the current best known integral solution. A similar condition was helpful for the Ising spin glass problems.

6.3 Restarting

When cutting planes are added, the current primal iterate is no longer feasible, and the algorithm must be restarted. It is possible to restart from the current iterate using a primal-dual infeasible interior point method, perhaps with an initial centering step, but it has been observed that other techniques have proved superior in practice.

After adding cutting planes, the primal relaxation becomes

$$\begin{array}{lccccl} \min & c^T x & & & & \\ \text{subject to} & Ax & & = & b & \\ & A^o x & + \; x^o & = & b^o & (LPnew) \\ & 0 & \leq \; x & \leq & u & \\ & 0 & \leq \; x^o & \leq & u^o, & \end{array}$$

where x_o is the vector of slack variables for the cutting planes given by $A^o x \leq b^o$. The dual $(LDnew)$ to this problem can be written

$$\begin{array}{lcccccccl} \max & b^T y & + \; b^{oT} y^o & & - \; u^T w & & - \; u^{oT} w^o & & \\ \text{subject to} & A^T y & + \; A^{oT} y^o & + \; z & - \; w & & & = & c \\ & & y^o & & & + \; z^o & - \; w^o & = & 0 \\ & & & & & z, z^o & w, w^o & \geq & 0. \end{array}$$

Since one uses an interior point method, and did not solve the last relaxation to optimality, the last iterate is a primal dual pair $\bar{x}$, $(\bar{y}, \bar{z}, \bar{w})$ satisfying $A\bar{x} = b$, $0 < \bar{x} < u$, $A^T \bar{y} + \bar{z} - \bar{w} = c$, $\bar{z} > 0$, $\bar{w} > 0$.

A feasible interior solution to $(LDnew)$ is obtained by setting $y^o = 0$ and $z^o = w^o = \epsilon$ for any positive ϵ (a typical value is 10^{-3}). It is often beneficial to update the dual solution to an older iterate than $(\bar{y}, \bar{z}, \bar{w})$, which will be more centered. It is also useful to increase any small components of w or z up to ϵ if necessary; if w_i is increased, then z_i is also increased to maintain dual feasibility, and vice versa.

Primal feasibility is harder to maintain, since $A^o \bar{x} > b$. Possible updating schemes are based upon knowing a point x^Q in the interior of the convex hull of feasible integer points. Of course, such a point will be an interior point in $(LPnew)$. One can either update to this point, or to an appropriate convex combination of this point and $\bar{x}$. It is often straightforward to initialize x^Q: for the linear ordering problem and for the maximum cut problem, one can take x^Q to be the vector of halves; for the undirected traveling salesman problem on a complete graph with n cities, one can take each component of x^Q to be $2/(n-1)$ (each component corresponds to an edge; an edge e is in the tour if and only if $x_e = 1$). The point x^Q can be updated by moving towards the current primal iterate or by moving towards the best integral solution found so far. For the Ising spin glass problem, Mitchell found it best not to update x^Q, but to restart by taking a convex combination of x^Q and $\bar{x}$ which was 95% of the way from x^Q to the boundary of new the relaxation. On the other hand, updating x^Q by moving towards the current primal iterate worked well for the linear ordering problem. Another possible restarting scheme is to store earlier iterates, and take the most recent iterate that is feasible in the current relaxation. This works well on some instances, but it is generally outperformed by methods based on x^Q.

6.4 Primal heuristics and termination

A good primal heuristic can save many iterations and stages, especially if the objective function data is integral. The algorithm terminates when the gap between the best known integral solution and the best lower bound drops below some tolerance. If the data is integer, then a tolerance of one is sufficient, so it is not necessary to refine the relaxation to such a degree that it completely describes the convex hull in the region of the optimal integral solution.

The importance of the primal heuristic varies from problem to problem. Mitchell found that his runtimes improved dramatically for the Ising spin glass problem when he implemented a good local search heuristic, even though the heuristic itself required as much as 60% of the total runtime on some large instances. The primal heuristic was not nearly so important for the linear ordering problem, where it was relatively easy to generate the optimal ordering from a very good fractional solution. Another indication of the difference in the importance of the primal heuristic for these two problems could be observed when Mitchell tried to solve them so that the gap between the best integral solution and the lower bound was less than, say, 10^{-6}. The linear ordering problems could be solved almost as easily as before, but the larger spin glass problems became computationally intractable.

6.5 Separation routines

Separation routines are problem specific. Good routines for simplex based cutting plane algorithms can usually be adapted to interior point cutting plane methods. Because the iterates generated by the interior point approach are more centered, it may be possible to find deeper cuts and cutting planes that are more important. This is a topic that warrants further investigation.

One issue that is specific to separation routines for interior point cutting plane algorithms is the effect of the cutting planes on the sparsity of the matrix AA^T. (Here, A represents the whole constraint matrix.) If the structure of this matrix is unfavourable, then a simplex method will outperform an interior point method based on Cholesky factorization, even for the linear programming relaxation (see, for example, [90]). For this reason, it is often useful to add cuts that are variable-disjoint, that is, a particular x_i appears in just one of the constraints added at a particular stage.

6.6 Fixing variables

When using a simplex cutting plane algorithm, it is well known that a variable can be fixed at zero or one if the corresponding reduced cost is sufficiently large (see, for example [108]). The dual variables can be used for the same purpose if an interior point method is used.

When using a cutting plane algorithm, an upper bound v_U on the optimal value is provided by a feasible integral solution. Let $\bar{v}$ be the value of the current dual iterate $(\bar{y}, \bar{z}, \bar{w})$. It was shown in [97] that if $z_i > v_U - \bar{v}$ then x_i must be zero in any optimal integral solution. Similarly, if $\bar{w}_i > v_U - \bar{v}$ then x_i must be one in any optimal solution.

These techniques can be very useful for reducing the size of the relaxations. They are most useful when the objective function data is fractional, since the gap between the upper and lower bounds has to become small in order to prove optimality, so many of the dual variables will eventually be large enough that the integral variables can be fixed.

Notice that if a variable is fixed, and thus eliminated from the relaxation, the point x^Q is no longer feasible. Therefore care has to be taken when restarting the algorithm. In particular, it is useful to examine the logical implications of fixing a variable; it may be possible to fix further variables, or to impose constraints on the remaining variables. For example, when solving a maximum cut problem, if one fixes two of the edges of a cycle of length 3 then the third edge can also be fixed. If one fixes one edge of a cycle of length 3, then the variables for the other two edges can be constrained to either take the same values as each other, or to take opposite values, depending on the value of the fixed edge. Fixing variables and adding these logical constraints can worsen the conditioning of the constraint matrix, perhaps introducing rank deficiencies. Thus, care must be exercised.

6.7 Dropping constraints

Dropping unimportant constraints reduces the size of the relaxation and so enables the relaxation to be solved more quickly. It is possible to develop tests based on ellipsoids to determine when a constraint can be dropped, but the cost of these tests outweighs the computational savings. Therefore, an implementation will generally drop a constraint based on the simple test that its slack variable is large. Of course, it is undesirable to have a constraint be repeatedly added and dropped; a possible remedy is to insist that a constraint cannot be dropped for several stages.

The development of efficient, rigorous tests for dropping constraints would be useful.

6.8 The complete algorithm

The complete algorithm is contained in figure 10.

If a primal feasible solution is known, v^U can be initialized in Step 1 to take the value of that solution; otherwise v^U should be some large number. If all the objective function coefficients c_i correspond to binary variables, then the lower bound v^L can be initialized to be $\sum_i min\{c_i, 0\}$; otherwise, the lower bound can be taken to be a negative number with large absolute value.

1. **Initialize:** Read in the problem. Set up the initial relaxation. Find initial interior feasible primal and dual points. Find a point x^Q in the interior of the convex hull of feasible integral solutions. Choose a tolerance τ on optimality for the integer program. Choose a tolerance ρ on the duality gap for the relaxation. Initialize the upper and lower bounds v^U and v^L on the optimal value appropriately.

2. **Iterate:** Take a primal-dual predictor-corrector step from the current iterate.

3. **Add cuts?** If the relative duality gap δ is smaller than ρ (and perhaps if other conditions on the primal and dual values are met), then go to Step 4; otherwise, return to Step 2.

4. **Primal heuristic:** Search for a good integral solution, starting from the current primal iterate. Update v^U if a solution is found which is better than this bound.

5. **Check for optimality:** If $v^U - v^L < \tau$, STOP: the best integer solution found so far is optimal.

6. **Search for cutting planes:** Use the separation routines to find cutting planes. If cutting planes are found, go to Step 7. If none are found and $\delta \geq 10^{-8}$, reduce ρ and return to Step 2. If none are found and $\delta < 10^{-8}$ then STOP with a nonoptimal solution; use branch and bound to find the optimal solution.

7. **Modify the relaxation:** Add an appropriate subset of the violated constraints to the relaxation. Increase ρ if it appears that the relaxations do not need to be solved so accurately. Decrease ρ if it appears that the relaxations need to be solved more accurately. Fix any variables if possible, and add any resulting constraints. Drop unimportant constraints.

8. **Restart:** Update the primal and dual solutions to give feasible interior points in the new relaxation. Return to Step 2.

Figure 10: An interior point cutting plane algorithm

L	Sample Size	Mean	Std Dev	Minimum	Maximum
10	100	0.42	0.20	0.17	1.17
20	100	4.87	2.01	1.30	12.48
30	100	24.32	11.84	7.42	87.00
40	100	88.46	43.68	32.50	259.02
50	100	272.86	151.59	96.35	795.50
60	100	860.57	969.79	227.38	7450.18
70	100	1946.14	1286.13	593.57	8370.37
80	100	5504.11	4981.00	1403.27	32470.40
90	100	10984.82	6683.37	2474.20	28785.30
100	100	12030.69	3879.55	3855.02	21922.60

Table 10: Time (seconds) to solve Ising spin glass problems

6.9 Some computational results

We present some computational results for Ising spin glass problems on grid of sizes up to 100×100 in Table 10. For comparison, De Simone et al. [26] have solved problems of size up to 70×70 with a simplex cutting plane algorithm using CPLEX3.0 on a Sun Sparc 10 workstation, requiring up to a day for each problem. The results in Table 10 were obtained on a Sun Sparc 20/71, and are taken from [98]. As can be seen, even the largest problems were solved in an average of less than $3\frac{1}{2}$ hours. They needed approximately nine iterations per relaxation — the later relaxations required more iterations and the earlier relaxations fewer. The primal heuristic took approximately 40% of the total runtime.

6.10 Combining interior point and simplex cutting plane algorithms

Practical experience with interior point cutting plane algorithms has shown that often initially they add a large number of constraints at a time (hundreds or even thousands), and the number of added constraints decreases to just a handful at a time towards the end. The number of iterations to reoptimize increases slightly as optimality is approached, because the relaxations are solved to a higher degree of accuracy.

When a simplex method is used to solve the relaxations, the number of iterations to reoptimize depends greatly on the number of added constraints. Initially, when many constraints are added, the dual simplex method can take a long time to reoptimize, but towards the end it can reoptimize in very few iterations, perhaps as few as ten.

Because of the time required for an iteration of an interior point method, it is very hard to compete with the speed of simplex for solving these last few relaxations. Conversely, the interior point method is considerably faster for the first few stages. The interior point method may also make a better selection of cutting planes in these initial stages, because it is cutting off an interior point that is well-centered, a property that is intensified because it is looking for cutting planes before termination.

Mitchell and Borchers [100] investigated solving linear ordering problems with a cutting plane code that uses an interior point method for the first few stages and a dual simplex method for the last few stages. Computational results are contained in table 11. These problems have up to 250 sectors, with a

n	% zeros	Interior	Simplex	Combined
150	0	206	75	68
200	0	755	385	209
250	0	4492	3797	592
100	20%	1405	1296	230
150	10%	2247	1294	208
200	10%	N/A	9984	879

Table 11: Preliminary Results on Linear Ordering Problems.

percentage of the cost entries zeroed out. The nonzero costs above the diagonal were uniformly distributed between 0 and 99, and those below the diagonal were uniformly distributed between 0 and 39. The table contains runtimes in seconds on a Sun SPARC 20/71 for an interior point cutting plane code, a simplex cutting plane code using CPLEX 4.0, and a combined cutting plane code. The interior point code was unable to solve the problems with 200 sectors and 20% of the entries zeroed out because of space limitations. As can be seen the combined code is more than 10 times faster than the simplex code on the largest problems, and the interior point and simplex codes require similar amounts of time, at least on the harder problems.

6.11 Interior point column generation methods for other problems

A cutting plane method can be regarded as a column generation method applied to the dual problem. Interior point methods have been successfully applied in several other situations amenable to solution by a column generation approach. Goffin et al. [42] have solved nondifferentiable optimization problems. Bahn et al. [9] have used an interior point method within the L-shaped decomposition method of Van Slyke and Wets [136] for stochastic programming problems.

Goffin et al. [41] have also solved multicommodity network flow problems using an interior point column generation approach. In this method, the columns correspond to different paths from an origin to a destination, and they are generated by solving a shortest path problem with an appropriate cost vector.

6.12 Theoretical issues and future directions

As mentioned earlier, the ellipsoid algorithm can be used to solve an integer programming problem in polynomial time if the separation problem can be solved in polynomial time. It is not currently known how to use an interior point method in an exactly analogous way. Atkinson and Vaidya [8] developed an interior point algorithm for this process, but their algorithm requires that unimportant constraints be dropped, unlike the ellipsoid algorithm. Vaidya later obtained a similar result for an algorithm that used the volumetric center [138]. Goffin et al. [44] have proposed a fully polynomial algorithm that does not require that unimportant constraints be removed. It is an interesting open theoretical question to find an interior point algorithm that does not require that unimportant constraints be removed, and also solves the optimization problem in polynomial time provided the separation problem can be solved in polynomial time.

The algorithms proposed in [8, 138, 44] required that only a single constraint be added at a time, and that the constraint be added far from the current iterate. These algorithms have been extended to situations where many cuts are added at once, and the constraints are added right through the current iterate, with no great increase in the complexity bound [124, 125, 43]. It has been shown that if p constraints are added through the analytic center then the analytic center of the new feasible region can be found in $O(\sqrt{p})$ iterations [124].

There are several open computational questions with interior point cutting plane methods. Combining interior point methods with the simplex algorithm needs to be investigated further. When a direct method is used to calculate the Newton direction, it is necessary to choose an ordering of the columns of AA^T to reduce fill in the Cholesky factor; it would be interesting to see if the ordering from one stage can be efficiently modified to give an ordering for the next stage, rather than calculating an ordering from scratch. When the constraint matrix contains many dense columns, it becomes expensive to use a direct method to calculate the Newton direction; it would be interesting to examine whether it is efficient to switch to a preconditioned conjugate gradient method in the later stages.

7 Nonconvex potential function minimization

Consider the problem of maximizing a convex quadratic function defined as

$$\max\ w^T w = \sum_{i=1}^{m} w_i^2 \tag{62}$$

subject to

$$A^T w \leq b. \tag{63}$$

The significance of this optimization problem is that many combinatorial optimization problems can be formulated as above with the additional requirement that the variables are binary.

In [73, 77] a new affine scaling algorithm was proposed for solving the above problem using a logarithmic potential function. Consider the nonconvex optimization problem

$$\min\ \{\varphi(w) \mid A^T w \leq b\}, \tag{64}$$

where

$$\begin{aligned} \varphi(w) &= \log(m - w^T w)^{1/2} - \frac{1}{n}\sum_{i=1}^{n} \log d_i(w) & (65) \\ &= \log\left\{\frac{m - w^T w}{2\prod_{i=1}^{n} d_i(w)^{1/n}}\right\} & (66) \end{aligned}$$

and where

$$d_i(w) = b_i - a_i^T w, \ \ i = 1, \ldots, n, \tag{67}$$

are the slacks. The denominator of the log term of $\varphi(w)$ is the geometric mean of the slacks and is maximized at the analytic center of the polytope defined by

$$\mathcal{L} = \left\{ w \in \mathsf{R}^m \mid A^T w \leq b \right\}.$$

To find a local (perhaps global) solution of (64), an approach similar to the classical Levenberg-Marquardt methods [86, 91] is used. Let

$$w^0 \in \mathcal{L}^0 = \left\{ w \in \mathsf{R}^m \mid A^T w < b \right\}$$

be a given initial interior point. The algorithm generates a sequence of interior points of $\mathcal{L}$.

Let $w^k \in \mathcal{L}^0$ be the k-th iterate. Around w^k a quadratic approximation of the potential function is set up. Let $D = \mathrm{diag}(d_1(w), \ldots, d_n(w))$, $e = (1, \ldots, 1)$, $f_0 = m - w^T w$ and C be a constant. The quadratic approximation of $\varphi(w)$ around w^k is given by

$$Q(w) = \frac{1}{2}(w - w^k)^T H (w - w^k) + h^T (w - w^k) + C \tag{68}$$

where the Hessian is

$$H = -\frac{1}{f_0} I - \frac{2}{f_0^2} w^k {w^k}^T + \frac{1}{n} A D^{-2} A^T \tag{69}$$

and the gradient is

$$h = -\frac{1}{f_0} w^k + \frac{1}{n} A D^{-1} e. \tag{70}$$

Recall that minimizing (68) over a polytope is NP-complete. However, if the polytope is substituted by an inscribed ellipsoid, the resulting approximate problem can be solved in polynomial time [147]. Since preliminary implementations of this algorithm indicate that trust region methods are more efficient for solving these problems, in the discussion that follows we consider a trust region approach.

Consider the ellipsoid

$$\mathcal{E}(r) = \left\{ w \in \mathsf{R}^m \mid (w - w^k)^T A D^{-2} A^T (w - w^k) \leq r^2 \right\}.$$

To see that the ellipsoid $\mathcal{E}(r)$ is inscribed in the polytope $\mathcal{L}$, assume that $r = 1$ and let $y \in \mathcal{E}(1)$. Then

$$(y - w^k)^T A D^{-2} A^T (y - w^k) \leq 1$$

and consequently

$$D^{-1} A^T (y - w^k) \leq e,$$

where $w^k \in \mathcal{L}^0$. Denoting the i-th row of A^T by $a_{i\cdot}^T$, we have

$$\frac{1}{b_i - a_{i\cdot}^T w^k} a_{i\cdot}^T (y - w^k) \leq 1, \; \forall i = 1, \ldots, n.$$

Hence,

$$a_{i\cdot}^T (y - w^k) \leq b_i - a_{i\cdot}^T w^k, \quad \forall i = 1, \ldots, n,$$

and consequently

$$a_{i\cdot}^T y \leq b_i, \quad \forall i = 1, \ldots, n,$$

i.e. $A^T y \leq b$, showing that $y \in \mathcal{L}$. This shows that $\mathcal{E}(1) \subset \mathcal{L}$ and since $\mathcal{E}(r) \subset \mathcal{E}(1)$, for $0 \leq r < 1$, then $\mathcal{E}(r) \subset \mathcal{L}$, i.e. $\mathcal{E}(r)$ is an inscribed ellipsoid in $\mathcal{L}$.

Substituting the polytope by the appropriate inscribed ellipsoid and letting $\Delta w \equiv w - w^k$ results in the minimization of a quadratic function over an ellipsoid, i.e.

$$\min \; \frac{1}{2} (\Delta w)^T H \Delta w + h^T \Delta w \tag{71}$$

subject to

$$(\Delta w)^T A D^{-2} A^T (\Delta w) \leq r^2. \tag{72}$$

The optimal solution Δw^* to (71–72) is a descent direction of $Q(w)$ from w^k. For a given radius $r > 0$, the value of the original potential function $\varphi(w)$ may increase by moving in the direction Δw^*, because of the higher order terms ignored in the approximation. It can be easily verified, however, that if the radius is decreased sufficiently, the value of the potential function will decrease by moving in the new Δw^* direction. We shall say a *local minimum* to (64) has been found if the radius must be reduced below a tolerance ϵ to achieve a reduction in the value of the potential function.

The following result, proved in [73], characterizes the optimal solution of (71–72). Using a linear transformation, the problem is transformed into the minimization of a quadratic function over a sphere.

Consider the optimization problem

$$\min \frac{1}{2} x^T Q x + c^T x \tag{73}$$

subject to

$$x^T x \leq r^2, \tag{74}$$

where $Q \in \mathsf{R}^{m \times m}$ is symmetric and indefinite, $x, c \in \mathsf{R}^m$ and $0 < r \in \mathsf{R}$. Let $u_1, \ldots, u_m$ denote a full set of orthonormal eigenvectors spanning R^m and let $\lambda_1, \ldots, \lambda_m$ be the corresponding eigenvalues ordered so that $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{m-1} \leq \lambda_m$. Denote $0 > \lambda_{min} = \min\{\lambda_1, \ldots, \lambda_m\}$ and u_{min} the corresponding eigenvector. Furthermore, let q be such that $\lambda_{min} = \lambda_1 = \cdots = \lambda_q < \lambda_{q+1}$. To describe the solution to (73–74) consider two cases:

Case 1: Assume $\sum_{i=1}^{q} (c^T u_i)^2 > 0$. Let the scalar $\lambda \in (-\infty, \lambda_{min})$ and consider the parametric family of vectors

$$x(\lambda) = -\sum_{i=1}^{m} \frac{(c^T u_i) u_i}{\lambda_i - \lambda}.$$

For any $r > 0$, denote by $\lambda(r)$ the unique solution of the equation $x(\lambda)^T x(\lambda) = r^2$ in λ. Then $x(\lambda(r))$ is the unique optimal solution of (73–74).

Case 2: Assume $c^T u_i = 0, \forall i = 1, \ldots, q$. Let the scalar $\lambda \in (-\infty, \lambda_{min})$ and consider the parametric family of vectors

$$x(\lambda) = -\sum_{i=q+1}^{m} \frac{(c^T u_i) u_i}{\lambda_i - \lambda}. \tag{75}$$

Let

$$r_{max} = \|x(\lambda_{min})\|_2.$$

If $r < r_{max}$ then for any $0 < r < r_{max}$, denote by $\lambda(r)$ the unique solution of the equation $x(\lambda)^T x(\lambda) = r^2$ in λ. Then $x(\lambda(r))$ is the unique optimal solution

of (73–74).
If $r \geq r_{max}$, then let $\alpha_1, \alpha_2, \ldots, \alpha_q$ be any real scalars such that

$$\sum_{i=1}^{q} \alpha_i^2 = r^2 - r_{max}^2.$$

Then

$$x = \sum_{i}^{q} \alpha_i u_i - \sum_{i=q+1}^{m} \frac{(c^T u_i) u_i}{(\lambda_i - \lambda_{min})}$$

is an optimal solution of (73–74). Since the choice of α_i's is arbitrary, this solution is not unique.

This shows the existence of a unique optimal solution to (73–74) if $r < r_{max}$. The proof of this result is based on another fact, used to develop the algorithm described in [73, 77], that we state next.

Let the length of $x(\lambda)$ be

$$l\,(x(\lambda)) \equiv \|x(\lambda)\|_2^2 = x(\lambda)^T x(\lambda),$$

then $l\,(x(\lambda))$ is monotonically increasing in λ in the interval $\lambda \in (-\infty, \lambda_{min})$. To see this is so, consider two cases. First, assume $\sum_{i=1}^{q} (c^T u_i)^2 > 0$. Consider the parametric family of vectors

$$x(\lambda) = -\sum_{i=1}^{m} \frac{(c^T u_i) u_i}{\lambda_i - \lambda},$$

for $\lambda \in (-\infty, \lambda_{min})$. Now, assume that $c^T u_i = 0, \forall i = 1, \ldots, q$ and consider the parametric family of vectors

$$x(\lambda) = -\sum_{i=q+1}^{m} \frac{(c^T u_i) u_i}{\lambda_i - \lambda}, \tag{76}$$

for $\lambda \in (-\infty, \lambda_{min})$. Furthermore, assume

$$r < \|x(\lambda_{min})\|_2.$$

Then $l\,(x(\lambda))$ is monotonically increasing in λ in the interval $\lambda \in (-\infty, \lambda_{min})$.

The above result suggests an approach to solve the nonconvex optimization problem (64). At each iteration, a quadratic approximation of the potential function $\varphi(w)$ around the iterate w^k is minimized over an ellipsoid inscribed in the polytope $\{w \in \mathsf{R}^m | A^T w \leq b\}$ and centered at w^k. Either a descent direction Δw^* of $\varphi(w)$ is produced or w^k is said to be a local minimum. A new iterate w^{k+1} is computed by moving from w^k in the direction Δw^* such that $\varphi(w^{k+1}) < \varphi(w^k)$. This can be done by moving a fixed step α in the direction

```
**procedure** cmq($n, A, b, \mu_0, \underline{l}_0, \bar{l}_0$)
1    $k = 0;\ \gamma = 1/(\mu_0 + 1/n);\ \underline{l} = \underline{l}_0;\ \bar{l} = \bar{l}_0;\ K = 0;$
2    $w^k$ = get_start_point($A, b$);
3    **do** $\bar{l} > \epsilon \rightarrow$
4        $\Delta w^*$ = descent_direction($\gamma, w^k, \underline{l}, \bar{l}$);
5        **do** $\varphi(w^k + \alpha\Delta w^*) \geq \varphi(w^k)$ **and** $\bar{l} > \epsilon \rightarrow$
6            $\bar{l} = \bar{l}/\bar{l}_r$;
7            $\Delta w^*$ = descent_direction($\gamma, w^k, \underline{l}, \bar{l}$);
8        **od**;
9        **if** $\varphi(w^k + \alpha\Delta w^*) < \varphi(w^k) \rightarrow$
10           $w^{k+1} = w^k + \alpha\Delta w^*$;
11           $k = k + 1$;
12       **fi**;
13   **od**;
**end** cmq;
```

Figure 11: Procedure cmq: Algorithm for nonconvex potential function minimization

Δw^* or by doing a line search to find α that minimizes the potential function $\varphi(w^k + \alpha\Delta w^*)$ [134].

Figure 11 shows a pseudo-code procedure cmq, for finding a local minimum of the convex quadratic maximization problem. Procedure cmq takes as input the problem dimension n, the A matrix, the b right hand side vector, an initial estimate μ_0 of parameter μ and initial lower and upper bounds on the acceptable length, $\underline{l}_0$ and $\bar{l}_0$, respectively. In line 2, get_start_point returns a strict interior point of the polytope under consideration, i.e. $w^k \in \mathcal{L}^0$.

The algorithm iterates in the loop between lines 3 and 13, terminating when a local optimum is found. At each iteration, a descent direction of the potential function $\varphi(w)$ is produced in lines 4 through 8. In line 4, the minimization of a quadratic function over an ellipsoid (71–72) is solved. Because of higher order terms the direction returned by descent_direction may not be a descent direction for $\varphi(w)$. In this case, loop 5 to 8 is repeated until an improving direction for the potential function is produced or the largest acceptable length falls below a given tolerance ϵ.

If an improving direction for $\varphi(w)$ is found, a new point w^{k+1} is defined (in line 10) by moving from the current iterate w^k in the direction Δw^* by a step length $\alpha < 1$.

7.1 Computing the descent direction

Now consider in more detail the computation of the descent direction for the potential function. The algorithm described in this section is similar to the trust region method described in Moré and Sorensen [104].

As discussed previously, the algorithm solves the optimization problem

$$\min \ \frac{1}{2}(\Delta w)^T H \Delta w + h^T \Delta w \tag{77}$$

subject to

$$(\Delta w)^T AD^{-2}A^T \Delta w \le r^2 \le 1 \tag{78}$$

to produce a descent direction Δw^* for the potential function $\varphi(w)$. A solution $\Delta w^* \in \mathsf{R}^m$ to (77–78) is optimal if and only if there exists $\mu \ge 0$ such that

$$\left(H + \mu AD^{-2}A^T\right) \Delta w^* = -h \tag{79}$$

$$\mu\left((\Delta w^*)^T AD^{-2}A^T \Delta w^* - r^2\right) = 0 \tag{80}$$

$$H + \mu AD^{-2}A^T \text{ is positive semidefinite.} \tag{81}$$

With the change of variables $\gamma = 1/(\mu + 1/n)$ and substituting the Hessian (69) and the gradient (70) into (79) we obtain

$$\begin{aligned}\Delta w^* \ &= \ -\left(AD^{-2}A^T - \frac{2\gamma}{f_0^2} w^k {w^k}^T - \frac{\gamma}{f_0} I\right)^{-1} \times \\ &\qquad \gamma\left(-\frac{1}{f_0} w^k + \frac{1}{n} AD^{-1}e\right)\end{aligned} \tag{82}$$

that satisfies (79). Note that r does not appear in (82) and that (82) is not defined for all values of r. However, if the radius r of the ellipsoid (78) is kept within a certain range, then there exists an interval $0 \le \gamma \le \gamma_{max}$ such that

$$AD^{-2}A^T - \frac{2\gamma}{f_0^2} w^k {w^k}^T - \frac{\gamma}{f_0} I \tag{83}$$

is nonsingular. Next, we show that for γ small enough Δw^* is a descent direction of $\varphi(w)$. Note that

$$\begin{aligned}\Delta w^* \ &= \ -\left(AD^{-2}A^T - \frac{2\gamma}{f_0^2} w^k {w^k}^T - \frac{1\gamma}{f_0} I\right)^{-1} \gamma\left(-\frac{1}{f_0} w^k + \frac{1}{n} AD^{-1}e\right) \\ &= \ -\left[AD^{-2}A^T\left\{I - \gamma(AD^{-2}A^T)^{-1}\left(-\frac{2}{f_0^2} w^k {w^k}^T - \frac{1}{f_0} I\right)\right\}\right]^{-1} \times \\ &\quad \gamma\left(-\frac{1}{f_0} w^k + \frac{1}{n} AD^{-1}e\right)\end{aligned}$$

$$\begin{aligned} &= -\gamma \left[I + \gamma (AD^{-2}A^T)^{-1} \left(\frac{2}{f_0^2} w^k {w^k}^T + \frac{1}{f_0} I \right) \right]^{-1} (AD^{-2}A^T)^{-1} \times \\ &\quad \left(-\frac{1}{f_0} w^k + \frac{1}{n} AD^{-1} e \right) \\ &= \gamma \left[I + \gamma (AD^{-2}A^T)^{-1} \left(\frac{2}{f_0^2} w^k {w^k}^T + \frac{1}{f_0} I \right) \right]^{-1} \times \\ &\quad (AD^{-2}A^T)^{-1} (-h). \end{aligned} \tag{84}$$

Let $\gamma = \epsilon > 0$ and consider $\lim_{\epsilon \to 0^+} h^T \Delta w^*$. Since

$$\lim_{\epsilon \to 0^+} \Delta w^* = \epsilon \, (AD^{-2}A^T)^{-1}(-h)$$

then

$$\lim_{\epsilon \to 0^+} h^T \Delta w^* = -\epsilon \, h^T (AD^{-2}A^T)^{-1} h.$$

Since, by assumption, $\epsilon > 0$ and $h^T(AD^{-2}A^T)^{-1}h > 0$ then

$$\lim_{\epsilon \to 0^+} h^T \Delta w^* < 0,$$

showing that there exists $\gamma > 0$ such that the direction Δw^*, given in (82), is a descent direction of $\varphi(w)$.

The idea of the algorithm is to solve (77–78), more than once if necessary, with the radius r as a variable. Parameter γ is varied until r takes a value in some given interval. Each iteration of this algorithm is comprised of two tasks. To simplify notation, let

$$H_c = AD^{-2}A^T \tag{85}$$

$$H_o = -\frac{2}{f_0^2} w^k {w^k}^T - \frac{1}{f_0} I \tag{86}$$

and define

$$M = H_c + \gamma H_o.$$

Given the current iterate w^k, we first seek a value of γ such that $M \Delta w = \gamma h$ has a solution Δw^*. This can be done by binary search, as we will see shortly. Once such a parameter γ is found, the linear system

$$M \Delta w^* = \gamma h \tag{87}$$

is solved for $\Delta w^* \equiv \Delta w^*(\gamma(r))$. As was shown previously, the length $l(\Delta w^*(\gamma))$ is a monotonically increasing function of γ in the interval $0 \le \gamma \le \gamma_{max}$. Optimality condition (80) implies that $r = \sqrt{l(\Delta w^*(\gamma))}$ if $\mu > 0$. Small lengths result in small changes in the potential function, since r is small and the optimal solution lies on the surface of the ellipsoid. A length that is too large may not correspond to an optimal solution of (77–78), since this may require $r > 1$. An

```
procedure descent_direction($\gamma, w^k, \underline{l}, \overline{l}$)
1    $l = \infty$; $LD_{key} = \textbf{false}$; $\overline{\gamma}_{key} = \textbf{false}$; $\underline{\gamma}_{key} = \textbf{false}$;
2    do $l > \overline{l}$ or ($l < \underline{l}$ and $LD_{key} = \textbf{false}$) $\rightarrow$
3        $M = H_c + \gamma H_o$; $b = \gamma h$;
4        do $M \Delta w = b$ has no solution $\rightarrow$
5            $\gamma = \gamma / \gamma_r$; $LD_{key} = \textbf{true}$;
6            $M = H_c + \gamma H_o$; $b = \gamma h$;
7        od;
8        $\Delta w^* = M^{-1} b$; $l = (\Delta w^*)^T A D^{-2} A^T \Delta w^*$;
9        if $l < \underline{l}$ and $LD_{key} = \textbf{false} \rightarrow$
10           $\underline{\gamma} = \gamma$; $\underline{\gamma}_{key} = \textbf{true}$;
11           if $\overline{\gamma}_{key} = \textbf{true} \rightarrow \gamma = \sqrt{\underline{\gamma}\overline{\gamma}}$ fi;
12           if $\overline{\gamma}_{key} = \textbf{false} \rightarrow \gamma = \gamma \cdot \gamma_r$ fi;
13       fi;
14       if $l > \overline{l} \rightarrow$
15           $\overline{\gamma} = \gamma$; $\overline{\gamma}_{key} = \textbf{true}$;
16           if $\underline{\gamma}_{key} = \textbf{true} \rightarrow \gamma = \sqrt{\underline{\gamma}\overline{\gamma}}$ fi;
17           if $\underline{\gamma}_{key} = \textbf{false} \rightarrow \gamma = \gamma / \gamma_r$ fi;
18       fi;
19   od;
20   do $l < \underline{l}$ and $LD_{key} = \textbf{true} \rightarrow \underline{l} = \underline{l} / l_r$ od;
21   return($\Delta w^*$);
end descent_direction;
```

Figure 12: Procedure `descent_direction`: Algorithm to compute descent direction in nonconvex potential function minimization

interval $(\underline{l}, \overline{l})$ called the *acceptable length region*, is defined such that a length $l(\Delta w^*(\gamma))$ is accepted if $\underline{l} \leq l(\Delta w^*(\gamma)) \leq \overline{l}$. If $l(\Delta w^*(\gamma)) < \underline{l}$, γ is increased and (87) is resolved with the new M matrix and h vector. On the other hand, if $l(\Delta w^*(\gamma)) > \overline{l}$, γ is reduced and (87) is resolved. Once an acceptable length is produced we use $\Delta w^*(\gamma)$ as the descent direction.

Figure 12 presents pseudo-code for procedure `descent_direction`, where (77–78) is optimized. As input, procedure `descent_direction` is given an estimate for parameter γ, the current iterate w^k around which the inscribing ellipsoid is to be constructed and the current acceptable length region defined by $\underline{l}$ and $\overline{l}$. The value of γ passed to `descent_direction` at minor iteration k of `cmq` is the value returned by `descent_direction` at minor iteration $k-1$. It returns a descent direction Δw^* of the quadratic approximation of the potential function $Q(w)$ from w^k, the next estimate for parameter γ and the current lower bound of the acceptable length region $\underline{l}$.

In line 1, the length l is set to a large number and several logical keys are initialized: LD_{key} is **true** if a linear dependency in the rows of M is ever found during the solution of the linear system (87) and is **false** otherwise; $\overline{\gamma}_{key}$ ($\underline{\gamma}_{key}$) is **true** if an upper (lower) bound for an acceptable γ has been found and **false** otherwise.

The problem of minimizing a nonconvex quadratic function over an ellipsoid is carried out in the loop going from line 2 to 19. The loop is repeated until either a length l is found such that $\underline{l} \le l \le \overline{l}$ or $l \le \underline{l}$ due to a linear dependency found during the solution of (87), i.e. if $LD_{key} = \textbf{true}$. Lines 3 to 8 produce a descent direction that may not necessarily have an acceptable length. In line 3 the matrix M and the right hand side vector b are formed. The linear system (87) is tentatively solved in line 4. The solution procedure may not be successful, i.e. M may be singular. This implies that parameter γ is too large and parameter γ is reduced in line 5 of loop 4–7, which is repeated until a nonsingular matrix M is produced.

Once a nonsingular M matrix is available, a descent direction Δw^* is computed in line 8 along with its corresponding length l. Three cases can occur: (i) - the length is too small even though no linear dependency was detected in the factorization; (ii) - the length is too large; or (iii) - the length is acceptable. Case (iii) is the termination condition for the main loop 2-19. In lines 9-13 the first case is considered. The value of γ is a lower bound on an acceptable value of γ and is recorded in line 10 and the corresponding logical key is set. If an upper bound $\overline{\gamma}$ for an acceptable value of γ has been found the new estimate for γ is set to the geometric mean of $\underline{\gamma}$ and $\overline{\gamma}$ in line 11. Otherwise γ is increased by a fixed factor in line 12.

Similar to the treatment of case (i), case (ii) is handled in lines 14-18. The current value of γ is an upper bound on an acceptable value of γ and is recorded in line 15 and the corresponding logical key is set. If a lower bound $\underline{\gamma}$ for an acceptable value of γ has been found the new estimate for γ is set to the geometric mean of $\underline{\gamma}$ and $\overline{\gamma}$ in line 16. Otherwise γ is decreased by a fixed factor in line 17.

Finally, in line 20, the lower bound $\underline{l}$ may have to be adjusted if $l < \underline{l}$ and $LD_{key} = \textbf{true}$. Note that the key LD_{key} is used only to allow the adjustment in the range of the acceptable length, so that the range returned contains the current length l.

7.2 Some computational considerations

The density of the linear system solved at each iteration of `descent_direction` is determined by the density of the Hessian matrix. Using the potential function described in the previous section, this Hessian,

$$M = AD^{-2}A^T - \frac{2}{f_0^2} w^k {w^k}^T - \frac{1}{f_0} I,$$

is totally dense, because of the rank one component $\frac{2}{f_0^2} w^k {w^k}^T$. Consequently, direct factorization solution techniques must be ruled out for large instances. However, in the case where the matrix A is sparse, iterative methods can be applied to approximately solve the linear system. In [71], a preconditioned conjugate gradient algorithm, using diagonal preconditioning, was used to solve the system efficiently taking advantage of the special structure of the coefficient matrix. In this approach, the main computational effort is the multiplication of a dense vector ξ and the coefficient matrix M, i.e. $M\xi$. This multiplication can be done efficiently, by considering fact that M is the sum of three matrices, each of which has special structure. The first multiplication,

$$\frac{1}{f_0} I \xi$$

is simply a scaling of ξ. The second product,

$$\frac{2}{f_0^2} w^k {w^k}^T \xi$$

is done in two steps. First, an inner product ${w^k}^T \xi$ is computed. Then, the vector $\frac{4}{f_0^2} w^k$ is scaled by the inner product. The third product,

$$AD^{-2}A^T \xi$$

is done in three steps. First the product $A^T \xi$ is carried out. The resulting vector is scaled by D^{-2} and multiplies A. Therefore, if A is sparse, the entire matrix vector multiplication can be done efficiently.

In a recent study, Warners et al. [142] describe a new potential function

$$\phi_\rho(w) = m - w^T w - \sum_{i=1}^{n} \rho_i \log d_i(w),$$

whose gradient and Hessian are given by

$$h = -2w + AD^{-1}\rho,$$

and

$$H = -2I + AD^{-1}PD^{-1}A^T,$$

where $\rho = (\rho_1, \ldots, \rho_n)$ and $P = \text{diag}(\rho)$. Note that the density of the Hessian depends only on the density of AA^T. Consequently, direct factorization methods can be used efficiently when the density of AA^T is small.

7.3 Application to combinatorial optimization

The algorithms discussed in this section have been applied to the following integer programming problem: Given $A' \in \mathsf{R}^{m \times n'}$ and $b' \in \mathsf{R}^{n'}$, find $w \in \mathsf{R}^m$ such that:

$$\begin{aligned} A'^T w &\leq b' && (88) \\ w_i &= \{-1, 1\}, \; i = 1, \ldots, m. && (89) \end{aligned}$$

The more common form of integer programming, where variables x_i take on (0,1) values, can be converted to the above form with the change of variables

$$x_i = \frac{1 + w_i}{2}, \; i = 1, \ldots, m.$$

More specifically, let I denote an $m \times m$ identity matrix,

$$A = \left[A' \, \vdots \, I \, \vdots \, -I \right] \in \mathsf{R}^{m \times n}$$

and

$$b = \begin{bmatrix} b' \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathsf{R}^n$$

and let

$$\mathcal{I} = \left\{ w \in \mathsf{R}^m \mid A^T w \leq b \text{ and } w_i = \{-1, 1\} \right\}.$$

With this notation, we can state the integer programming problem as: Find $w \in \mathcal{I}$.

As before, let

$$\mathcal{L} = \left\{ w \in \mathsf{R}^m \mid A^T w \leq b \right\}$$

and consider the linear programming relaxation of (88–89), i.e. find $w \in \mathcal{L}$. One way of selecting ± 1 integer solutions over fractional solutions in linear programming is to introduce the quadratic objective function,

$$\max \; w^T w = \sum_{i=1}^{m} w_i^2$$

and solve the nonconvex quadratic programming problem (62–63). Note that $w^T w \leq m$, with the equality only occurring when $w_j = \pm 1, \; j = 1, \ldots, m$. Furthermore, if $w \in \mathcal{I}$ then $w \in \mathcal{L}$ and $w_i = \pm 1, \; i = 1, \ldots, m$ and therefore $w^T w = m$. Hence, if w is the optimal solution to (62–63) then $w \in \mathcal{L}$. If $w^T w = m$ then $w_i = \pm 1, \; i = 1, \ldots, m$ and therefore $w \in \mathcal{I}$. Consequently, this shows that if $w \in \mathcal{L}$ then $w \in \mathcal{I}$ if and only if $w^T w = m$.

In place of (62–63), one solves the nonconvex potential function minimization

$$\min \ \{\varphi(w) \mid A^T w \leq b\}, \tag{90}$$

where $\varphi(w)$ is given by (65–67). The generally applied scheme rounds each iterate to an integer solution, terminating if a feasible integer solution is produced. If the algorithm converges to a nonglobal local minimum of (90), then the problem is modified by adding a cut and the algorithm is applied to the augmented problem. Let v be the integer solution rounded off from the local minimum. A valid cut is

$$v^T w \leq m - 2. \tag{91}$$

Observe that if $w = v$ then $v^T w = m$. Otherwise, $v^T w \leq m - 2$. Therefore, the cut (91) excludes v but does not exclude any other feasible integral solution of (88–89).

We note that adding a cut of the type above will not, theoretically, prevent the algorithm from converging to the same local minimum twice. In practice [77], the addition of the cut changes the objective function, consequently altering the trajectory followed by the algorithm.

Most combinatorial optimization problems have very natural equivalent integer and quadratic programming formulations [113]. The algorithms described in this section have been applied to a variety of problems, including maximum independent set [78], set covering [77], satisfiability [71, 134], inductive inference [69, 70], and frequency assignment in cellular telephone systems [143].

8 A lower bounding technique

A lower bound for the globally optimal solution of the quadratic program

$$\min \ q(x) = \frac{1}{2} x^T Q x + c^T x \tag{92}$$

subject to

$$x \in \mathcal{P} = \{x \in R^n \mid Ax = b, \ x \geq 0\}, \tag{93}$$

where $Q \in \mathsf{R}^{n \times n}$, $A \in \mathsf{R}^{m \times n}$, $c \in \mathsf{R}^n$, and $b \in \mathsf{R}^m$, can be obtained by minimizing the objective function over the largest ellipsoid inscribed in $\mathcal{P}$. This technique can be applied to quadratic integer programming, a problem that is NP-hard in the general case. Kamath and Karmarkar [66] proposed a polynomial time interior point algorithm for computing these bounds. This is one of the first computational approaches to solve semidefinite programming relaxations. The problem is solved as a minimization of the trace of a matrix subject to positive definiteness conditions. The algorithm takes no more than $O(nL)$ iterations (where L is the the number of bits required to represent the input). The algorithm does two matrix inversions per iteration.

Consider the quadratic integer program

$$\min\ f(x) = x^T Q x \tag{94}$$

subject to

$$x \in S = \{-1, 1\}^n, \tag{95}$$

where $Q \in \mathsf{R}^{n \times n}$ is symmetric. Let f_{min} be the value of the optimal solution of (94–95).

Consider the problem of finding good lower bounds on f_{min}. To apply an interior point method to this problem, one needs to embed the discrete set S in a continuous set $T \supseteq S$. Clearly, the minimum of $f(x)$ over T is a lower bound on f_{min}.

A commonly used approach is to choose the continuous set to be the box

$$B = \{x \in \mathsf{R}^n \mid -1 \leq x_i \leq 1, i = 1, \ldots, n\}.$$

However, if $f(x)$ is not convex, the problem of minimizing $f(x)$ over B is NP-hard. Consider this difficult case, and therefore assume that Q has at least one negative eigenvalue. Since optimizing over a box can be hard, instead enclose the box in an ellipsoid E. Let

$$U = \{w = (w_1, \ldots, w_n) \in \mathsf{R}^n \ \mid \ \textstyle\sum_{i=1}^n w_i = 1 \text{ and } w_i > 0,\ i = 1, \ldots, n, \},$$

and consider the parameterized ellipsoid

$$E(w) = \{x \in \mathsf{R}^n \mid x^T W x \leq 1\},$$

where $w \in U$ and $W = \text{diag}(w)$.

Clearly, the set S is contained in $E(w)$. If $\lambda_{min}(w)$ is the minimum eigenvalue of $W^{-1/2} Q W^{-1/2}$, then

$$\min \frac{x^T Q x}{x^T W x} = \min \frac{x^T W^{-1/2} Q W^{-1/2} x}{x^T x} = \lambda_{min}(w),$$

and therefore

$$x^T Q x \geq \lambda_{min}(w)\,,\ \forall x \in E(w).$$

Hence, the minimum value of $f(x)$ over $E(w)$ can be obtained by simply computing the minimum eigenvalue of $W^{-1/2} Q W^{-1/2}$. To further improve the bound on f_{min} requires that $\lambda_{min}(w)$ be maximized over the set U. Therefore, the problem of finding a better lower bound is transformed into the optimization problem

$$\max\ \mu$$

subject to

$$\frac{x^T Q x}{x^T W x} \geq \mu\,,\ \forall x \in \mathsf{R}^n \setminus \{0\} \ \text{ and } \ w \in U.$$

One can further simplify the problem by defining $d = (d_1, \ldots, d_n) \in \mathsf{R}^n$ such that $\sum_{i=1}^n d_i = 0$. Let $D = \text{diag}(d)$. If

$$\frac{x^T(Q-D)x}{x^TWx} \geq \mu,$$

then, since $\sum_{i=1}^n w_i = 1$ and $\sum_{i=1}^n d_i = 0$,

$$x^TQx \geq \mu x^TWx + x^TDx = \mu,$$

for $x \in S$. Now, define $z = \mu w + d$ and let $Z = \text{diag}(z)$. For all $x \in S$,

$$x^TZx = e^Tz = \mu,$$

and therefore the problem becomes

$$\max\ e^Tz$$

subject to

$$x^T(Q-Z)x \geq 0.$$

Let $M(z) = Q - Z$. Observe that solving the above problem amounts to minimizing the trace of $M(z)$ while keeping $M(z)$ positive semidefinite. Since $M(z)$ is real and symmetric, it has n real eigenvalues $\lambda_i(M(z)), i = 1, \ldots, n$. To ensure positive definiteness, the eigenvalues of $M(z)$ must be nonnegative. Hence, the above problem is reformulated as

$$\min\ \text{tr}(M(z))$$

subject to

$$\lambda_i(M(z)) \geq 0\,,\ i = 1, \ldots, n.$$

Kamath and Karmarkar [66, 67] proposed an interior point approach to solve the above trace minimization problem, that takes no more that $O(nL)$ iterations, having two matrix inversions per iteration. Figure 13 shows a pseudo code for this algorithm.

To analyze the algorithm, consider the parametric family of potential functions given by

$$g(z, v) = 2n\ln(\text{tr}(M(z)) - v) - \ln\det(M(z)),$$

where $v \in \mathsf{R}$ is a parameter. This algorithm will generate a monotonically increasing sequence of parameters $v^{(k)}$ that converges to the optimal value v^*. The sequence $v^{(k)}$ is constructed together with the sequence $z^{(k)}$ of interior points, as shown in the pseudo code in Figure 13. Since $Q - Z^*$ is a positive definite matrix, $v^{(0)} = 0 \leq v^*$ is used as the initial point in the sequence.

```
procedure qplb($Q, \epsilon, z$, opt)
1     $z^{(0)} = (\lambda_{min}(Q) - 1)e$;
2     $v^{(0)} = 0$;
3     $M(z^{(0)}) = Q - Z^{(0)}$; $k = 0$;
4     do $\mathrm{tr}(M(z^{(k)})) - v^{(k)}) \geq \epsilon \rightarrow$
5         Construct $H^{(k)}$ where $H_{ij}^{(k)} = (e_i^T M(z^{(k)})^{-1} e_j)^2$;
6         $f^{(k)}(z) = 2n \ln(\mathrm{tr}(M(z^{(k)})) - v^{(k)}) - \ln \det M(z^{(k)})$;
7         $g^{(k)} = \nabla f^{(k)}(z^{(k)})$;
8         $\beta = 0.5/\sqrt{g^{(k)T} H^{(k)-1} g^{(k)}}$;
9         Solve $H^{(k)} \Delta z = -\beta g^{(k)}$;
10        if $g^{(k)T} \Delta z < 0.5 \rightarrow$
11            Increase $v^{(k)}$ until $g^{(k)T} \Delta z = 0.5$;
12        fi;
13        $z^{(k+1)} = z^{(k)} + \Delta z$; $k = k + 1$;
14    od;
15    $z = z^{(k)}$; opt $= \mathrm{tr}(Q) - v^{(k)}$;
end qplb;
```

Figure 13: Procedure `qplb`: Interior point algorithm for computing lower bounds

Let $g_1^{(k)}(z, v)$ be the linear approximation of $g(z, v)$ at $z^{(k)}$. Then

$$g_1^{(k)}(z, v) = -\frac{2n}{\mathrm{tr}(M(z^{(k)}) - v} e^T z + \nabla \ln \det(M(z^{(k)})^T z + C,$$

where C is a constant. Kamath and Karmarkar show how $g_1^{(k)}(z, v)$ can be reduced by a constant amount at each iteration. They prove that it is possible to compute $v^{(k+1)} \in \mathsf{R}$ and a point $z^{(k+1)}$ in a closed ball of radius α centered at $z^{(k)}$ such that $v^{(k)} \leq v^{(k+1)} \leq v^*$ and

$$g_1^{(k)}(z^{(k+1)}, v^{(k+1)}) - g_1^{(k)}(z^{(k)}, v^{(k+1)}) \leq -\alpha.$$

Using this fact, they show that, if $z^{(k)}$ is the current interior point and $v^{(k)} \leq v^*$ is the current estimate of the optimal value, then

$$g(z^{(k+1)}, v^{(k+1)}) - g(z^{(k)}, v^{(k)}) \leq -\alpha + \frac{\alpha^2}{2(1-\alpha)},$$

where $z^{(k+1)}$ and $v^{(k+1)}$ are the new interior point and new estimate, respectively. This proves polynomial-time complexity for the algorithm.

9 Semidefinite Programming Relaxations

There has been a great deal of interest recently in solving semidefinite programming relaxations of combinatorial optimization problems [5, 6, 40, 144, 145, 152, 54, 53, 57, 112, 123, 111]. The semidefinite relaxations are solved by an interior point approach. These papers have shown the strength of the relaxations, and some of these papers have discussed cutting plane and branch and cut approaches using these relaxations. The bounds obtained from semidefinite relaxations are often better than those obtained using linear programming relaxations, but they are also usually more expensive to compute.

Semidefinite programming relaxations of some integer programming problems have proven to be very powerful, and they can often provide better bounds than those given by linear programming relaxations. There has been interest in semidefinite programming relaxations since at least the seventies (see Lovász [89]). These were regarded as being purely of theoretical interest until the recent development of interior point methods for semidefinite programming problems [6, 106, 56, 107, 82, 139]. Interest was increased further by Goemans and Williamson [40], who showed that the bounds generated by semidefinite programming relaxations for the maximum cut and satisfiability problems were considerably better than those that could be obtained from a linear programming relaxation, in the worst case, and that the solutions to these relaxations could be exploited to generate good integer solutions.

For an example of a semidefinite programming relaxation, consider the quadratic integer programming problem

$$\min \; f(x) = x^T Q x$$

subject to

$$x \in S = \{-1, 1\}^n,$$

where $Q \in \mathbb{R}^{n \times n}$ is symmetric, first discussed in equations (94) and (95) in section 8.

We let trace(M) denote the trace of a square matrix. By exploiting the fact that trace(AB)=trace(BA), we can rewrite the product $x^T Q x$:

$$x^T Q x = \text{trace}(x^T Q x) = \text{trace}(Q x x^T)$$

This lets us reformulate the quadratic program as

$$\begin{array}{ll} \min & \text{trace}(QX) \\ \text{subject to} & X = xx^T \\ & X_{ii} = 1 \qquad i = 1, \ldots, n. \end{array}$$

The constraint that $X = xx^T$ is equivalent to saying that X must have rank equal to one. This is a hard constraint to enforce, so it is relaxed to the constraint that X is positive semi-definite, written $X \succeq 0$. This gives the following semidefinite programming relaxation:

$$\begin{array}{lll} \min & \operatorname{trace}(QX) & \\ \text{subject to} & X_{ii} = 1 & i = 1, \ldots, n \\ & X \succeq 0. & \end{array}$$

Once we have a good relaxation, we can then (in principle) use a branch and bound (or branch and cut) method to solve the problem to optimality. Helmberg et al. [53] showed that the semidefinite programming relaxations of general constrained $0 - 1$ quadratic programming problems could be strengthened by using valid inequalities of the cut-polytope. There are a large number of such inequalities, and in [54], a branch and cut approach using semidefinite relaxations is used to solve quadratic $\{-1, 1\}$ problems with dense cost matrices. They branch using the criterion that two variables either take the same value or they take opposite values. This splits the current SDP relaxation into two SDP subproblems, each corresponding to quadratic $\{-1, 1\}$ problems of dimension one less. They are able to solve problems with up to about 100 variables in a reasonable amount of time.

Helmberg et al. [57] contains a nice discussion of different families of constraints for semidefinite relaxations of the quadratic knapsack problem. They derive semidefinite constraints from both the objective function and from the knapsack constraint. Many of the semidefinite constraints derived from the objective function are manipulations of the linear constraints for the Boolean quadric polytope. Similarly, they derive semidefinite constraints from known facets of the knapsack polytope. They attempt to determine the relative importance of different families of constraints.

The bottleneck with the branch and cut approach is the time required to solve each relaxation, and in particular to calculate the interior point directions. One way to reduce this time is to fix variables at -1 or 1, in much the same way that variables with large reduced costs can be fixed when we use a branch and cut algorithm that solves linear programming relaxations at each node. Helmberg [52] has proposed a method to determine whether a variable can be fixed when solving an SDP relaxation. This method examines the dual to the SDP relaxation. If it appears that a variable should be fixed at 1, say, then the effect of adding an explicit constraint that the variable should take the value -1 is examined. The change in the dual value that would result is then bounded; if this change is large enough then the variable can be fixed at 1.

The papers [54, 53, 57, 18] all contain semidefinite relaxations of quadratic programming problems with at most one constraint. By contrast, Wolkowicz and Zhao [144, 145] and Zhao et al. [152] have looked at semidefinite relaxations of more complicated integer programming problems. This required the development of some techniques that appear to be widely applicable. For these problems, the primal semidefinite programming relaxation does not have an interior feasible point, that is, there is no positive definite matrix that satisfies all the constraints. This implies that the dual problem will have an unbounded optimal face, so the problem is computationally intractable for an interior point method. To overcome this difficulty, the authors recast the problem in a lower dimensional space, where the barycenter of the known integer solutions corresponds to an interior point. In particular, if X is the matrix of variables for the original semidefinite formulation, a constant matrix V is determined so that the problem can be recast in terms of a matrix Z of variables, with $X = VZV^T$ and Z is of smaller dimension than X. To ensure that the new problem corresponds to the original problem, a *gangster operator* is used, which forces some components of VZV^T to be zero. With this reformulation, an interior point method can be used successfully to solve the semidefinite relaxations. An extension of the gangster operator may make it possible to use these relaxations in a branch and cut approach.

Another interesting aspect of [145] is the development of an alternative, slightly weaker, semidefinite relaxation that allows the exploitation of some sparsity in the original matrix for the set covering problem. The resulting relaxation contains both semidefinite constraints and linear constraints. This may make the semidefinite approach viable for problems of this type which are far larger than those previously tackled with semidefinite programming approaches. Whether this approach can be extended to other problems is an interesting question.

Some work on attempting to exploit sparsity in the general setting has been performed by Fujisawa et al. [38], and by Helmberg et al. [56] in their MATLAB implementation. Zhao et al. [152, 151] propose using a preconditioned conjugate gradient method to calculate the directions for the quadratic assignment problem (QAP) within a primal-infeasible dual-feasible variant of the method proposed in [56]. In the setting of solving a QAP, the semidefinite relaxation is used to obtain a lower bound on the optimal value; this bound is provided by the dual solution. Thus, only dual feasibility is needed to get a lower bound, and so primal feasibility is not as important, and it is possible to solve the Newton system of equations only approximately while still maintaining dual feasibility. It should be possible to extend this approach to other problems.

Zhao et al. [152] also developed another relaxation for the QAP which contains a large number of constraints. This second relaxation is stronger than the relaxation that uses the gangster operator, but because of the number of constraints, they could only use it in a cutting plane algorithm. Due to memory limitations, the gangster approach provided a better lower bound than the other relaxation for larger problems.

One way to solve sparse problems using semidefinite programming techniques is to look at the dual problem. Benson et al. [16] and Helmberg and Rendl [55] have both recently proposed methods that obtain very good bounds and sometimes optimal solutions for sparse combinatorial optimization problems by looking at the dual problem or relaxations of the dual.

There are several freely available implementations of SDP methods. Many of these codes are written in MATLAB. One of the major costs in an iteration of an SDP algorithm is constructing the Newton system of equations, with a series of `for` loops. MATLAB does not appear to handle this well, because of the slowness of its interpreted loops: in compiled C code, each iteration of these loops takes half a dozen machine language instructions, while in the interpreted code, each pass through one of these loops takes 100 or more instructions. For details of a freely available C implementation, see [17].

10 Concluding Remarks

Optimization is of central importance in both the natural sciences, such as physics, chemistry and biology, as well as artificial or man-made sciences, such as computer science and operations research. Nature inherently seeks optimal solutions. For instance, crystalline structure is the minimum energy state for a set of atoms, and light travels through the shortest path. The behavior of nature can often be explained on the basis of variational principles. Laws of nature then simply become optimality conditions. Concepts from continuous mathematics have always played a central role in the description of these optimality conditions and in analysis of the structure of their solutions. On the other hand, in artificial sciences, the problems are stated using the language of discrete mathematics or logic, and a simple-minded search for their solution confines one to a discrete solution set.

With the advent of interior point methods, the picture is changing, because these methods do not confine their working to a discrete solution set, but instead view combinatorial objects as limiting cases of continuous objects and exploit the topological and geometric properties of the continuous space. As a result, the number of efficiently solvable combinatorial problems is expanding. Also, the interior-point methods have revealed that the mathematical structure relevant to optimization in the natural and artificial sciences have a great deal in common. Recent conferences on global optimization (e.g. [35, 36]) are attracting researchers from diverse fields, ranging from computer science to molecular biology, thus merging the development paths of natural and artificial sciences. The phenomena of multiple solutions to combinatorial problems is intimately related to multiple configurations a complex molecule can assume. Thus, understanding the structure of solution sets of nonlinear problems is a common challenge faced by both natural and artificial sciences, to explain natural phenomena in the former case and to create more efficient interior-point algorithms in the latter case.

In the last decade we have witnessed computational breakthroughs in the approximate solution of large scale combinatorial optimization problems. Many of these breakthroughs are due to the development of interior point algorithms and implementations. Starting with linear programming in 1984 [72], these developments have spanned a wide range of problems, including network flows, graph problems, and integer programming. There is a continuing activity with new papers and codes being announced almost on daily basis. The interested reader can consult the following web sites:

1. http://www.mcs.anl.gov:80/home/otc/InteriorPoint
   is an archive of technical reports and papers on interior-point methods, maintained by S.J. Wright at Argonne National Laboratory.
2. http://www.zib.de/helmberg/semidef.html
   contains a special home page for semidefinite programming organized by C. Helmberg, at Berlin Center for Scientific Computing, Konrad Zuse Zentrum fur Informationstechnik, Berlin.
3. ftp://orion.uwaterloo.ca/pub/henry/reports/psd.bib.gz
   contains a bib file with papers related to SDP.

Acknowledgement

The first author acknowledges support in part by ONR grant N00014–94–1–0391, and by a grant from the Dutch NWO and by Delft University of Technology for 1997–98, while visiting TWI/SSOR at Delft University of Technology. The second author acknowledges support in part by NSF grant BIR-9505913 and U.S. Department of Air Force grant F08635-92-C-0032.

References

[1] W.P. Adams and T.A. Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 43–75. American Mathematical Society, 1994.

[2] I. Adler, N. Karmarkar, M.G.C. Resende, and G. Veiga. Data structures and programming techniques for the implementation of Karmarkar's algorithm. *ORSA Journal on Computing*, 1:84–106, 1989.

[3] I. Adler, N. Karmarkar, M.G.C. Resende, and G. Veiga. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44:297–335, 1989.

[4] Navindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[5] F. Alizadeh. Optimization over positive semi-definite cone: Interior-point methods and combinatorial applications. In P.M. Pardalos, editor,

Advances in Optimization and Parallel Computing, pages 1–25. North–Holland, Amsterdam, 1992.

[6] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[7] E. D. Andersen, J. Gondzio, C. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior Point Methods in Mathematical Programming*, chapter 6. Kluwer Academic Publishers, 1996.

[8] D. S. Atkinson and P. M. Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69:1–43, 1995.

[9] O. Bahn, O. Du Merle, J. L. Goffin, and J. P. Vial. A cutting plane method from analytic centers for stochastic programming. *Mathematical Programming*, 69:45–73, 1995.

[10] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993.

[11] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[12] E.R. Barnes. A variation on Karmarkar's algorithm for solving linear programming problems. *Mathematical Programming*, 36:174–182, 1986.

[13] D. A. Bayer and J. C. Lagarias. The nonlinear geometry of linear programming, I. Affine and projective scaling trajectories. *Transactions of the American Mathematical Society*, 314:499–526, 1989.

[14] D. A. Bayer and J. C. Lagarias. The nonlinear geometry of linear programming, II. Legendre transform coordinates and central trajectories. *Transactions of the American Mathematical Society*, 314:527–581, 1989.

[15] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear Programming and Network Flows.* Wiley, New York, NY, 1990.

[16] S. J. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. Technical report, Department of Management Sciences, University of Iowa, Iowa City, Iowa 52242, September 1997.

[17] B. Borchers. CSDP, a C library for semidefinite programming. Technical report, Mathematics Department, New Mexico Tech, Socorro, NM 87801, March 1997.

[18] B. Borchers, S. Joy, and J. E. Mitchell. Three methods to the exact solution of max-sat problems. Talk given at *INFORMS Conference, Atlanta, 1996.* Slides available from the URL $http://www.nmt.edu/\tilde{}borchers/atlslides.ps$, 1996.

[19] B. Borchers and J. E. Mitchell. Using an interior point method in a branch and bound algorithm for integer programming. Technical Report 195, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, March 1991. Revised July 7, 1992.

[20] C. Chevalley. *Theory of Lie Groups.* Princeton University Press, Princeton, New Jersey, 1946.

[21] T. Christof and G. Reinelt. Parallel cutting plane generation for the tsp (extended abstract). Technical report, IWR Heidelberg, Germany, 1995.

[22] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tj. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. Wiley, New York, 1951.

[23] G.B. Dantzig. Application of the simplex method to a transportation problem. In T.C. Koopsmans, editor, *Activity Analysis of Production and Allocation.* John Wiley and Sons, 1951.

[24] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[25] A. de Silva and D. Abramson. A parallel interior point method and its application to facility location problems. Technical report, School of Computing and Information Technology, Griffith University, Nathan, QLD 4111, Australia, 1995.

[26] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. *Journal of Statistical Physics*, 84:1363–1371, 1996.

[27] M. Deza and M. Laurent. Facets for the cut cone I. *Mathematical Programming*, 56:121–160, 1992.

[28] M. Deza and M. Laurent. Facets for the cut cone II: Clique-web inequalities. *Mathematical Programming*, 56:161–188, 1992.

[29] I. I. Dikin. Iterative solution of problems of linear and quadratic programming. *Doklady Akademiia Nauk SSSR*, 174:747–748, 1967. English Translation: *Soviet Mathematics Doklady*, 1967, Volume 8, pp. 674–675.

[30] D. Z. Du, J. Gu, and P. M. Pardalos, editors. *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1997.

[31] Ding-Zhu Du and Panos M. Pardalos, editors. *Network Optimization Problems: Algorithms, Applications and Complexity.* World Scientific, 1993.

[32] J. Edmonds. Maximum matching and a polyhedron with 0, 1 vertices. *Journal of Research National Bureau of Standards*, 69B:125–130, 1965.

[33] A. S. El-Bakry, R. A. Tapia, and Y. Zhang. A study of indicators for identifying zero variables in interior–point methods. *SIAM Review*, 36:45–72, 1994.

[34] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley and Sons, New York, 1968. Reprinted as Volume 4 of the SIAM Classics in Applied Mathematics Series, 1990.

[35] C. Floudas and P. Pardalos. *Recent Advances in Global Optimization.* Princeton Series in Computer Science. Princeton University Press, 1992.

[36] C. Floudas and P. Pardalos. *State of the Art in Global Optimization: Computational Methods and Applications.* Kluwer Academic Publishers, 1996.

[37] L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1990.

[38] K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. Technical report, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152, Japan, January 1997.

[39] A. George and M. Heath. Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra and Its Applications*, 34:69–83, 1980.

[40] Michel X. Goemans and David P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *J. Assoc. Comput. Mach.*, 42:1115–1145, 1995.

[41] J.-L. Goffin, J. Gondzio, R. Sarkissian, and J.-P. Vial. Solving nonlinear multicommodity network flow problems by the analytic center cutting plane method. *Mathematical Programming*, 76:131–154, 1997.

[42] J.-L. Goffin, A. Haurie, and J.-P. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, 38:284–302, 1992.

[43] J.-L. Goffin, Z.-Q. Luo, and Y. Ye. Further complexity analysis of a primal-dual column generation algorithm for convex or quasiconvex feasibility problems. Technical report, Faculty of Management, McGill University, Montréal, Québec, Canada, November 1993.

[44] J.-L. Goffin, Z.-Q. Luo, and Y. Ye. On the complexity of a column generation algorithm for convex or quasiconvex problems. In *Large Scale Optimization: The State of the Art.* Kluwer Academic Publishers, 1993.

[45] G.H. Golub and C.F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, 1983.

[46] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[47] M. Grötschel and O. Holland. Solution of large-scale travelling salesman problems. *Mathematical Programming*, 51(2):141–202, 1991.

[48] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32:1195–1220, 1984.

[49] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin, Germany, 1988.

[50] G.M. Guisewite. Network problems. In Reiner Horst and Panos M. Pardalos, editors, *Handbook of global optimization.* Kluwer Academic Publishers, 1994.

[51] G.M. Guisewite and P.M. Pardalos. Minimum concave cost network flow problems: Applications, complexity, and algorithms. *Annals of Operations Research*, 25:75–100, 1990.

[52] C. Helmberg. Fixing variables in semidefinite relaxations. Technical Report SC-96-43, Konrad-Zuse-Zentrum fuer Informationstechnik, Berlin, December 1996.

[53] C. Helmberg, S. Poljak, F. Rendl, and H. Wolkowicz. Combining semidefinite and polyhedral relaxations for integer programs. In E. Balas and J. Clausen, editors, *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science*, volume 920, pages 124–134. Springer, 1995.

[54] C. Helmberg and F. Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. Technical Report SC-95-35, Konrad-Zuse-Zentrum fuer Informationstechnik, Berlin, 1995.

[55] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. Technical Report SC-97-37, Konrad-Zuse-Zentrum fuer Informationstechnik, Berlin, August 1997. Revised: October 1997.

[56] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. *SIAM Journal on Optimization*, 6:342–361, 1996.

[57] C. Helmberg, F. Rendl, and R. Weismantel. Quadratic knapsack relaxations using cutting planes and semidefinite programming. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science*, volume 1084, pages 175–189. Springer, 1996.

[58] F.L. Hitchcock. The distribution of product from several sources to numerous facilities. *Journal of Mathematical Physics*, 20:224–230, 1941.

[59] K. L. Hoffman and M. Padberg. Improving LP-representation of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing*, 3(2):121–134, 1991.

[60] E. Housos, C. Huang, and L. Liu. Parallel algorithms for the AT&T KORBX System. *AT&T Technical Journal*, 68:37–47, 1989.

[61] D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[62] A. Joshi, A.S. Goldstein, and P.M. Vaidya. A fast implementation of a path-following algorithm for maximizing a linear function over a network polytope. In David S. Johnson and Catherine C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, volume 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 267–298. American Mathematical Society, 1993.

[63] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In *Combinatorial Optimization: DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 111–152. AMS, 1995.

[64] J.A. Kaliski and Y. Ye. A decomposition variant of the potential reduction algorithm for linear programming. *Management Science*, 39:757–776, 1993.

[65] A. P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. Computational experience with an interior point algorithm on the Satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.

[66] A.P. Kamath and N. Karmarkar. A continuous method for computing bounds in integer quadratic optimization problems. *Journal of Global Optimization*, 2:229–241, 1992.

[67] A.P. Kamath and N. Karmarkar. An $O(nL)$ iteration algorithm for computing bounds in quadratic optimization problems. In P.M. Pardalos, editor, *Complexity in Numerical Optimization*, pages 254–268. World Scientific, Singapore, 1993.

[68] A.P. Kamath, N. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.

[69] A.P. Kamath, N. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.

[70] A.P. Kamath, N. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. An interior point approach to Boolean vector function synthesis. In *Proceedings of the 36th MSCAS*, pages 185–189, 1993.

[71] A.P. Kamath, N. Karmarkar, N. Ramakrishnan, and M.G.C. Resende. Computational experience with an interior point algorithm on the Satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.

[72] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[73] N. Karmarkar. An interior-point approach for NP-complete problems. *Contemporary Mathematics*, 114:297–308, 1990.

[74] N. Karmarkar. An interior-point approach to NP-complete problems. In *Proceedings of the First Integer Programming and Combinatorial Optimization Conference*, pages 351–366, University of Waterloo, 1990.

[75] N. Karmarkar. A new parallel architecture for sparse matrix computation based on finite projective geometries. In *Proceedings of Supercomputing '91*, pages 358–369. IEEE Computer Society, 1991.

[76] N. Karmarkar, J. Lagarias, L. Slutsman, and P. Wang. Power series variants of Karmarkar-type algorithms. *AT&T Technical Journal*, 68:20–36, 1989.

[77] N. Karmarkar, M.G.C. Resende, and K. Ramakrishnan. An interior point algorithm to solve computationally difficult set covering problems. *Mathematical Programming*, 52:597–618, 1991.

[78] N. Karmarkar, M.G.C. Resende, and K.G. Ramakrishnan. An interior point approach to the maximum independent set problem in dense random graphs. In *Proceedings of the XIII Latin American Conference on Informatics*, volume 1, pages 241–260, Santiago, Chile, July 1989.

[79] N. K. Karmarkar and K. G. Ramakrishnan. Computational results of an interior point algorithm for large scale linear programming. *Mathematical Programming*, 52:555–586, 1991.

[80] J.L. Kennington and R.V. Helgason. *Algorithms for network programming.* John Wiley and Sons, New York, NY, 1980.

[81] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 224:1093–1096, 1979. English Translation: *Soviet Mathematics Doklady*, Volume 20, pp. 1093–1096.

[82] M. Kojima, S. Shindoh, and S. Hara. Interior point methods for the monotone semidefinite linear complementarity problem in symmetric matrices. *SIAM Journal on Optimization*, 7:86–125, 1997.

[83] J.B. Kruskal. On the shortest spanning tree of graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[84] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[85] E. K. Lee and J. E. Mitchell. Computational experience in nonlinear mixed integer programming. In *Proceedings of Symposium on Operations Research, August 1996, Braunschweig, Germany*, pages 95–100. Springer-Verlag, 1996.

[86] K. Levenberg. A method for the solution of certain problems in least squares. *Quart. Appl. Math.*, 2:164–168, 1944.

[87] Y. Li, P.M. Pardalos, K.G. Ramakrishnan, and M.G.C. Resende. Lower bounds for the quadratic assignment problem. *Annals of Operations Research*, 50:387–410, 1994.

[88] Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–262. American Mathematical Society, 1994.

[89] L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979.

[90] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Interior point methods for linear programming: Computational state of the art. *ORSA Journal on Computing*, 6(1):1–14, 1994. See also the following commentaries and rejoinder.

[91] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM J. Appl. Math.*, 11:431–441, 1963.

[92] S. Mehrotra. On the implementation of a (primal–dual) interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[93] S. Mehrotra and J. Wang. Conjugate gradient based implementation of interior point methods for network flow problems. In L. Adams and J. Nazareth, editors, *Linear and Nonlinear Conjugate Gradient Related Methods*. SIAM, 1995.

[94] J. E. Mitchell. Interior point algorithms for integer programming. In J. E. Beasley, editor, *Advances in Linear and Integer Programming*, chapter 6, pages 223–248. Oxford University Press, 1996.

[95] J. E. Mitchell. Interior point methods for combinatorial optimization. In Tamás Terlaky, editor, *Interior Point Methods in Mathematical Programming*, chapter 11, pages 417–466. Kluwer Academic Publishers, 1996.

[96] J. E. Mitchell. Computational experience with an interior point cutting plane algorithm. Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, February 1997. Revised: April 1997.

[97] J. E. Mitchell. Fixing variables and generating classical cutting planes when using an interior point branch and cut method to solve integer programming problems. *European Journal of Operational Research*, 97:139–148, 1997.

[98] J. E. Mitchell. An interior point cutting plane algorithm for Ising spin glass problems. Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, July 1997.

[99] J. E. Mitchell and B. Borchers. Solving real-world linear ordering problems using a primal-dual interior point cutting plane method. *Annals of Operations Research*, 62:253–276, 1996.

[100] J. E. Mitchell and B. Borchers. Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm. Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, September 1997.

[101] J. E. Mitchell and M. J. Todd. Solving combinatorial optimization problems using Karmarkar's algorithm. *Mathematical Programming*, 56:245–284, 1992.

[102] R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. Part I: Linear programming. *Mathematical Programming*, 44(1):27–41, 1989.

[103] R. D. C. Monteiro, I. Adler, and M. G. C. Resende. A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and its power series extension. *Mathematics of Operations Research*, 15(2):191–214, 1990.

[104] J.J. Moré and D.C. Sorenson. Computing a trust region step. *SIAM J. of Stat. Sci. Comput.*, 4:553–572, 1983.

[105] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, New York, 1988.

[106] Y. E. Nesterov and A. S. Nemirovsky. *Interior Point Polynomial Methods in Convex Programming : Theory and Algorithms*. SIAM Publications. SIAM, Philadelphia, USA, 1993.

[107] Y. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22:1–42, 1997.

[108] M. W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[109] R. Pai, N. K. Karmarkar, and S. S. S. P. Rao. A global router for gate-arrays based on Karmarkar's interior point methods. In *Proceedings of the Third International Workshop on VLSI System Design*, pages 73–82, 1990.

[110] P. M. Pardalos, K. G. Ramakrishnan, M. G. C. Resende, and Y. Li. Implementation of a variance reduction-based lower bound in a branch-and-bound algorithm for the quadratic assignment problem. *SIAM Journal on Optimization*, 7:280–294, 1997.

[111] P. M. Pardalos and M.G.C. Resende. Interior point methods for global optimization problems. In T. Terlaky, editor, *Interior Point Methods of Mathematical Programming*, pages 467–500. Kluwer Academic Publishers, 1996.

[112] P. M. Pardalos and H. Wolkowicz, editors. *Topics in Semidefinite and Interior-Point Methods.* Fields Institute Communications Series. American Mathematical Society, New Providence, Rhode Island, 1997.

[113] P.M. Pardalos. Continuous approaches to discrete optimization problems. In G. Di Pillo and F. Giannessi, editors, *Nonlinear optimization and applications.* Plenum Publishing, 1996.

[114] P.M. Pardalos and H. Wolkowicz, editors. *Quadratic assignment and related problems*, volume 16 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1994.

[115] R. G. Parker and R. L. Rardin. *Discrete Optimization.* Academic Press, San Diego, CA 92101, 1988.

[116] L. Portugal, F. Bastos, J. Júdice, J. Paixão, and T. Terlaky. An investigation of interior point algorithms for the linear transportation problem. *SIAM J. Sci. Computing*, 17:1202–1223, 1996.

[117] L. Portugal, M.G.C. Resende, G. Veiga, and J. Júdice. An efficient implementation of an infeasible primal-dual network flow method. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey, 1994.

[118] R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[119] K. G. Ramakrishnan, M. G. C. Resende, and P. M. Pardalos. A branch and bound algorithm for the quadratic assignment problem using a lower bound based on linear programming. In C. Floudas and P.M. Pardalos, editors, *State of the Art in Global Optimization: Computational Methods and Applications.* Kluwer Academic Publishers, 1995.

[120] K.G. Ramakrishnan, N.K. Karmarkar, and A.P. Kamath. An approximate dual projective algorithm for solving assignment problems. In David S. Johnson and Catherine C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, volume 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 431–451. American Mathematical Society, 1993.

[121] K.G. Ramakrishnan, M.G.C. Resende, and P.M. Pardalos. A branch and bound algorithm for the quadratic assignment problem using a lower bound based on linear programming. In *State of the Art in Global Optimization: Computational Methods and Applications*, pages 57–73. Kluwer Academic Publishers, 1996.

[122] K.G. Ramakrishnan, M.G.C. Resende, B. Ramachandran, and J.F. Pekny. Tight QAP bounds vias linear programming. In *From Local to Global Optimization.* Kluwer Academic Publishers, 1998. To appear.

[123] M. Ramana and P. M. Pardalos. Semidefinite programming. In T. Terlaky, editor, *Interior Point Methods of Mathematical Programming*, pages 369–398. Kluwer Academic Publishers, 1996.

[124] S. Ramaswamy and J. E. Mitchell. On updating the analytic center after the addition of multiple cuts. Technical Report 37–94–423, DSES, Rensselaer Polytechnic Institute, Troy, NY 12180, October 1994.

[125] S. Ramaswamy and J. E. Mitchell. A long step cutting plane algorithm that uses the volumetric barrier. Technical report, DSES, Rensselaer Polytechnic Institute, Troy, NY 12180, June 1995.

[126] M.G.C. Resende, P.M. Pardalos, and Y. Li. FORTRAN subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, To appear.

[127] M.G.C. Resende, K.G. Ramakrishnan, and Z. Drezner. Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. *Operations Research*, 43(5):781–791, 1995.

[128] M.G.C. Resende, T. Tsuchiya, and G. Veiga. Identifying the optimal face of a network linear program with a globally convergent interior point method. In W.W. Hager, D.W. Hearn, and P.M. Pardalos, editors, *Large scale optimization: State of the art*, pages 362–387. Kluwer Academic Publishers, 1994.

[129] M.G.C. Resende and G. Veiga. Computing the projection in an interior point algorithm: An experimental comparison. *Investigación Operativa*, 3:81–92, 1993.

[130] M.G.C. Resende and G. Veiga. An efficient implementation of a network interior point method. In David S. Johnson and Catherine C. McGeoch, editors, *Network Flows and Matching: First DIMACS Implementation Challenge*, volume 12 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 299–348. American Mathematical Society, 1993.

[131] M.G.C. Resende and G. Veiga. An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapaciated networks. *SIAM Journal on Optimization*, 3:516–537, 1993.

[132] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization: An Interior Point Approach.* John Wiley, Chichester, 1997.

[133] Günther Ruhe. *Algorithmic Aspects of Flows in Networks.* Kluwer Academic Publishers, Boston, MA, 1991.

[134] C.-J. Shi, A. Vannelli, and J. Vlach. An improvement on Karmarkar's algorithm for integer programming. In P.M. Pardalos and M.G.C. Resende, editors, *COAL Bulletin – Special issue on Computational Aspects of Combinatorial Optimization*, volume 21, pages 23–28. Mathematical Programming Society, 1992.

[135] L.P. Sinha, B.A. Freedman, N.K. Karmarkar, A. Putcha, and K.G. Ramakrishnan. Overseas network planning. In *Proceedings of the Third International Network Planning Symposium – NETWORKS'86*, pages 8.2.1–8.2.4, June 1986.

[136] R. Van Slyke and R. Wets. L-shaped linear programs with applications to optimal control and stochastic linear programs. *SIAM Journal on Applied Mathematics*, 17:638–663, 1969.

[137] R.E. Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[138] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical Programming*, 73:291–341, 1996.

[139] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996.

[140] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Boston, 1996.

[141] R.J. Vanderbei, M.S. Meketon, and B.A. Freedman. A modification of Karmarkar's linear programming algorithm. *Algorithmica*, 1:395–407, 1986.

[142] J.P. Warners, T. Terlaky, C. Roos, and B. Jansen. Potential reduction algorithms for structured combinatorial optimization problems. Technical Report 95-88, Delft University of Technology, Delft, 1995.

[143] J.P. Warners, T. Terlaky, C. Roos, and B. Jansen. A potential reduction approach to the frequency assignment problem. Technical Report 95-98, Delft University of Technology, Delft, 1995.

[144] H. Wolkowicz and Q. Zhao. Semidefinite programming relaxations for the graph partitioning problem. Technical report, Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada, October 1996.

[145] H. Wolkowicz and Q. Zhao. Semidefinite programming relaxations for the set partitioning problem. Technical report, Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, N2L 3G1 Canada, October 1996.

[146] S. Wright. *Primal-dual interior point methods.* SIAM, Philadelphia, 1996.

[147] Y. Ye. On affine scaling algorithms for nonconvex quadratic programming. *Mathematical Programming*, 56:285–300, 1992.

[148] Y. Ye. *Interior Point Algorithms: Theory and Analysis.* John Wiley, New York, 1997.

[149] Quey-Jen Yeh. *A reduced dual affine scaling algorithm for solving assignment and transportation problems.* PhD thesis, Columbia University, New York, NY, 1989.

[150] Y. Zhang. On the convergence of a class of infeasible interior-point methods for the horizontal linear complementarity problem. *SIAM Journal on Optimization*, 4(1):208–227, 1994.

[151] Q. Zhao. *Semidefinite programming for assignment and partitioning problems.* PhD thesis, Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada, 1996.

[152] Q. Zhao, S. E. Karisch, F. Rendl, and H. Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problem. Technical Report 95–27, Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada, September 1996.