Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 156 (2008) 2166–2177
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

The colourful feasibility problem

Antoine Deza[a], Sui Huang[a], Tamon Stephen[b,*], Tamás Terlaky[a]

[a]Advanced Optimization Laboratory, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada L8S 4K1
[b]Department of Mathematics, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada V5A 1S6

Received 29 November 2005; received in revised form 30 November 2007; accepted 10 January 2008
Available online 6 March 2008

Dedicated to Leonid Khachiyan

Abstract

We study a colourful generalization of the linear programming feasibility problem, comparing the algorithms introduced by Bárány and Onn with new methods. This is a challenging problem on the borderline of tractability, its complexity is an open question. We perform benchmarking on generic and ill-conditioned problems, as well as recently introduced highly structured problems. We show that some algorithms can lead to cycling or slow convergence and we provide extensive numerical experiments which show that others perform much better than predicted by complexity arguments. We conclude that the most efficient method is a proposed multi-update algorithm.

MSC: 52C45; 68W40; 90C60; 68Q25

Keywords: Colourful linear programming; Colourful feasibility problem

1. Introduction

Given colourful sets $S_1, \ldots, S_{d+1}$ of points in $\mathbb{R}^d$ and a point p in $\mathbb{R}^d$, the *colourful linear programming* problem is to express p as a convex combination of points $x_1, \ldots, x_{d+1}$ with $x_i \in S_i$ for each i. This problem was presented by Bárány and Onn in 1997 [3], it is still not known if a polynomial-time algorithm for the problem exists. The monochrome version of this problem, expressing p as a convex combination of points in a set S, is a traditional linear programming feasibility problem.

In this paper, we study algorithms for colourful linear programming with a core condition from an experimental point of view. We learn several things. First, in our experience this problem is easy—we expend more effort to generate difficult examples than to solve them. Second, while the classical algorithms for this problem already perform quite well, we introduce modifications that achieve a substantial improvement in practical performance. Third, we construct examples where ill-conditioning leads to slow convergence for the some otherwise very effective algorithms. And finally, we remark that a simple greedy heuristic provides competitive results in practice but we find a case where it fails to solve the problem at all. Additionally we provide benchmarking that, we hope, will encourage research on this attractive problem.

* Corresponding author. Tel.: +1 778 782 7429.
E-mail addresses: deza@mcmaster.ca (A. Deza), huangs3@optlab.mcmaster.ca (S. Huang), tamon@sfu.ca (T. Stephen), terlaky@mcmaster.ca (T. Terlaky).

0166-218X/$ - see front matter
doi:10.1016/j.dam.2008.01.016

2. Definitions and background

We call a system of $d+1$ sets of $d+1$ points a *configuration*, and often denote it as $\mathbf{S}=\{S_1, \ldots, S_{d+1}\}$. Such configurations are the simplest non-degenerate cases of colourful linear programming. We define the *core* of a configuration to be

$$\bigcap_{i=1}^{d+1} \operatorname{conv}(S_i).$$

In this paper we consider the *colourful feasibility* problem of expressing a given p in the interior of the core as a colourful convex combination of points in the configuration. By Bárány's colourful Carathéodory theorem [1], a solution is guaranteed to exist, and the problem is to exhibit one. This problem is described in [3] as "an outstanding problem on the border line between tractable and intractable problems".

Several close relatives of the colourful feasibility problem are known to be difficult. For example, the case where we have d colours in $\mathbb{R}^d$ and no restriction on the size of the sets has been shown to be strongly NP-complete through a reduction of 3-SAT. We refer to [3] for more details.

In [1], Bárány proposed a finite algorithm **A1** to solve colourful feasibility, and in [3] Bárány and Onn analysed the complexity of **A1** and a second algorithm **A2**. (See Section 3 for a detailed description of these and other algorithms.) Both these algorithms are essentially geometric, and the complexity guarantees depend crucially on having the point p in the *interior* of the core. In effect, the distance between p and the boundary of the core can be considered as a measure of the conditioning of the problem. Thus for a configuration $\mathbf{S}$ we define ρ to be the radius of the largest ball around p that is contained in the core. The results for **A1** and **A2** are effectively that they are polynomial in d and $1/\rho$. While this is not polynomial in the input, it suggests that a polynomial algorithm may be possible. We remark that for configurations of $d+1$ points in $d+1$ colours on the unit sphere $\mathbb{S}^d \subseteq \mathbb{R}^d$, ρ will be small even if the problem has a favourable special structure, and quite small otherwise.

It is helpful to preprocess the problem by translating the point p to be the vector $\vec{0}$ in $\mathbb{R}^d$. If $\vec{0}$ is a point in one of the S_i's, then the solution to the colourful feasibility problem is trivial. Otherwise, we can also scale the points of the S_i's so that they lie on the unit sphere $\mathbb{S}^d$. The coordinates in any resulting convex combination can then be unscaled as a post-processing step.

We remark that restricting the sets to have size $d+1$ is not a burden since, given a larger set, solving a monochrome linear feasibility problem allows us to efficiently find a basis of size $d+1$ with $\vec{0}$ in its convex hull.

The colourful feasibility problem models a data mining situation where we want to select a set of points that is both diverse, in the sense that it includes representatives from predetermined classes (colours), and representative, in the sense that the selected points surround a specified point common to all the classes [9]. Application of this problem to combinatorics are discussed in [4].

3. Seven algorithms

In this paper we consider the theoretical and practical performance of seven algorithms for finding a colourful basis. The algorithms considered are the algorithms of Bárány **A1** and of Bárány and Onn **A2**, modifications of these algorithms which update multiple colours at each stage, which we will call **A3** and **A4** and a hybrid **A5** of these designed to take advantage of the strengths of both algorithms. For purposes of comparison, we also consider two simple approaches that perform well under certain circumstances: a greedy heuristic where we choose the adjacent simplex of maximum volume **A6** and a random sampling approach **A7**. All our implementations are initialized with the first points from each colour. Following are descriptions of the algorithms, see [7] for MATLAB implementations of each. Besides **A7**, they are implemented as pivoting algorithms with the respective pivot selection rule.

3.1. *Bárány's algorithm* A1

We begin with the algorithm proposed by Bárány [1], which is a pivoting algorithm. It begins with say a random colourful simplex $\mathbf{\Delta}$. The point x nearest to $\vec{0}$ in $\mathbf{\Delta}$ is computed. If $x \neq \vec{0}$, then x must lie on at least one facet of $\mathbf{\Delta}$. Consider the colour i of the vertex of $\mathbf{\Delta}$ that is not on this facet. Look for the point t of colour i minimizing the inner product $\langle t, x\rangle$. Then we replace the point of colour i from $\mathbf{\Delta}$ with the point t to get a new simplex. The algorithm then repeats beginning with the new simplex.

The convergence of this algorithm relies on the fact that $\vec{0}$ is in the core of the configuration. For this reason the affine hyperplane perpendicular to the vector x cannot separate $\vec{0}$ from the points of colour i. Thus the next simplex will have a point closer to $\vec{0}$ than $\mathbf{\Delta}$ did, and the algorithm will converge in finitely many steps. If, additionally, the core has radius at least ρ around $\vec{0}$, then there is a guarantee that a given step will decrease the squared norm of the nearest point by at least a factor of $(1-\rho^2/4)$. Using this, it is possible to show that **A1** will approach the solution in $\mathrm{O}(1/\rho^2)$ iterations. Since an iteration can be done in polynomial time, this proves that **A1** runs in time polynomial in the input data and $1/\rho$. Consult [3] for details and a proof.

We note that the complexity of a single iteration is dominated by the cost of the nearest point subroutine. This can be solved as a continuous convex quadratic optimization problem, but involves numerical issues: It can be solved to less or greater precision, either risking numerical error or increasing the running time. For the purposes of our benchmarking, we used the `MATLAB` built-in `quadprog()` which gave fairly good results, see Section 5.2.

3.2. Bárány and Onn's algorithm **A2**

The reliance of **A1** on nearest point calculations is a disadvantage. Partly motivated by this, Bárány and Onn proposed an alternate algorithm for the colourful feasibility problem whose calculations involve only linear algebra. This algorithm, **A2**, is described in [3].

The key idea is to replace the closest point x to $\vec{0}$ on the simplex $\mathbf{\Delta}$ by a point y on the boundary of $\mathbf{\Delta}$ that can be computed algebraically. The initial choice of y could be one of the vertices of the initial simplex. In subsequent iterations, a colour j corresponding to a zero coefficient in y is chosen. An improving vertex v of colour j is found, and y_{new} is updated by projecting $\vec{0}$ onto the line segment between y and v and finding where the resulting vector enters the new simplex. As with **A1**, this algorithm takes $\mathrm{O}(1/\rho^2)$ iterations, and hence is polynomial in the input data and $1/\rho$, see [3].

The implementation of **A2** proposed in [3] takes time $\Theta(d^4)$ for a single iteration. The bottleneck is computing y_{new}, which is the intersection of the line segment from $\vec{0}$ to a point p and the new simplex. In fact we observe that this can be done in time $\mathrm{O}(d^3)$. First, compute the defining equations for the simplex $Ay_{\text{new}} \geqslant b$ by inverting the homogenized matrix of the vertices. We know the intersection point will be of the form $y_{\text{new}} = \alpha p$. We can substitute this into the above inequalities to get $\alpha(Ap) \geqslant b$ and simply take α to be the maximum value of $b_i/A_i p$ for $i = 1, 2, \ldots, d+1$. This is implemented in [7].

As noted by Maurice Queyranne, it is possible to modify **A2** to compute the nearest point on the simplex using Wolfe's algorithm for finding the nearest point on a polytope [12]. While it does not have a polynomial time guarantee, it may work well for this problem. Like **A2**, Wolfe's algorithm uses simple linear algebra to pivot through faces; it could be adapted to use y_{new} as a warm start.

3.3. Multi-update Bárány **A3**

We propose the following modification of **A1**: if it happens that the nearest point x to $\vec{0}$ of the current simplex $\mathbf{\Delta}$ lies on a low-dimensional face of $\mathbf{\Delta}$—i.e., on more than one facet—then we update *every* colour that is not a vertex of that face. After finding each new point, we replace x by x_{new}, the projection of $\vec{0}$ onto the line segment from x to the vertex we are adding to the simplex. The advantage of this new algorithm, which we call **A3**, is that when possible it updates several colours without recomputing a nearest point.

Since this algorithm makes at least as much progress as **A1** at each iteration, we get convergence in at most the same number of iterations. A given iteration may take longer, since it has to update multiple points. However, aside from the nearest point calculation, all steps in an iteration of **A1** can be performed in $\mathrm{O}(d^2)$ arithmetic operations. Hence the additional work per iteration of **A3** is $\mathrm{O}(d^3)$, and the bottleneck remains the single nearest point calculation.

3.4. Multi-update Bárány and Onn **A4**

Similarly, we can adjust algorithm **A2** to pivot multiple colours when y lies on a low-dimensional face. As in **A3** we update y by setting y_{new} to the projection of $\vec{0}$ onto the line from y to the new vertex. This is faster than the computation of y from **A2** at the end of the iteration, which remains the bottleneck. We call this algorithm **A4**. It is particularly useful at the start of the algorithm since the initial point y is a vertex of $\mathbf{\Delta}$. This algorithm will take no more iterations than **A2**, and each iteration costs at most a constant factor more than an iteration of **A2**.

3.5. *Hybrid* A5

In Section 5 we describe a situation where **A2** and **A4** are slow because they repeatedly return to the same simplex, see the example in Section 6.1. A practical solution to this is to run **A4**, but use a computationally heavy step from **A3** if we detect that **A4** is returning to the same simplex. We implemented such a hybrid algorithm **A5**.

3.6. *Maximum volume* A6

We also considered the performance of some greedy heuristics. The most effective of these was to pivot from $\mathbf{\Delta}$ to an adjacent simplex of maximum volume given that the pivoting hyperplane separates $\mathbf{\Delta}$ from $\vec{0}$. This heuristic, which we call **A6**, uses simpler linear algebra than **A2**, and by taking large simplices often gets to $\vec{0}$ in a small number of steps. We can perform an iteration of this algorithm in $\mathrm{O}(d^4)$ time.

3.7. *Random sampling* A7

Finally, we remark on a very simple guess and check algorithm where we sample simplices at random and check to see if they contain $\vec{0}$. Intuitively we would not expect such an algorithm to work well. However, as discussed in [6] solutions to a given colourful feasibility problem may not be all that rare, and in some cases can be quite frequent. Since guessing and checking are relatively fast operations, it is worth considering the possibility that this naive algorithm may perform well in special cases or low dimension. We call this algorithm **A7**.

One attractive feature of **A7** is that the cost of an iteration is low—we only have to generate a random simplex and then test if it contains $\vec{0}$. The test can be done in $\mathrm{O}(d^3)$ time by solving a linear system.

4. Random, ill-conditioned and extremal problems

To better understand how various algorithms perform in practice, we produced a test suite of challenging colourful feasibility problems, which includes unstructured random problems, ill-conditioned problems and problems with a restricted number of solutions. In this section we describe three types of colourful feasibility problems that we consider when evaluating the practical performance of an algorithm. See [7] for a MATLAB implementation of each of these problem generators.

4.1. *Unstructured random problems*

The first class of problems we consider are unstructured random problems. We take $d+1$ points in each of $d+1$ colours on $\mathbb{S}^d$. The only restriction we require is that $\vec{0}$ is in the core. This is achieved by taking the last point to be a random convex combination of the antipodes on $\mathbb{S}^d$ of the first d points. We call this generator **G1**.

4.2. *Ill-conditioned random problems*

Next, we consider ill-conditioned problems. We place d points of a given colour on the spherical cap around the point $(0, 0, \ldots, 0, 1)$ and the final point of that colour in the opposite spherical cap, again as a convex combination of the antipodes. The maximum angle between a chosen vector and the final coordinate axis is a parameter, and points are concentrated towards the centre rather than uniformly distributed on the cap. Since the points all lie in a tube around the final coordinate axis, we call these *tube* generators. We implemented two tube generators: **G2** randomly places either 1 or d points of colour i on the positive side of the axis, while **G3** always places d points of colour i on the positive side of the axis.

4.3. Problems with a restricted number of solutions

Finally, we consider problems where we control the number of colourful simplices containing $\vec{0}$. The paper [6] provides new bounds for the number of possible solutions to a colourful feasibility problem with $\vec{0}$ in the interior of the core. It turns out that the number of simplices containing $\vec{0}$ in dimension d can be as low as quadratic in d, but not lower, see [2,10], or as high as $d^{d+1} + 1$ (with $\rho > 0$), which is more than one-third of the total number of simplices. Constructions are given for colourful feasibility problems attaining both these values.

The probability that a simplex generated by $d + 1$ points chosen randomly on $\mathbb{S}^d$ contains $\vec{0}$ is $1/2^d$, see for example [11]. Thus in a uniformly generated random problem of the type generated by **G1**, we would expect about $1/2^d$ of the $(d + 1)^{d+1}$ colourful simplices to contain $\vec{0}$. This is not a large fraction, but in the context of an effective pivoting algorithm such as **A1** which may pivot several neighbours to a given solution, and pivot several neighbours of the first neighbour onto it, etc., we can entertain the idea that for a random configuration most simplices are close to a solution. See Section 6.4 for further discussion.

We might expect that the difficulty of a colourful feasibility problem increases as the number of solutions, i.e. simplices containing $\vec{0}$, decreases, so we wrote three problem generators based on the constructions in [6]. The first, **G4** generates perturbed versions of the configuration from [6] with many solutions. These problems have $d^{d+1} + 1$ of the $(d + 1)^{d+1}$ simplices containing $\vec{0}$, many more than random configurations, and we expect them to be quite easy. The second, **G5**, generates configurations where one point of each colour is close to each vertex of a regular simplex on $\mathbb{S}^d$. There are $d!$ solutions corresponding to picking a different colour from each vertex, this is still much less than the $(d + 1)^{d+1}/2^d$ expected in a random configuration. Finally, we have **G6**, which generates perturbed versions of the configuration from [6] which has only $d^2 + 1$ solutions. The generators **G4–G6** randomly permute the order of the points that appear within each colour.

All these problems are ill-conditioned in the sense that points are clustered closely together. Also ρ will be quite small for **G4** and **G6**, although the construction **G5** maximizes ρ for configurations on $\mathbb{S}^d$, with $\rho = 1/d$.

5. Benchmarking and results

In this section, we describe the results of computational experiments in which we run the colourful feasibility algorithms against our problem generators. We focus on the number of iterations that an algorithm takes to find a solution, but in Section 5.2 we also include information about the cost of iterations. Graphs summarizing these results are in Appendix C. The two particularly difficult, but fragile, examples of Sections 6.1 and 6.2 are not included in these results.

5.1. Iteration counts

For each type of problem we ran tests of the algorithms in dimensions 3×2^n for $n = 0, 1, 2, 3, 4, 5, 6, 7$. Dimension 3 is our starting point since the seven algorithms degenerate to three simple and effective algorithms in dimension 2. We use the factor 2 increase to sample higher dimensions with less frequency as we get higher. We believe this yields a reasonable sample of low, intermediate and high dimensional problems.

Note that a colourful feasibility problem instance in dimension d consists of $(d + 1)^2$ points in dimension d. Thus the size of the input is cubic in d. At present it is logistically difficult to generate and store a colourful feasibility problem in dimension $d = 1000$. After dimension 100, it also becomes increasingly difficult to cope with numerical errors, especially for the algorithms that include nearest point calculations, namely **A1**, **A3** and **A5**. For this reason we do not include results for these algorithms beyond $d = 96$ for except for the relatively well-conditioned **G1** problems where we stopped at $d = 192$. We only include results from the **A7** algorithm when they can be completed in a reasonable amount of time.

The results of our computational experiments are presented in the graphs in Appendix C. We have made the tables containing the raw data for these graphs available at [5]. Each graph presents results for a single random generator on a log–log scale with the average iteration count of each algorithm plotted against the dimension. Additionally, the tables contain the values of the largest iteration count observed in each type of trial; these show similar trends to the averages, although we notice that **A2** and **A4** sometimes perform substantially worse than the average, especially in the presence of ill-conditioning. The reasons for this are discussed in Section 6.2.

For each generator at $d = 3$ we sampled 100,000 problems, at $d = 6$ and 12 we sampled 10,000 problems, at $d = 24$ and 48 we sampled 1000 problems and finally for $d \geqslant 96$ we sampled 100 problems. The results are plotted on as log–log graphs in Figs. C1–C6. We remark that polynomials appear asymptotically linear in log–log plots, with the slope of the asymptote being the exponent of the leading term of the polynomial and the y-intercept of the asymptote representing the lead coefficient.

For tube experiments **G2** and **G3**, we used an angle parameter of $\pi/6$, that is, all the vectors in the configuration made an angle of at most $\pi/6$ with the x-axis. If we decrease the angle parameter which controls the width of the tube and hence the conditioning, all the algorithms become less stable numerically and experience a degradation in performance. In the cases of **A2** and **A4** they become substantially slower.

5.2. Cost per iteration

In Fig. C7 we present the average iteration times observed for all seven algorithms on problems from the **G1** generator. We comment that the average time to complete an iteration does not change significantly with the problem type, so we have not included the similar graphs for other generators. The data show that in our implementation of these algorithms, the average time for an iteration is never very large. For the slowest algorithms in the highest dimensions the average iteration took less than 2 s.

Unlike the other algorithms, the average iteration time for **A5** will be substantially affected by the conditioning of the problem. Using the well-conditioned **G1** problems, **A5** usually degenerates to **A4** and has a very similar average iteration time. As the problems become more ill-conditioned, **A5** will begin to use **A3** steps as well, and the average iteration time will increase towards the average iteration time for **A3**.

6. Discussion and worst-case constructions

Our experiments reveal several features of colourful feasibility algorithms. After considerable searching, we found a problem instance which caused **A6** to cycle. We also found that **A2** and **A4** can converge extremely slowly in the face of ill-conditioning although **A1** and **A3** continue to perform reasonably well on the same examples. We conclude that computationally the best algorithms are the multi-update variants and remark that these tightened algorithms do yield substantial gains over the originals.

6.1. A cycling example for **A6** *in dimension 4*

In Appendix A we exhibit an example in dimension 4 for which the maximum volume heuristic cycles. Since this example shows that **A6** can cycle, it is remarkable that it happens so rarely. It did not occur in the entire test suite of Section 5. We were unable to find any examples of cycling in dimension 3 or any examples of cycling in dimension 4 with cycle length shorter than 6. Higher dimensions and longer cycle lengths do occur.

One explanation for the results is that as one might expect, **A6** is an effective heuristic in a typical situation. The distinguishing feature of the few bad examples is that the points are placed in such a way that the simplices cluster into a few groups of similar shape and volume. The heuristic of taking the maximum volume is then not very helpful in choosing promising simplices. We note that this example is solved easily by the other algorithms.

6.2. Flip-flopping during convergence for **A2***: 40,847 iterations in dimension 3*

We constructed an example of a colourful feasibility problem in dimension 3 that takes 40,847 iterations to solution using a basic implementation of **A2**. The exact points we used are contained in Appendix B. The algorithm is initialized with the simplex that uses the first point of each colour. At the fifth iteration, the algorithm reaches a situation where the current point y lies on a facet F of colours 2, 3 and 4 very close to $\vec{0}$. Using this point the algorithm will pick the point of colour 1 that has minimum dot product with y. The second and third points of colour 1 lie almost in the directions of y and $-y$; however, neither of these forms a simplex with F containing $\vec{0}$. In fact the fourth point of colour 1 does form a simplex containing $\vec{0}$ with F, but it is nearly orthogonal to y. As a result, after two iterations, **A2** returns to the same simplex. The point y will be recomputed at each step, and is slightly closer to $\vec{0}$ when the algorithm returns to the previous simplex. However, the improvement is quite small. Of course ρ is also very small, so this is consistent with the performance guarantee described in Section 3.2. The algorithm then proceeds to return to the same simplex more than 20,000 times, with an incremental improvement to y at each iteration before finally taking the fourth point of colour 1 and terminating.

As one would expect with a very ill-conditioned problem, this example is numerically fragile—the current version of our code normalizes the coordinates before starting and does not suffer the same fate. However, bad behaviour is fairly typical. The tube generator for ill-conditioned problems in [7] produces problems whose ill-conditioning depends on a parameter defining the width of the tube. As the width decreases, we get an increasing number of cases where **A2** and **A4** take enormous numbers of iterations.

We remark that, in contrast, **A1** never returns to the same simplex, so it cannot suffer from this type of flip-flopping. Indeed in dimension 3 it could do no worse than visiting all $4^4 = 256$ simplices. At least 10 of these must contain $\vec{0}$, see [2], so the algorithm must terminate in at most 246 iterations. It is quite hard to see how this limit could be approached. The authors wonder if a Klee–Minty-like example, see [8], of worst-case behaviour for Bárány's pivoting algorithm could be constructed.

6.3. Advantages of multiple updates and initialization

The multi-update algorithms **A3** and **A4** do provide substantial gains over their single update counterparts, **A1** and **A2**. In the case of **A3**, we get a large reduction in iteration count at very little cost in terms of iteration time. In our benchmarking experiments, this produced times that were competitive with **A2** and much better than **A1**. The gains for **A4** relative to **A2** are less impressive. In our benchmarking experiments, **A4** consistently averaged a 10–40% savings in total time to solution.

We have not discussed the effects of the initial simplex in this paper, but we can employ various heuristics to choose a good initial simplex. A few of these are implemented in [7]. We found that the most useful initialization heuristic was to run the first iteration of **A4**. This runs in $O(d^3)$ time and improves the subsequent iteration counts of the algorithms, with the obvious exception of **A7**.

6.4. Theoretical complexity of the algorithms

In Section 3, we remarked that Bárány and Onn proved a worst-case bound for **A1** and **A2** of $O(1/\rho^2)$ iterations up to numerical considerations and we improved their iteration time for **A2** from $O(d^4)$ to $O(d^3)$. We also mentioned that we do not expect the multi-update and hybrid algorithms to improve the theoretical bounds. From the example of Section 6.1, we see that **A6** is not guaranteed to converge. The expected running time of **A7** is 1 over the probability that random simplex contains $\vec{0}$, i.e. around 2^d for random problems, and as bad as $(d+1)^{d+1}/(d^2+1)$ for the type of problems generated by **G6**.

The poor performance of **A2** on ill-conditioned problems and examples like that of Section 6.2 confirm the worst-case predictions of Bárány and Onn's analysis. On the other hand, we did not see this type of behaviour for **A1**, and it is hard to see how it could occur.

The model proposed in Section 4.3 is that a pure pivoting algorithm such as **A1**, defines a set of rooted trees on the $(d+1)^{d+1}$ simplices. Each simplex which contains $\vec{0}$ is the root of a tree, and we draw an edge between the vertices representing simplices $\mathbf{\Delta}_1$ and $\mathbf{\Delta}_2$ if when **A1** encounters $\mathbf{\Delta}_1$ it pivots to $\mathbf{\Delta}_2$. Then the worst performance of the algorithm in terms of the number of iterations would be the height of the highest tree. A smart algorithm will produce short trees by pivoting several simplices to a given simplex at a lower level.

Consider a situation where trees have a constant expansion factor k near the base, that is, low level vertices are connected to roughly k vertices in the level above. The number of trees is $p(d+1)^{d+1}$ where p is the probability that a simplex contains $\vec{0}$. If the trees expand up to height h, each tree will contain on the order of k^h vertices. Then we must have $k^h p(d+1)^{d+1} \leqslant (d+1)^{d+1}$, the total number of vertices. Rearranging, we get $h \leqslant -\log_k(p)$. This expression predicts the average iteration count for **A1** to grow linearly for **G1** problems, to be constant for **G4** problems and to grow at $\Theta(d \log d)$ for **G6** problems. All of these match very well with our observed results. The **G5** problems are predicted to be more difficult than they are observed to be, but that is not surprising given their simple structure.

7. Summary and future work

Despite the examples of Sections 6.1 and 6.2, the results presented in Section 5 show that, except for **A7** and to a lesser degree **A6**, all the algorithms did a good job of solving all the problems. We did find that the methods which include nearest point calculations were more vulnerable to numerical errors than **A2** and **A4**, since our implementations began to crash once we got past $d = 100$, especially on ill-conditioned problems. For the most part, the reduced iteration counts of the nearest point algorithms do not offset the extra time spent per iteration compared to **A2** and **A4**. In some cases of extreme ill-conditioning, such as in Section 6.2, **A2** and **A4** will take many additional iterations and be much slower compared to the nearest point algorithms. In this situation either a hybrid algorithm such as **A5** or the basic **A1** or **A3** would work better.

We finish by returning to the motivating question of Bárány and Onn: Is there a polynomial time algorithm for colourful feasibility? By improving the implementation of **A2**, we have improved the worst case for this algorithm from $O(d^4/\rho^2)$ to $O(d^3/\rho^2)$; however, the dependence on ρ has not improved. Indeed our experiments give strong evidence that the analysis for **A2** is tight.

The situation for **A1** is less clear. We do not see the same bad behaviour with ill-conditioned problems that we found for **A2**, so it is possible that a better guarantee exists for this algorithm. In light of the model suggested in Section 6.4 it is quite difficult to see how to construct a Klee–Minty-like bad case for **A1** as discussed in Section 6.2. We view this as an appealing challenge.

Acknowledgements

We thank the referees for helpful comments and Zhaosong Lu for suggesting data mining as an application in Section 2. This research was supported by NSERC Discovery grants for the four authors, by the Canada Research Chair program for the first and last authors and by an MITACS grant for the second and third authors. The third author worked on this project as part of the Discrete Optimization project of the IMO at the University of Magdeburg.

Appendix A. Example in dimension 4 where A6 cycles

This example consists of five points in each of the five colours in $\mathbb{R}^4$. The points are presented in Table A1. They are grouped by colour, with the rows representing x, y, z and w coordinates, respectively.

Table A1
Coordinates of points of an example where **A6** cycles in dimension 4

Red points				
7/52	1/89	−1/60	−1/28	4/127
1/176	−8/65	5/49	6/35	9/118
4/29	1/961	−8/191	1/40	−1/75
$-\frac{\sqrt{4238906047}}{66352}$	$\frac{\sqrt{30434652805951}}{5559385}$	$\frac{\sqrt{11360296502737439}}{107254140}$	$-\frac{\sqrt{69789743}}{31640}$	$-7\frac{\sqrt{25600756871}}{1123950}$
Green points				
3/85	−5/71	8/45	3/88	−1/114
−1/67	1/10	−38/155	−2/131	−24/185
1/173	−2/101	1/95	3/53	7/85
$\frac{\sqrt{29008089867051134}}{170445655}$	$-\frac{\sqrt{5063381959}}{71710}$	$2\frac{\sqrt{159502559}}{26505}$	$5\frac{\sqrt{14863381455}}{610984}$	$\frac{\sqrt{125498719055}}{358530}$
Blue points				
−3/77	4/141	3/22	16/111	−3/46
−3/20	−4/63	−3/17	5/29	3/47
−2/71	−3/173	−5/79	−1/210	1/33
$-\frac{\sqrt{470161115387}}{694309}$	$-8\frac{\sqrt{122080034994545}}{88619769}$	$-\frac{\sqrt{826050579}}{29546}$	$-\frac{\sqrt{48208184671}}{225330}$	$\frac{\sqrt{5043188147}}{71346}$
Tan points				
1/59	6/151	8/45	−3/29	11/76
1/29	−1/122	−7/32	4/43	−1/8
3/56	1/536	8/97	−1/14	9/59
$25\frac{\sqrt{14625287}}{95816}$	$\frac{\sqrt{554855708771634695}}{745501496}$	$\frac{\sqrt{17827555751}}{139680}$	$-\frac{\sqrt{297327743}}{17458}$	$\frac{\sqrt{75612155}}{8968}$
White points				
1/167	3/43	11/52	−19/65	−3/100
1/241	−1/244	−5/134	2/129	1/62
1/53	2/9	13/142	1/4386	−4/73
$-5\frac{\sqrt{120112106864502146289 1}}{173320847963}$	$-\frac{\sqrt{8432767415}}{94428}$	$-\frac{\sqrt{57852799351}}{247364}$	$-\frac{\sqrt{74312211919}}{285090}$	$\frac{\sqrt{50998516979}}{226300}$

The initial simplex is taken to be (1, 1, 1, 1, 1), i.e., the first point of each colour. The algorithm proceeds to visit simplices (1, 1, 4, 1, 1), (3, 1, 4, 1, 1), (3, 1, 4, 3, 1), (3, 1, 1, 3, 1) and (1, 1, 1, 3, 1) before returning to the original simplex and repeating. At steps one, three and five, there are two candidate colours for pivoting, the candidates that are not chosen for pivoting are 1, 3 and 4, respectively. In the even numbered steps there is a single candidate colour for pivoting.

Appendix B. Example in dimension 3 where A2 takes 40,847 iterations

This example consists of four unnormalized points in each of the four colours in $\mathbb{R}^3$. The points are presented in Table B1. They are grouped by colour, with the rows representing x, y and z coordinates, respectively.

The initial simplex is taken to be (1, 1, 1, 1), i.e., the first point of each colour. It then updates to (1, 3, 1, 1), (1, 3, 2, 1), (1, 3, 2, 3), (1, 3, 2, 2) and reaches (3, 3, 2, 2) on the fifth iteration. At this point, it begins to flip between (3, 3, 2, 2) and (2, 3, 2, 2) with y initially alternating between values close to (0.2, ±0.00200, 0.00285). The values of all these coordinates decrease very slowly as the algorithm continues. At iteration 40,847 it chooses fourth point of colour 1 instead of the third. This makes the current simplex (4, 3, 2, 2) which contains $\vec{0}$.

Table B1
Coordinates of points of an example taking 40,847 iterations of **A2** in dimension 3

Red points			
1.00000320775369	−0.01000436049274	−0.01000129525998	1.00000089660284
0.00000340785030	0.99999739350954	−1.00000497855619	0.00000051797159
0.00999859615603	0.00000371775824	0.00000030149139	−0.01999639732055
Green points			
1.00000363763560	−0.00999644886160	−0.00999943004295	1.00000335962280
−0.00000325123594	1.00000064545156	−1.00000169806216	−0.00000080450760
0.01000493174811	−0.00000024088601	0.00000009099437	−0.01999811804365
Blue points			
0.99999949817337	−0.00999587145461	−0.00999627213896	0.99999551963712
−0.00000260397964	1.00000485455718	−1.00000419710665	−0.00000024626161
0.00999854691703	0.00000123671997	−0.00000381812529	−0.01999801526314
Tan points			
0.99999980645233	0.10000000280522	−0.60000327600988	0.99999642880542
0.00000024487465	−0.98999719313413	0.79999695643245	−0.00000429109491
0.01000455311709	−0.00000405877812	0.00000372117690	−0.01000272055280

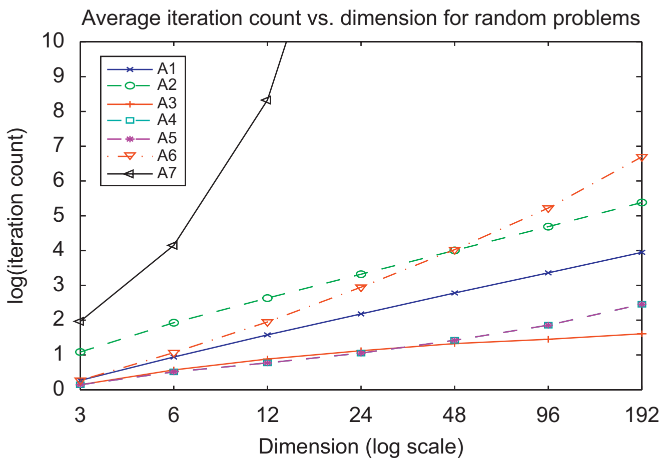

Fig. C1. Results for **G1**.

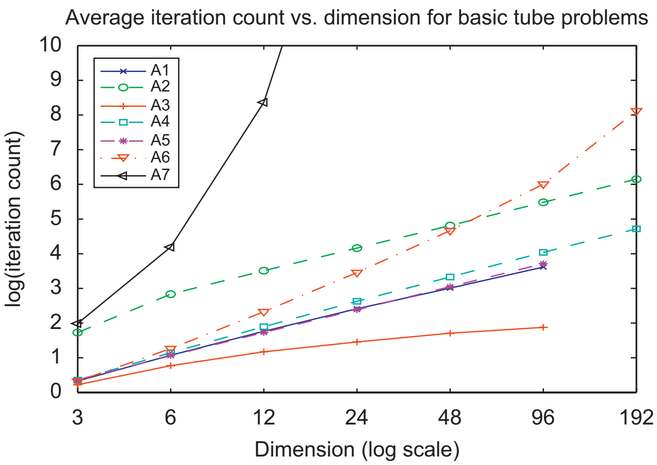

Fig. C2. Results for **G2**.

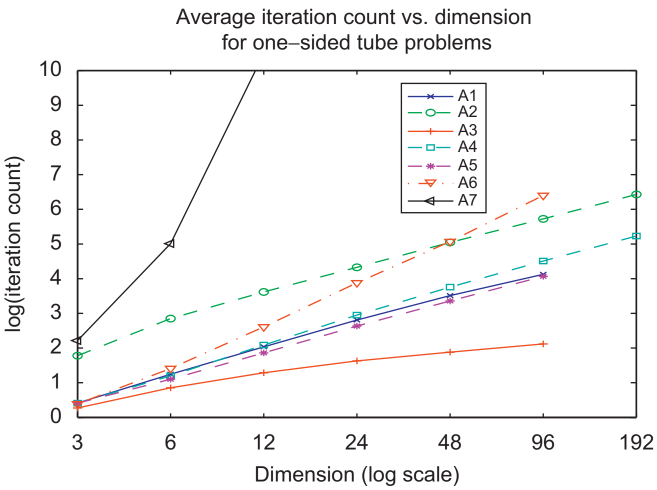

Fig. C3. Results for **G3**.

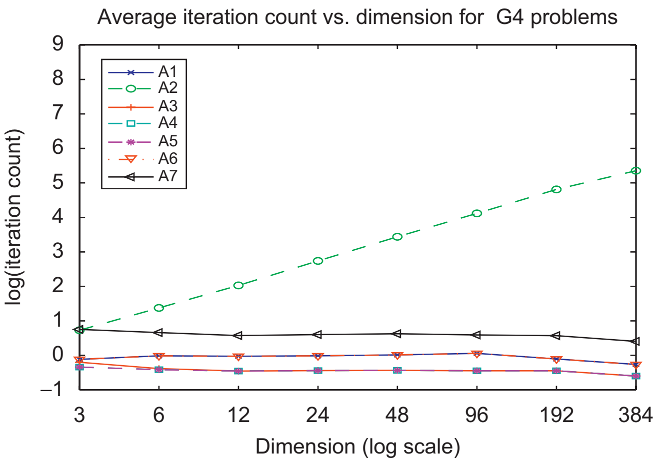

Fig. C4. Results for **G4**.

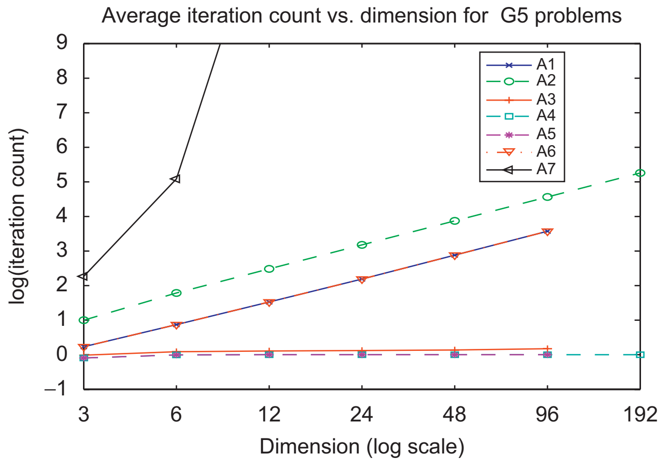

Fig. C5. Results for **G5**.

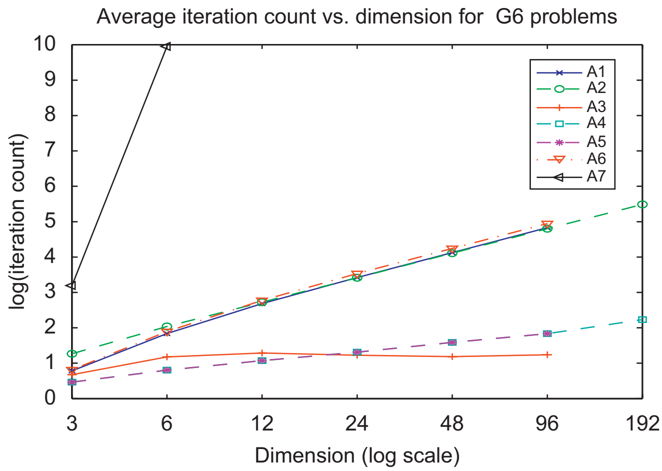

Fig. C6. Results for **G6**.

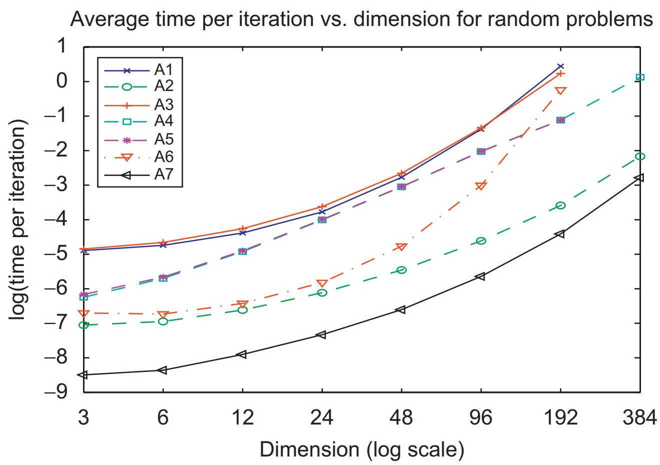

Fig. C7. Average iteration time of the algorithms.

Appendix C. Computational results

Results of the generators **G1–G6** and the average iteration time of the algorithms are shown in Figs. C1–C7.

References

[1] I. Bárány, A generalization of Carathéodory's theorem, Discrete Math. 40 (2–3) (1982) 141–152.
[2] I. Bárány, J. Matoušek, Quadratically many colorful simplices, SIAM J. Discrete Math. 21 (1) (2007) 191–198.
[3] I. Bárány, S. Onn, Colourful linear programming and its relatives, Math. Oper. Res. 22 (3) (1997) 550–567.
[4] I. Bárány, S. Onn, Carathéodory's theorem, colourful and applicable, in: Intuitive Geometry, Budapest, 1995, Bolyai Society Mathematical Studies, vol. 6, János Bolyai Mathematical Society, Budapest, 1997, pp. 11–21.
[5] A. Deza, S. Huang, T. Stephen, T. Terlaky, Algorithms and tests for the colourful feasibility problem, AdvOL-Report 2005/20, McMaster University, 2005, updated 2007.
[6] A. Deza, S. Huang, T. Stephen, T. Terlaky, Colourful simplicial depth, Discrete Comput. Geom. 35 (4) (2006) 597–604.
[7] S. Huang, MATLAB code for colourful linear programming, available at: ⟨http://optlab.mcmaster.ca/~huangs3/CLP/⟩ and ⟨http://www.math.sfu.ca/~tamon/Software/CLP/⟩.
[8] V. Klee, G.J. Minty, How good is the simplex algorithm?, in: Inequalities III, Proceedings of the 3rd Symposium, Los Angeles 1969, Academic Press, New York, 1972, pp. 159–175.
[9] Z. Lu, personal communication, 2006.
[10] T. Stephen, H. Thomas, A quadratic lower bound for colourful simplicial depth, J. Comb. Optim., to appear.
[11] U. Wagner, E. Welzl, A continuous analogue of the upper bound theorem, Discrete Comput. Geom. 26 (2) (2001) 205–219.
[12] P. Wolfe, Finding the nearest point in a polytope, Math. Programming 11 (1976) 128–149.