Noname manuscript No.
(will be inserted by the editor)

Recent Results on Douglas–Rachford Methods for Combinatorial Optimization Problems

Francisco J. Aragón Artacho · Jonathan M. Borwein · Matthew K. Tam

the date of receipt and acceptance should be inserted later

Abstract We discuss recent positive experiences applying convex feasibility algorithms of Douglas–Rachford type to highly combinatorial and far from convex problems.

Keywords Douglas–Rachford · projections · reflections · combinatorial optimization · modelling · feasibility · satisfiability · Sudoku · Nonograms

Mathematics Subject Classification (2010) 90C27 · 90C59 · 47N10

F.J. Aragón Artacho

Systems Biochemistry Group, University of Luxembourg, 4362, Esch-sur-Alzette, Luxembourg.

E-mail: francisco.aragon@ua.es

J.M. Borwein · M.K. Tam (✉)

CARMA, University of Newcastle, Callaghan, NSW 2308, Australia.

E-mail: matthew.k.tam@gmail.com

J.M. Borwein

Also Distinguished Professor, KAU Jeddah, SA.

E-mail: jon.borwein@gmail.com

1 Introduction

Douglas–Rachford iterations, as defined in Section 2, are moderately well understood when applied to finding a point in the intersection of two convex sets. Over the past decade, they have proven very effective in some highly non-convex settings; even more surprisingly this is the case for some highly discrete problems. In this paper we wish to advertise the use of Douglas–Rachford methods in such combinatorial settings. The remainder of the paper is organized as follows.

In Section 2, we recapitulate what is proven in the convex setting. This is followed, in Section 3, by a review of the normal way of handling a (large) finite number of sets in the product space. In Section 4, we reprise what is known in the non-convex setting. Now there is less theory but significant and often positive experience. In Section 5, we turn to more detailed discussions of combinatorial applications before focusing, in Section 6, on solving *Sudoku puzzles*, and, in Section 7, on solving *Nonograms*. It is worth noting that both of these are NP-complete as decision problems. We complete the paper with various concluding remarks in Section 8.

2 Convex Douglas–Rachford Methods

In this section we review what is known about the behaviour of Douglas–Rachford methods applied to a finite family of closed and convex sets.

2.1 The Classical Douglas–Rachford Method

The classical Douglas–Rachford scheme was originally introduced in connection with partial differential equations arising in heat conduction [1], and convergence later proven in [2], who proposed the scheme for finding zeros of the sum of two maximal monotone operators. Throughout this paper, we consider application of the Douglas–Rachford scheme to feasibility problems. For the precise connection between these two applications, we refer the reader to [3] and the references therein.

Given two subsets A, B of a Hilbert space, $\mathcal{H}$, the scheme iterates by repeatedly applying the 2-*set Douglas–Rachford operator*,

$$T_{A,B} := \frac{I + R_B R_A}{2},$$

where I denotes the *identity mapping*, and $R_A(x)$ denotes the *reflection* of a point $x \in \mathcal{H}$ in the set A. The reflection can be defined as

$$R_A(x) := 2P_A(x) - x,$$

where $P_A(x)$ is the *closest point projection* of the point x onto the set A, that is,

$$P_A(x) := \left\{ z \in A \colon \|x - z\| = \inf_{a \in A} \|x - a\| \right\}.$$

In general, the projection P_A is a set-valued mapping. If A is closed and convex, the projection is uniquely defined for every point in $\mathcal{H}$, thus yielding a single-valued mapping (see e.g. [4, Th. 4.5.1]).

In the literature, the Douglas–Rachford scheme is also known as "*reflect-reflect-average*" [5], and "*averaged alternating reflections (AAR)*" [6].

Applied to closed and convex sets, convergence is well understood and can be explained by using the theory of (firmly) nonexpansive mappings.

Theorem 2.1 (Douglas–Rachford Scheme) *Let $A, B \subseteq \mathcal{H}$ be closed and convex with nonempty intersection. For any $x_0 \in \mathcal{H}$, set $x_{n+1} = T_{A,B}x_n$. Then (x_n) converges weakly to a point x such that $P_A x \in A \cap B$.*

As part of their analysis of *von Neumann's alternating projection method*, Bauschke and Borwein [7] introduced the notion of the *displacement vector*, v, and used the sets E and F to generalize $A \cap B$.

$$v := P_{\overline{B-A}}(0), \quad E := A \cap (B - v), \quad F := (A + v) \cap B.$$

Note, if $A \cap B \neq \emptyset$ then $E = F = A \cap B$.

The same framework was utilized by Bauschke, Combettes and Luke [6] to analyze the Douglas–Rachford method.

Theorem 2.2 (Infeasible case [6, Th. 3.13]) *Let $A, B \subseteq \mathcal{H}$ be closed and convex. For any $x_0 \in \mathcal{H}$, set $x_{n+1} = T_{A,B}x_n$. Then the following hold.*

(i) $x_{n+1} - x_n = P_B R_A x_n - P_A x_n \to v$ *and* $P_B P_A x_n - P_A x_n \to v$.

(ii) If $A \cap B \neq \emptyset$ then (x_n) converges weakly to a point in

$$\mathrm{Fix}(T_{A,B}) = (A \cap B) + N_{\overline{A-B}}(0);$$

otherwise, $\|x_n\| \to +\infty$.

(iii) Exactly one of the following two alternatives holds.

(a) $E = \emptyset$, $\|P_A x_n\| \to +\infty$, *and* $\|P_B P_A x_n\| \to +\infty$.

(b) $E \neq \emptyset$, the sequences $(P_A x_n)$ and $(P_B P_A x_n)$ are bounded, and their weak cluster points belong to E and F, respectively; in fact, the weak cluster points of

$$((P_A x_n, P_B R_A x_n)) \text{ and } ((P_A x_n, P_B P_A x_n)) \tag{1}$$

are best approximation pairs relative to (A, B).

Here, $N_C(x) := \{u \in \mathcal{H} : \langle c - x, u\rangle \leq 0, \forall c \in C\}$ denotes the *normal cone* to a convex set $C \subset \mathcal{H}$ at a point $x \in C$, and $\mathrm{Fix}(T) := \{x \in \mathcal{H} : x \in T(x)\}$ denotes the set of *fixed points* of the mapping T.

Remark 2.1 (Behaviour of best approximation pairs) If best approximation pairs relative to (A, B) exist and P_A is weakly continuous, then the sequences in (1) actually converge weakly to such a pair [6, Remark 3.14(ii)].

Since $x_n/n \to -v$, $\|x_n/n\|$ can be used to approximate $\|v\| = d(A, B)$ [6, Remark 3.16(ii)]. ◊

For other pertinent references relating to the classical Douglas–Rachford method, see [8–13].

We turn next to an alternative new method:

2.2 The Cyclic Douglas–Rachford Method

There are many possible generalizations of the classic Douglas–Rachford iteration. Given three sets A, B, C and $x_0 \in \mathcal{H}$, an obvious candidate is the iteration defined by repeatedly setting $x_{n+1} := T_{A,B,C}x_n$ where

$$T_{A,B,C} := \frac{I + R_C R_B R_A}{2}. \tag{2}$$

For closed and convex sets, like $T_{A,B}$, the mapping $T_{A,B,C}$ is firmly nonexpansive, and has at least one fixed point provided $A \cap B \cap C \neq \emptyset$. Using a well known theorem of Opial [14, Th. 1], (x_n) can be shown to converge weakly to a fixed point. However, attempts to obtain a point in the intersection using said fixed point have, so far, been unsuccessful.

Example 2.1 (Failure of three set Douglas–Rachford iterations.) We give an example showing the iteration described in (2) can fail to find a feasible point. Consider the one-dimensional subspaces $A, B, C \subset \mathbb{R}^2$ defined by

$$A := \{\lambda(0,1) : \lambda \in \mathbb{R}\},$$
$$B := \{\lambda(\sqrt{3},1) : \lambda \in \mathbb{R}\},$$
$$C := \{\lambda(-\sqrt{3},1) : \lambda \in \mathbb{R}\}.$$

Then $A \cap B \cap C = \{(0,0)\}$.

Let $x_0 = (-\sqrt{3}, -1)$. Since $x_0 \in \operatorname{Fix} R_C R_B R_A$,

$$x_0 \in \operatorname{Fix} \frac{I + R_C R_B R_A}{2}.$$

However,

$$P_A x_0 = (0,-1), \quad P_B x_0 = x_0 = (-\sqrt{3},-1), \qquad P_C x_0 = (-\sqrt{3}/2, 1/2).$$

That is, $P_A x_0, P_B x_0, P_C x_0 \notin A \cap B \cap C$. The trajectory is illustrated in Fig. 1. $\Diamond$

Instead, Borwein and Tam [15] considered cyclic applications of 2-set Douglas–Rachford operators. Given N sets $C_1, C_2, \ldots, C_N$, and $x_0 \in \mathcal{H}$, their *cyclic Douglas–Rachford scheme* iterates by repeatedly setting $x_{n+1} := T_{[C_1,C_2,\ldots,C_N]} x_n$, where $T_{[C_1,C_2,\ldots,C_N]}$ denotes the *cyclic Douglas–Rachford operator* defined by

$$T_{[C_1,C_2,\ldots,C_N]} := T_{C_N,C_1} T_{C_{N-1},C_N} \ldots, T_{C_2,C_3} T_{C_1,C_2}.$$

In the consistent case, the iterations behave analogously to the classical Douglas–Rachford scheme (cf. Theorem 2.1).

Theorem 2.3 (Cyclic Douglas–Rachford) *Let $C_1, C_2, \ldots, C_N \subseteq \mathcal{H}$ be closed and convex sets with a nonempty intersection. For any $x_0 \in \mathcal{H}$, set $x_{n+1} = T_{[C_1\,C_2\,\ldots\,C_N]} x_n$.*

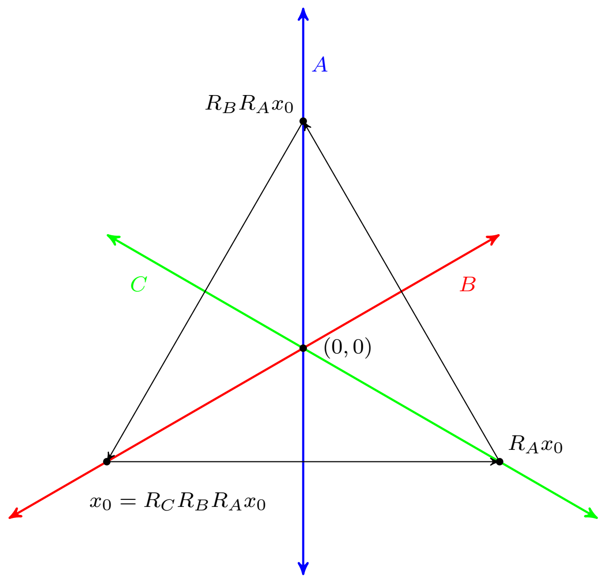


Fig. 1: Trajectory of Example 2.1.

Then (x_n) *converges weakly to a point* x *such that* $P_{C_i}x = P_{C_j}x$*, for all indices* i, j*. Moreover,* $P_{C_j}x \in \bigcap_{i=1}^N C_i$*, for each index* j*.*

Example 2.2 (Example 2.1 revisited) Consider the cyclic Douglas–Rachford scheme applied to the sets of Example 2.1. As before, let $x_0 = (-\sqrt{3}, -1)$. By Theorem 2.3, the sequence (x_n) converges to a point x such that

$$P_A x = P_B x = P_C x = (0, 0).$$

Furthermore, P_A, P_B, P_C are orthogonal projections, hence $x = (0, 0)$. The trajectory is illustrated in Fig. 2.

As a consequence of the problem's rotational symmetry, the sequence of Douglas–Rachford operators can be described by

$$T_{A,B}x_n = P_C x_n, \quad T_{B,C}T_{A,B}x_n = P_A P_C x_n, \quad x_{n+1} = T_{C,A}T_{B,C}T_{A,B}x_n = P_B P_A P_C x_n.$$

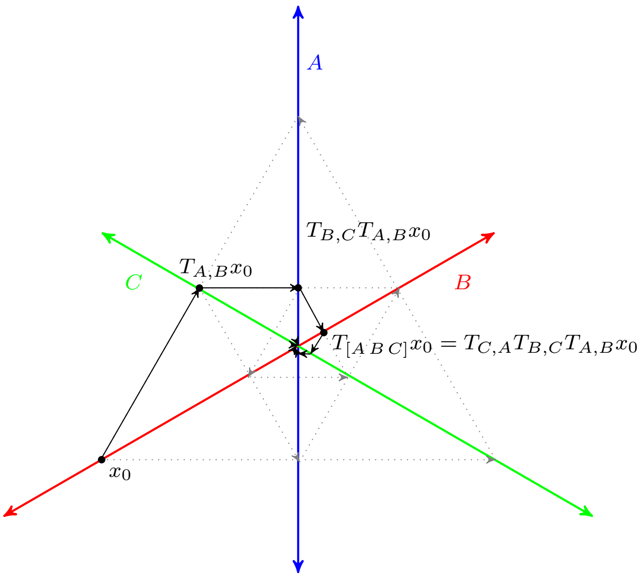


Fig. 2: Trajectory of Example 2.2. Solid black arrows represent 2-set Douglas–Rachford iterations (i.e. they connect the sequence $x_0, T_{A,B}x_0, T_{B,C}T_{A,B}x_0$, $T_{C,A}T_{B,C}T_{A,B}x_0, \dots$). Constructions (reflect-reflect-average) are dotted.

That is, starting at x_0, the cyclic Douglas–Rachford trajectory applied to the A, B, C, coincides with von Neumann's alternating projection method applied to C, A, B (cf. [15, Cor. 3.1]). ♢

If $N = 2$ and $C_1 \cap C_2 = \emptyset$ (the inconsistent case), unlike the classical Douglas–Rachford scheme, the iterates are not unbounded (cf. Theorem 2.2). Moreover, there is evidence to suggest that the scheme can be used to produce best approximation pairs relative to (C_1, C_2) whenever they exist.

The framework of Borwein and Tam [15], can also be used to derive a number of applicable variants. A particularly nice one is the *averaged Douglas–Rachford* *scheme* which, for any $x_0 \in \mathcal{H}$, iterates by repeatedly setting[1]

$$x_{n+1} := \frac{1}{N}\left(\sum_{i=1}^{N} T_{C_i,C_{i+1}}\right)x_n.$$

Since each 2-set Douglas–Rachford operator can be computed independently the iteration easily parallelizes.

Remark 2.2 (Failure of norm convergence) It is known that the alternating projection method may fail to converge in norm [16], and it follows that the cyclic Douglas-Rachford methods may also only converge weakly., see [15, Cor. 3.1.] for details. For the classical method Douglas-Rachford method, this seems to be unresolved and the examples of from [16, Section 5] do not apply. ◊

2.2.1 Numerical Performance

Applied to the problem of finding a point in the intersection of N balls in $\mathbb{R}^m$, initial numerical experiments suggest that the cyclic Douglas–Rachford outperforms the classical Douglas–Rachford scheme [15].

To ensure this performance is not an artefact of having highly symmetrical constraints, the same problem, replacing the balls with prolate spheroids (the type of ellipsoid obtained by rotating a 2-dimensional ellipse around its major axis) having one common focus was considered. Unlike ball constraints, there is no simple formula for computing the projection onto a spheroid. However, the projections can be computed efficiently. The process reduces to numerically solving, for t, the equation

$$\frac{a^2u^2}{(a^2-t)^2} + \frac{b^2v^2}{(b^2-t)^2} = 1,$$

[1] Here indices are understood modulo N. That is, $C_{N+1} := C_1$.

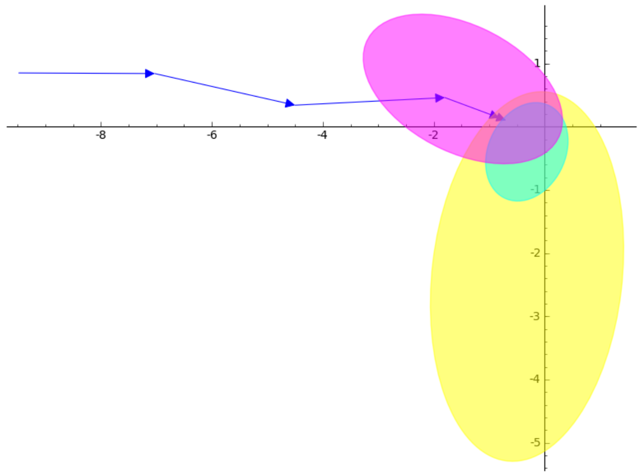


Fig. 3: A cyclic Douglas–Rachford trajectory for three ellipses in $\mathbb{R}^2$. Blue arrows represent 2-set Douglas–Rachford iterations (i.e. they connect the sequence x_0, $T_{A,B}x_0, T_{B,C}T_{A,B}x_0, T_{C,A}T_{B,C}T_{A,B}x_0, \ldots$).

for constants $a, b > 0$ and $u, v \in \mathbb{R}$. For further details, see [17, Ex. 2.3.18].

In the spheroid case, the computational results are very similar to the ball case, considered in [15]. An example having three spheroids in $\mathbb{R}^2$ is illustrated in Fig. 3.

3 Feasibility Problems in the Product Space

Given $C_1, C_2, \cdots, C_N \subset \mathbb{R}^m$, the *feasibility problem*[2] asks:

$$\text{Find } x \in \bigcap_{i=1}^{N} C_i \subset \mathbb{R}^m. \tag{3}$$

A great many optimization and reconstruction problems, both continuous and combinatorial, can be cast within this framework.

[2] In this context, "feasibility" and "satisfiability" can be used interchangeably.

Define two sets $C, D \subset (\mathbb{R}^m)^N$ by

$$C := \prod_{i=1}^{N} C_i, \quad D := \{(x, x, \ldots, x) \in (\mathbb{R}^m)^N : x \in \mathbb{R}^m\}.$$

While the set D, the *diagonal*, is always a closed subspace, the properties of C are largely inherited. For instance, when $C_1, C_2, \ldots, C_N$ are closed and convex, so is C.

Consider, now, the equivalent feasibility problem:

$$\text{Find } \mathbf{x} \in C \cap D \subset (\mathbb{R}^m)^N. \tag{4}$$

Equivalent in the sense that

$$x \in \bigcap_{i=1}^{N} C_i \iff (x, x, \ldots, x) \in C \cap D.$$

Moreover, knowing the projections onto $C_1, C_2, \ldots, C_N$, the projections onto C and D can be easily computed. The proof has recourse to the standard characterization of orthogonal projections,

$$p = P_D x \iff \langle x - p, d\rangle = 0 \text{ for all } d \in D.$$

Proposition 3.1 (Product projections) *Let* $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in (\mathbb{R}^m)^N$. *Then*

$$P_D\mathbf{x} = \left(\frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i, \ldots, \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i\right),$$

and if $P_{C_1}(\mathbf{x}_1), \ldots, P_{C_N}(\mathbf{x}_N)$ *are nonempty then*

$$P_C\mathbf{x} = \prod_{i=1}^{N} P_{C_i}(\mathbf{x}_i).$$

Proof Let $(\mathbf{p}, \ldots, \mathbf{p}) \in D$ be the projection of $\mathbf{x}$ onto D. For any $\mathbf{d} \in \mathbb{R}^m$, one has $(\mathbf{d}, \ldots, \mathbf{d}) \in D$. Now

$$0 = \langle \mathbf{x} - (\mathbf{p}, \ldots, \mathbf{p}), (\mathbf{d}, \ldots, \mathbf{d})\rangle = \sum_{i=1}^{N}\langle \mathbf{x}_i - \mathbf{p}, \mathbf{d}\rangle = \langle \sum_{i=1}^{N} \mathbf{x}_i - N\mathbf{p}, \mathbf{d}\rangle;$$

whence, $\mathbf{p} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$. The proves the projection onto D.

We now prove the projection formula for C. For any $\mathbf{c} = (\mathbf{c}_1, \ldots, \mathbf{c}_N) \in C$ and $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_N) \in \prod_{i=1}^{N} P_{C_i}(\mathbf{x}_i) \subseteq C$,

$$\|\mathbf{x}-\mathbf{c}\|^2 = \sum_{i=1}^{N}\|\mathbf{x}_i - \mathbf{c}_i\|^2 \geq \sum_{i=1}^{N}\|\mathbf{x}_i - \mathbf{p}_i\|^2 = \|\mathbf{x}-\mathbf{p}\|^2 .$$

Since $P_C(\mathbf{x}) \subseteq C$, this shows $P_C\mathbf{x} \supseteq \prod_{i=1}^{N} P_{C_i}(\mathbf{x}_i)$.

Conversely, let $\mathbf{p} = (\mathbf{p}_1, \ldots, \mathbf{p}_N) \in P_C(\mathbf{x})$ and suppose $\mathbf{p}_j \notin P_{C_j}(\mathbf{x}_j)$ for some j. Define $\mathbf{q} := (\mathbf{q}_1, \ldots, \mathbf{q}_N) \in (\mathbb{R}^m)^N$ where $\mathbf{q}_j \in P_{C_j}(\mathbf{x}_k)$ and $\mathbf{q}_i = \mathbf{p}_i$ if $i \neq j$. Then

$$\|\mathbf{x}-\mathbf{p}\|^2 = \sum_{i=1}^{N}\|\mathbf{x}_i - \mathbf{p}_i\|^2 > \sum_{i=1}^{N}\|\mathbf{x}_i - \mathbf{q}_i\|^2 = \|\mathbf{x}-\mathbf{q}\|^2.$$

Since $\mathbf{q} \in C$, we conclude that $\mathbf{p} \notin P_C(\mathbf{x})$. This completes the proof. □

Most projection algorithms can be applied to feasibility problems with any finite number of sets without significant modification. An exception is the Douglas–Rachford scheme, which until [15] had only been successfully investigated for the case of two sets. This has made the product formulation crucial for the Douglas–Rachford scheme.

4 Non-convex Douglas–Rachford Methods

While there is not nearly so much theory in the non-convex setting, there are some useful beginnings:

4.1 Theoretical Underpinnings

As a prototypical non-convex scenario, Borwein and Sims [5] considered the Douglas–Rachford scheme applied to a Euclidean sphere and a line. More precisely, they

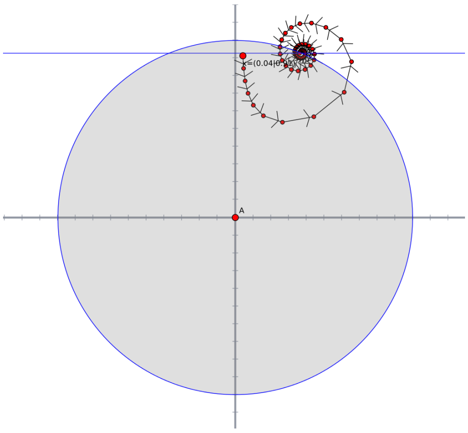


Fig. 4: A Douglas–Rachford trajectory showing local convergence to a feasible point, as in Theorem 4.1, exhibiting "spiralling" behaviour.

looked at the sets

$$S := \{x \in \mathbb{R}^m : \|x\| = 1\}, \quad L := \{\lambda a + \alpha b \in \mathbb{R}^m : \lambda \in \mathbb{R}\},$$

where, without loss of generality, $\|a\| = \|b\| = 1, a \perp b, \alpha > 0$. We summarize their findings.

Appropriately normalized the iteration becomes

$$\begin{aligned} x_{n+1}(1) &= x_n(1)/\rho_n, \\ x_{n+1}(2) &= \alpha + (1 - 1/\rho_n)x_n(2), \text{ and} \\ x_{n+1}(k) &= (1 - 1/\rho_n)x_n(k), \text{ for } k = 3, \dots, m, \end{aligned} \tag{5}$$

where $\rho_n := \|x_n\| := \sqrt{x_n(1)^2 + \ldots + x_n(m)^2}$, see [5] for details. The non-convex sphere, S, provides an accessible model of many reconstruction problems in which the magnitude, but not the phase, of a signal is measured.

Note $\alpha \in [0, 1]$ represents the consistent case, and $\alpha > 1$ the inconsistent one.

Theorem 4.1 (Sphere and line) *Given* $x_0 \in \mathbb{R}^m$ *define* $x_{n+1} := T_{S,L}x_n$*. Then:*

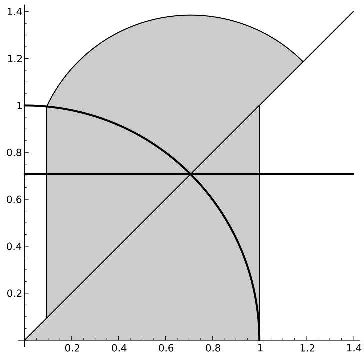


Fig. 5: The explicit region of convergence (grey) given in [18].

1. *If* $0 < \alpha < 1$, (x_n) *is locally convergent at each of* $\pm\sqrt{1-\alpha^2}a + \alpha b$.
2. *If* $\alpha = 0$ *and* $x_0(1) > 0$, (x_n) *converges to* a.
3. *If* $\alpha = 1$ *and* $x_0(1) \neq 0$, (x_n) *converges to* $\hat{y}b$ *for some* $\hat{y} > 1$.
4. *If* $\alpha > 1$ *and* $x_0(1) \neq 0$, $\|x_n\| \to \infty$.

Replacing L with the proper affine subspace, $A := A_0 + \alpha b$ for some non-trivial subspace A_0, (x_n) needs to be excluded from $A_0^\perp$. Now, if $x_0 \notin A_0^\perp$ then for some infeasible $q \neq 0$, $x_0 \in Q := A_0^\perp + \mathbb{R}q$, then (x_n) are confined to the subspace Q. Theorem 4.1 can, with some care then be extended to the following.

Corollary 4.1 (Sphere and non-trivial affine subspace) *For each feasible point* $p \in S \cap A \cap Q$ *there exists a neighbourhood* N_p *of* p *in* Q *such that starting from any* $x_0 \in N_p$ *the Douglas–Rachford scheme converges to* p.

If in Theorem 4.1 $x_0(1) = 0$, the behaviour of the scheme can provably be quite chaotic [5]. Indeed, this was a difficulty encountered by Aragón and Borwein [18], in giving an explicit region of convergence for the $\mathbb{R}^2$ case with $\alpha = 1/\sqrt{2}$.

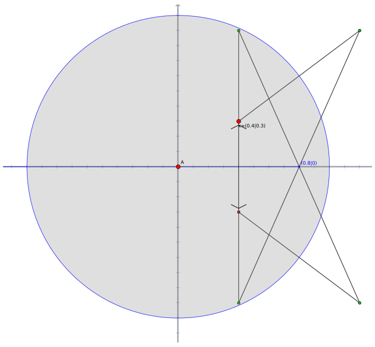


Fig. 6: A two cycle $(2/5, \pm 3/10)$.

Theorem 4.2 (Global convergence [18, Th. 2.1]) *Let* $x_0 \in [\epsilon, 1] \times [0, 1]$ *with*

$$\epsilon := (1 - 2^{-1/3})^{3/2} \approx 0.0937.$$

Then the sequence generated by the Douglas–Rachford scheme of (5) *with starting point* x_0 *is convergent to* $(1/\sqrt{2}, 1/\sqrt{2})$.

The restriction to $\alpha = 1/\sqrt{2}$ was largely made for notational simplicity.

In fact, a careful analysis show that the region of convergence is actually larger [18, Remark 2.12], as illustrated in Fig. 5.

Example 4.1 (Failure of Douglas–Rachford for a half-line and circle) Just replacing a line by a half line in the setting of Borwein–Sims [5, 18] is enough to allow complicated periodic behaviour.

Let

$$A := S_{\mathbb{R}^2} := \{x \in \mathbb{R}^2 : \|x\| = 1\}, \quad B := \{(x_1, 0) \in \mathbb{R}^2 : x_1 \leq a\}.$$

Then

$$P_A x = \begin{cases} x/\|x\| \text{ if } x \neq 0, \\ A \qquad \text{otherwise.} \end{cases}, \quad P_B x = \begin{cases} (x_1, 0) \text{ if } x_1 \leq a \\ (a, 0) \quad \text{otherwise.} \end{cases}$$

The following holds.

Proposition 4.1 *For each $a \in (0,1)$, there is a 2-cycle starting at*

$$x_0 = \left(a/2, \sqrt{1-a^2}/2\right).$$

Proof Since $\|x_0\| = \frac{1}{2}$,

$$R_A x_0 = 2\frac{x_0}{\|x_0\|} - x_0 = 3x_0.$$

Since $(R_A x_0)_1 = 3a/2 > a$, $P_B R_A x = (a, 0)$ and hence

$$T_{A,B} x_0 = \frac{x_0 + 2(a,0) - 3x_0}{2} = (a,0) - x_0 = \left(a/2, -\sqrt{1-a^2}/2\right).$$

By symmetry, $T^2_{A,B} x_0 = x_0$. □

If we replace B by the singleton $\{(a, 0)\}$ or the doubleton $\{(a, 0), (-1, 0)\}$ we obtain the same two-cycle. The case of a singleton shows the need for A to be non-trivial in Corollary 4.1.

This cycle is illustrated in Fig. 6 for $a = 4/5$ which leads to a rational cycle. For points near the cycle, the iteration generates remarkably subtle limit cycles as shown in Fig. 7.[3] ♢

In [19], Hesse and Luke utilize (S, ϵ)-*(firm) nonexpansiveness*, a relaxed local version of (firm) nonexpansiveness, a notion which quantifies how "close" to being (firmly) nonexpansive a mapping is. Together with a coercivity condition, and appropriate notions of super-regularity and linear strong regularity, their framework can be utilized to prove local convergence of the Douglas–Rachford scheme, if the first reflection is performed with respect to a subspace, see [19, Th. 42]. The order of reflection is reversed, so the results of Hesse and Luke do not directly overlap with that of Aragón, Borwein and Sims. This is not a substantive difference.

[3] See `http://carma.newcastle.edu.au/DRmethods/comb-opt/2cycle.html` for an animated version.

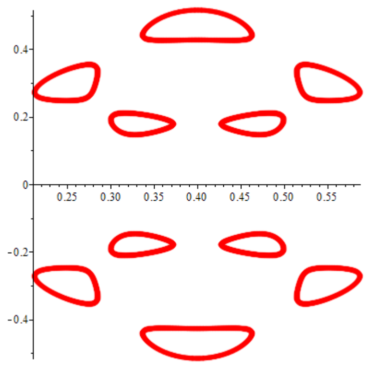


Fig. 7: The orbit starting at $(.49, .21)$.

Remark 4.1 Recently Bauschke, Luke, Phan and Wang [20] obtained local convergence results for a simpler algorithm, *von Neumann's alternating projection method (MAP)*, applied to sparsity optimization with affine constraints — a form of combinatorial optimization (Sudoku, for example, can be modelled in this framework [21]). In practice, however, our experience is that MAP often fails to converge satisfactorily when applied to these problems. See, for example, [22, Fig. 2]. ♢

4.2 A Summary of Applications

We briefly mention a variety of highly non-convex, primarily combinatorial, problems where some form of Douglas–Rachford algorithm has proven very fruitful.

1. *Protein folding* and *graph coloring* problems were first studied via Douglas–Rachford methods in [23] and [24], respectively.
2. *Image retrieval* and *phase reconstruction* problems are analyzed in some detail in [25, 26]. The *bit retrieval* problem is considered in [24].

3. *Matrix completion problems* were studied using Douglas–Rachford methods in [22]. This included finding various types of Hadamard matrices, and reconstruction of low-rank distance matrices. For a survey of matrix completion problem, see [27].
4. The *N-queens problem*, which requests the placement of N queens on a $N \times N$ chessboard, is studied and solved in [28].
5. *Boolean satisfiability* is treated in [24, 29]. Note that the three variable case, *3-SAT*, was the first problem to be shown *NP-complete* [30].
6. *TetraVex*[4] is an edge-matching puzzle (see Fig. 8), whose NP-completeness is discussed in [31], was studied in [32].[5] Problems up to size 4×4 could be solved in an average of 200 iterations. There are $10^{2n(n+1)}$ base-10 $n \times n$ boards, with $n = 3$ being the most popular.
7. Solutions of (very large) *Sudoku puzzles* have been studied in [28, 24]. For a discussion of NP-completeness of determining solvability of Sudokus see [33]. The effective solution of Suduko puzzles forms the basis of Section 6.
8. *Nonograms* [34, 35] are a more recent NP-complete Japanese puzzle whose solution by Douglas–Rachford methods is described in Section 7.[6]

[4] Also known as *McMahon Squares* in honour of the great English combinatorialist, Percy MacMahon, who examined them nearly a century ago.

[5] Pulkit Bansal did this as a 2010 NSERC summer student with Heinz Bauschke and Xianfu Wang.

[6] Japanese, being based on ideograms, does not lead itself to anagrams, crosswords or other word puzzles; this in part explains why so many good numeric and combinatoric games originate in Japan.

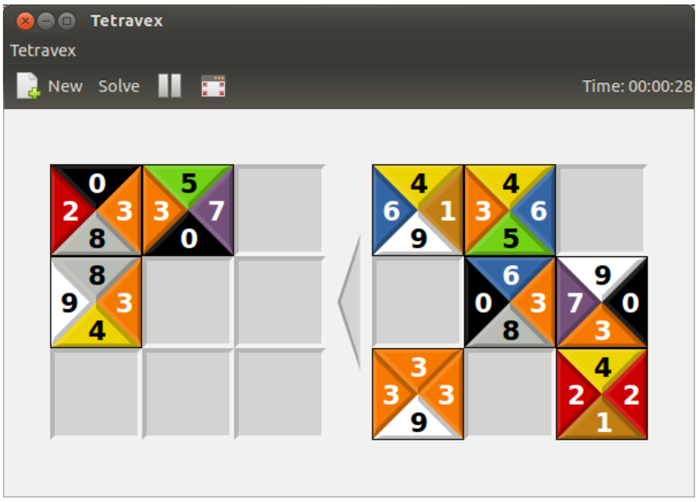


Fig. 8: A game of 3×3 TetraVex being played in *GNOME TetraVex*. Square tiles on the right board must be moved to the left board so that all touching numbers agree.

5 Successful Combinatorial Applications

The key to successful application is two-fold.

First, the iteration must converge—at least with high probability. Our experience is when that happens, random restarts in case of failure are very fruitful. As we shall show, often this depends on making good decisions about how to model the problem.

Second, one must be able to compute the requisite projections in closed form—or to approximate them efficiently numerically. As we shall indicate this is frequently possible for significant problems.

When these two events obtain, we are in the pleasant position of being able to lift much of our experience as continuous optimizers to the combinatorial milieu.

5.1 Model Formulation

Within the framework of feasibility problems, there can be numerous ways to model a given type of problem. The product space formulation (4) gives one example, even without assuming any additional knowledge of the underlying problem.

The chosen formulation heavily influences the performance of projection algorithms. For example, in initial numerical experiments, the cyclic Douglas–Rachford scheme of Section 2.2, was directly applied to (3). As a serial algorithm, it seems to outperform the classic Douglas–Rachford scheme, which must instead be applied to in the product space (4). For details see [15].

As a heuristic for problems involving one or more non-convex set, the sensitivity of the Douglas–Rachford method to the formulation used must be emphasized. In the (continuous) convex setting, the formulation influences performance of the algorithm, while in the combinatorial setting, the formulation determines whether or not the algorithm can successfully and reliably solve the problem at hand. Direct applications to feasibility problems with integer constraints have been largely unsuccessful. On the other hand, many of the successful applications outlined in Section 4.2 use binary formulations.

We now outline the basic idea behind these reformations. If

$$x \in \{c_1, c_2, \dots, c_m\} \subset \mathbb{R}. \tag{6}$$

We reformulate x as a vector $y \in \mathbb{R}^m$. If $x = c_i$, then $y = (y_1, \dots, y_m)$ is defined by

$$y_j = \begin{cases} 1 \text{ if } j = i, \\ 0 \text{ otherwise.} \end{cases}$$

With this interpretation (6) is equivalent to:

$$y \in \{e_1, e_2, \dots, e_m\} \subset \mathbb{R}^m,$$

with $y = e_i$ if and only if $x = c_i$.

Choosing $c_1, c_2, \ldots, c_m \in \mathbb{Z}$ takes care of the integer case.

5.2 Projection onto the Set of Permutations of Points

In many situations, in order to apply the Douglas–Rachford iteration, one needs to compute the projection of a point $x = (x_1, \ldots, x_m) \in \mathbb{R}^m$ onto the set of permutations of m given points $c_1, \ldots, c_m \in \mathbb{R}$, a set that will be denoted by $\mathcal{C}$. We shall see below that this is the case for the Sudoku puzzle.

As we show next, the projection can be easily and efficiently computed. In what follows, given $y \in \mathbb{R}^m$, we will denote by $[y]$ the vector with the same components permuted in nonincreasing order. We need the following classical rearrangement inequality, see [36, Th. 368].

Theorem 5.1 (Hardy–Littlewood–Pólya) *Any $x, y \in \mathbb{R}^m$ satisfy*

$$x^T y \leq [x]^T [y].$$

Fix $x \in \mathbb{R}^m$. Denote by $[\mathcal{C}]_x$ the set of vectors in $\mathcal{C}$ (which therefore have the same components but perhaps permuted) such that $y \in [\mathcal{C}]_x$ if the ith largest entry of y has the same index in y as the ith largest entry of x. As a consequence of Theorem 5.1, one has the following.

Proposition 5.1 (Projections on permutations) *Denote by $\mathcal{C} \subset \mathbb{R}^m$ the set of vectors whose entries are all permutations of $c_1, c_2, \ldots, c_m \in \mathbb{R}$. Then for any $x \in \mathbb{R}^m$,*

$$P_{\mathcal{C}} x = [\mathcal{C}]_x.$$

Proof For any $c \in \mathcal{C}$,

$$\begin{aligned} \|x - c\|^2 &= \|x\|^2 + \|c\|^2 - 2x^T c \\ &= \|[x]\|^2 + \|[c]\|^2 - 2x^T c \\ &\geq \|[x]\|^2 + \|[c]\|^2 - 2[x]^T [c] \\ &= \|[x] - [c]\|^2 \\ &= \|x - y\|^2, \text{ for } y \in [\mathcal{C}]_x. \end{aligned}$$

This completes the proof. □

Remark 5.1 In particular, taking $c_1 = 1$ and $c_2 = c_3 = \cdots = c_m = 0$ one has

$$\mathcal{C} = \{e_1, e_2, \ldots, e_m\},$$

where e_i denotes the ith standard basis vector; whence

$$P_{\mathcal{C}}(x) = \{e_i : x_i = \max\{x_1, x_2, \ldots, x_m\}\}.$$

A direct proof of this special case is given in [28, Section 5.9]. ◊

Remark 5.2 Proposition 5.1 suggests the following algorithm for computing a projection of x onto $\mathcal{C}$. Since the projection, in general, is not unique, we are content with finding the nearest point, p, in the set of projections or some other reasonable surrogate.

For convenience, given a vector $y \in (\mathbb{R}^2)^m$, we denote the projections onto the first and second product coordinates by Q and S, respectively. That is, if

$$y = ((x_1, c_2), (x_2, c_2), \ldots, (x_m, c_m)) \in (\mathbb{R}^2)^m,$$

then

$$Qy := (x_1, x_2, \ldots, x_m), \qquad Sy := (c_1, c_2, \ldots, c_m).$$

We can now can now state the following:

Algorithm 5.1 (Projection) Input: $x \in \mathbb{R}^m$ and $c_1, c_2, \ldots, c_m \in \mathbb{R}$.

1. By relabelling if necessary, assume $c_i \leq c_{i+1}$ for each i.
2. Set $y = ((x_1, c_1), (x_2, c_2), \ldots, (x_m, c_m)) \in (\mathbb{R}^2)^m$.
3. Set z to be a vector with the same components as y permuted such that Qz is in non-increasing order.
4. Output: $p = Sz$.

In our experience many projections required in combinatorial settings have this level of simplicity. $\Diamond$

6 Solving Sudoku Puzzles

We now demonstrate the reformulation described in Section 5 with Sudoku, modelled first as an integer feasibility problem, and secondly as a binary feasibility problem.

We introduce some notation. Denote by $A[i,j]$, the (i,j)-th entry of the matrix A. Denote by $A[i:i', j:j']$ the submatrix of A formed by taking rows i through i' and columns j through j' (inclusive). When i and i' are the indices of the first and last rows, we abbreviate by $A[:, j:j']$. We abbreviate similarly for the column indices. The *vectorization* of the matrix A by columns, is denoted by $\operatorname{vec} A$. For multidimensional arrays, the notation extends in the obvious way.

Let S denote the partially filled 9×9 integer matrix representing the incomplete Sudoku. For convenience, let $I = \{1, 2, \ldots, 9\}$ and let $J \subseteq I^2$ be the set of indices for which S is filled.

Whilst we will formulate the problem for 9×9 Sudoku, we note that the same principles can be applied to larger Sudoku puzzles.

6.1 Sudoku Modelled as an Integer Program

Sudoku is modelled as an integer feasibility problem in the obvious way. Denote by $\mathcal{C}$, the set of vectors which are permutations of $1, 2, \ldots, 9$. Then $A \in \mathbb{R}^{9\times 9}$ is a completion of S if and only if

$$A \in C_1 \cap C_2 \cap C_3 \cap C_4,$$

where

$$C_1 = \{A : A[i, :] \in \mathcal{C} \text{ for each } i \in I\},$$

$$C_2 = \{A : A[:, j] \in \mathcal{C} \text{ for each } j \in I\},$$

$$C_3 = \{A : \operatorname{vec} A[3i + 1 : 3(i + 1), 3j + 1 : 3(j + 1)] \in \mathcal{C} \text{ for } i, j = 0, 1, 2\},$$

$$C_4 = \{A : A[i, j] = S[i, j] \text{ for each } (i, j) \in J\}.$$

The projections onto C_1, C_2, C_3 are given by Proposition 5.1, and can be efficiently computed by using the algorithm outlined in Remark 5.2. The projection onto C_4 is given, pointwise, by

$$(P_{C_4}A)[i, j] = \begin{cases} S[i, j] \text{ if } (i, j) \in J, \\ A[i, j] \text{ otherwise;} \end{cases}$$

for each $(i, j) \in I^2$.

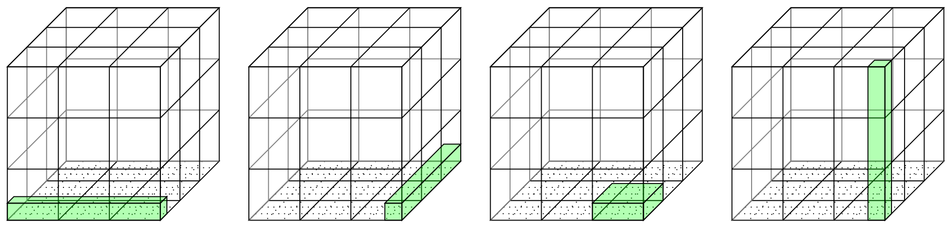


Fig. 9: Visualization of B showing constraints used in Sudoku modelled as a zero-one program. Green “blocks” are all “0”, save for a single “1”.

6.2 Sudoku Modelled as a Zero-One Program

Denote by $\mathcal{C}$, the set of all n-dimensional standard basis vectors. To model Sudoku as a binary feasibility problem, we reformulate $A \in \mathbb{R}^{9\times 9}$ as a $B \in \mathbb{R}^{9\times 9\times 9}$ where

$$B[i,j,k] = \begin{cases} 1 \text{ if } A[i,j] = k, \\ 0 \text{ otherwise.} \end{cases}$$

Let S' denote the partially filled $9 \times 9 \times 9$ zero-one array representing the incomplete Sudoku, S, under the reformulation, and let $J' \subseteq I^3$ be the set of indices for which S' is filled.

The four constraints of the previous section become

$C_1 = \{B : B[i,:,k] \in \mathcal{C} \text{ for each } i,k \in I\}$,

$C_2 = \{B : B[:,j,k] \in \mathcal{C} \text{ for each } j,k \in I\}$,

$C_3 = \{B : \operatorname{vec} B[3i+1 : 3(i+1), 3j+1 : 3(j+1), k] \in \mathcal{C} \ \text{ for } i,j = 0,1,2 \text{ and } k \in I\}$,

$C_4 = \{B : B[i,j,k] = 1 \text{ for each } (i,j,k) \in J'\}$.

In addition, since each Sudoku square has precisely one entry, we require

$C_5 = \{B : B[i,j,:] \in \mathcal{C} \text{ for each } i,j \in I\}$.

A visualization of the constraints is provided in Fig. 9.

Clearly there is a one-to-one correspondence between completed integer Sudokus, and zero-one arrays contained in the intersection of the five constraint sets. Moreover, B is a completion of S' if and only if

$$B \in C_1 \cap C_2 \cap C_3 \cap C_4 \cap C_5.$$

The projections onto C_1, C_2, C_3, C_5 are given in Remark 5.1. The projection onto C_4 is given, pointwise, by

$$(P_{C_4}B)[i,j,k] = \begin{cases} S[i,j,k] & \text{if } (i,j,k) \in J', \\ B[i,j,k] & \text{otherwise;} \end{cases}$$

for each $(i,j,k) \in I^3$.

6.3 Numerical Experiments

We have tested various large suites of Sudoku puzzles on the method of Section 6.2. We give some details regarding our implementation in C++.

- *Initialize:* Set $\mathbf{x}_0 := (y,y,y,y,y) \in D$ for some random $y \in [0,1]^{9\times 9\times 9}$.
- *Iteration:* Set $\mathbf{x}_{n+1} := T_{D,C}\mathbf{x}_n$.
- *Terminate:* Either, if a solution is found, or if 10000 iterations have been performed. More precisely, if $\mathrm{round}(P_D\mathbf{x}_n)$ denotes $P_D\mathbf{x}_n$ pointwise rounded to the nearest integer, then $\mathrm{round}(P_D\mathbf{x}_n)$ is a solution if

$$\mathrm{round}(P_D\mathbf{x}_n) \in C \cap D. \tag{7}$$

Remark 6.1 In our implementation condition (7) was used a termination criterion, instead of the condition

$$P_D\mathbf{x}_n \in C \cap D.$$

This improvement is due the following observation: If $P_D\mathbf{x}_n$ is a solution then all entries are either 0 or 1. $\Diamond$

Since the Douglas–Rachford method produces a point whose projection onto D is a solution, we also consider a variant which sets

$$\mathbf{x}_{n+1} := \begin{cases} P_D T_{D,C}\mathbf{x}_n, & \text{if } n \in \{400, 800, 1600, 3200, 6400\}; \\ T_{D,C}\mathbf{x}_n, & \text{otherwise.} \end{cases}$$

We will refer to this variant as *DR+Proj*.

6.3.1 Test Library Experience

We considered Sudokus from the following libraries:

- Dukuso's `top95`[7] and `top1465`[8] – collections containing 95 and 1465 test problems, respectively. They are frequently used by programmers to test their solvers. All instances are 9×9.
- Gordon Royle's minimum Sudoku[9] – a collection containing around 50000 distinct Sudokus with 17 entries (the best known lower bound on the number of entries required for a unique solution). All instances are 9×9. Our experiments were performed on the first 1000 problems. From herein we refer to these instances as `minimal1000`.
- `reglib-1.3`[10] – a collection containing around 1000 test problems, each suited to a particular human-style solving technique. All instances are 9×9.

---

7 `top95`: http://magictour.free.fr/top95
8 `top1465`: http://magictour.free.fr/top1465
9 Gordon Royle: http://school.maths.uwa.edu.au/~gordon/sudokumin.php
10 `reglib-1.3`: http://hodoku.sourceforge.net/en/libs.php

– ksudoku16 and ksudoku25[11] – collections containing around 30 Sudokus, of various difficulties, which we generated using *KSudoku*.[12] The collections contain 16×16 and 25×25 instances, respectively.

6.3.2 Methods Used for Comparison

Our *naive* binary implementation was compared with various specialized or optimized codes. A brief description of the methods tested follows.

1. *Douglas–Rachford* in C++ – Our implementation is outlined in Section 6.3. Our experiments were performed using both the normal Douglas–Rachford method (DR) and our variant (DR+Proj).
2. *Gurobi Binary Program*[13] – Solves a binary integer program formulation using *Gurobi Optimizer 5.5*. The formulation is the same $n \times n \times n$ binary array model used in the Douglas–Rachford implementation. Our experiments were performed using the default settings, and the default settings with the presolver off.
3. *YASS*[14] (Yet Another Sudoku Solver) in C++ – Solves the Sudoku problem in two phases. In the first phase, a *reasoning algorithm* determines the possible candidates for each of the empty Sudoku squares. If the Sudoku is not completely solved, the second phase uses a deterministic recursive algorithm.

[11] ksudoku16/25: http://carma.newcastle.edu.au/DRmethods/comb-opt/

[12] KSudoku: http://games.kde.org/game.php?game=ksudoku

[13] Gurobi Sudoku model: http://www.gurobi.com/documentation/5.5/example-tour/node155

[14] YASS: http://yasudokusolver.sourceforge.net/

4. *DLX*[15] in C – Solves an exact cover formulation using the *Dancing Links* implementation of Knuth's *Algorithm X* – a non-deterministic, depth-first, backtracking algorithm.

Since YASS and DLX were only designed to be applied to 9×9 instances, their performances on `ksudoku16` and `ksudoku25` were unable to be included in the comparison.

6.3.3 Computational Results

Table 1 shows a comparison of the time taken by each of the methods in Section 6.3.2, applied to the test libraries of Section 6.3.1. Computations were performed on an Intel Core i5-3210 @ 2.50GHz running 64-bit Ubuntu 12.10. For each Sudoku puzzle, 10 replications were performed. We make some general comments about the results.

– All methods easily solved instances from `reglib-1.3` – the test library consisting of puzzles suited to human-style techniques. Since human-style technique usually avoid excessive use of 'trial-and-error', less backtracking is required to solve puzzle aimed at human players. Since all of the algorithms, except the Douglas–Rachford method, utilize some form of backtracking, this may explain the observed good performance.
– The Gurobi binary program performed best amongst the methods, regardless of the test library. Of the methods tested, the Gurobi Optimizer is the most sophisticated. Whether or not the pre-solver was used did not significantly effect computational time.

[15] DLX: `http://cgi.cse.unsw.edu.au/~xche635/dlx_sodoku/`

– Our Douglas–Rachford implementation outperformed YASS on `top95`, `top1465` and DLX on `minimal1000`. For all other algorithm/test library combinations, the Douglas–Rachford was competitive. The performance of the normal Douglas–Rachford method appears slightly better than the variant which includes the additional projection step.
– The Douglas–Rachford solved Sudoku puzzles with a high success rate – no lower than 84% for any of the test libraries. For most test libraries the success rate was much higher (see Table 2). Puzzles solved by the method were typically done so in the first 2000 iterations (see Fig. 10).

6.4 Models that Failed

To our surprise, the integer formulation of Section 6.1 was ineffective, except for 4×4 Sudoku, while the binary reformulation of the cyclic Douglas–Rachford method described in Section 2.2 also failed in both the original space and the product space.

Clearly we have a lot of work to do to understand the model characteristics which lead to success and those which lead to failure.

We should also like to understand how to diagnose infeasibility in Sudoku via the binary model. This would give a full treatment of Sudoku as a NP-complete problem.

6.5 A 'Nasty' Sudoku Puzzle and Other Challenges

The incomplete Sudoku on the left of Fig. 11 has proven intractable for Douglas–Rachford. The unique solution is shown at the right of Fig. 11. As set, it can not be

Table 1: Mean (Max) time in seconds over all instances.

	top95	top1465	reglib-1.3	minimal1000	ksudoku16	ksudoku25
DR	1.432 (6.056)	0.929 (6.038)	0.279 (5.925)	0.509 (5.934)	5.064 (30.079)	4.011 (24.627)
DR+Proj	1.894 (6.038)	1.261 (12.646)	0.363 (6.395)	0.953 (5.901)	6.757 (31.949)	8.608 (84.190)
Gurobi (default)	0.063 (0.095)	0.063 (0.171)	0.059 (0.123)	0.063 (0.091)	0.168 (0.527)	0.401 (0.490)
Gurobi (pre-solve off)	0.077 (0.322)	0.076 (0.405)	0.058 (0.103)	0.064 (0.104)	0.635 (4.621)	0.414 (0.496)
YASS	2.256 (58.822)	1.440 (113.195)	0.039 (3.796)	0.654 (61.405)	-	-
DLX	1.386 (38.466)	0.310 (34.179)	0.105 (8.500)	3.871 (60.541)	-	-

Table 2: % of Sudoku instances successfully solved.

	top95	top1465	reglib-1.3	minimal1000	ksudoku16	ksudoku25
DR	86.53	93.69	99.35	99.59	92.00	100
DR+Proj	85.47	93.93	99.31	99.59	84.67	100

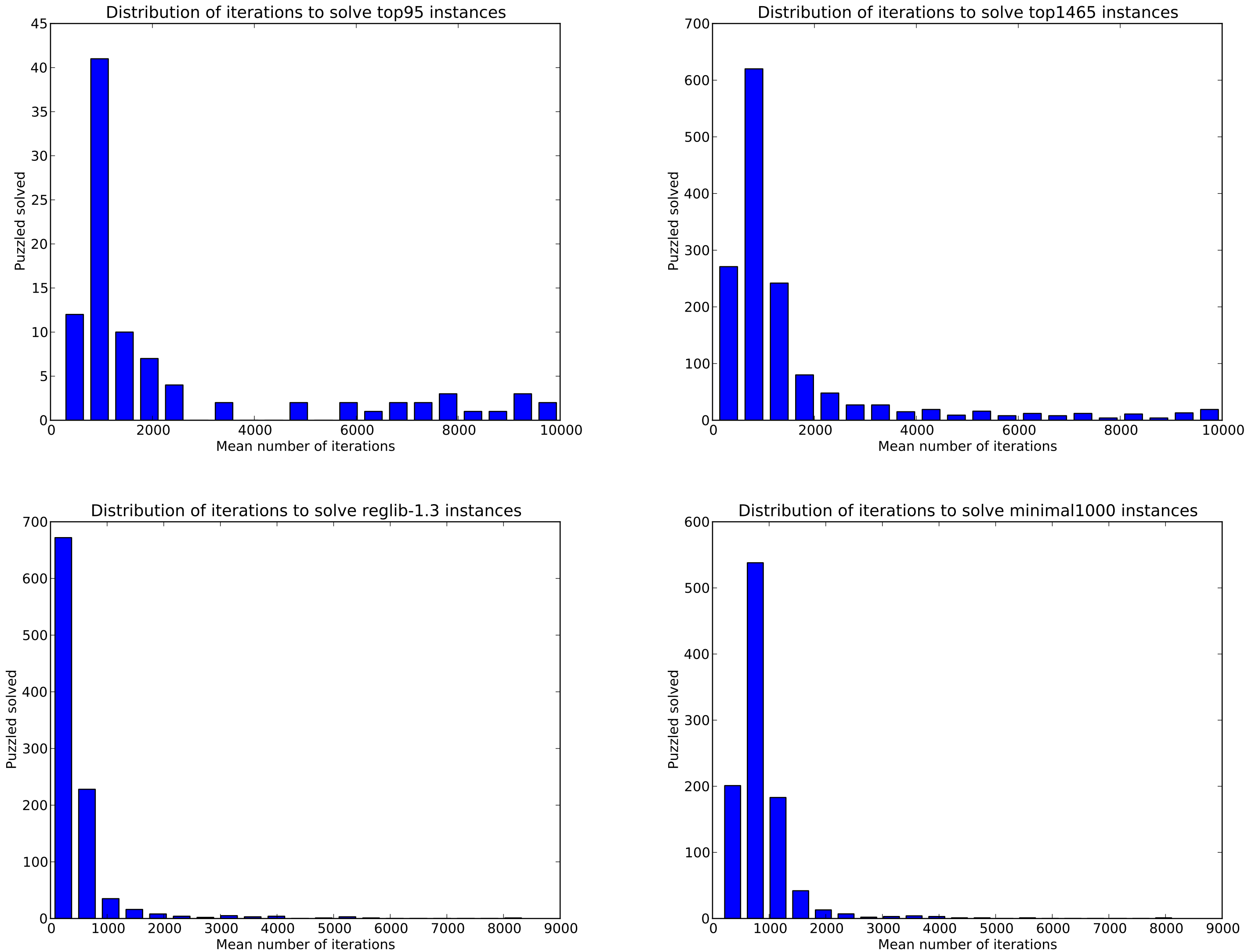


Fig. 10: Frequency histograms showing the distribution of puzzles solved by number of iterations for the Douglas–Rachford method.

7					9		5	
	1						3	
		2	3			7		
		4	5				7	
8						2		
					6	4		
	9			1				
	8			6				
		5	4					7

7	4	3	8	2	9	1	5	6
5	1	8	6	4	7	9	3	2
9	6	2	3	5	1	7	4	8
6	2	4	5	9	8	3	7	1
8	7	9	1	3	4	2	6	5
3	5	1	2	7	6	4	8	9
4	9	6	7	1	5	8	2	3
2	8	7	9	6	3	5	1	4
1	3	5	4	8	2	6	9	7

Fig. 11: The 'nasty' Sudoku (left), and its unique solution (right).

solved by Jason Schaad's Douglas–Rachford based Sudoku solver,[16] nor can it be solved reliably by our implementation. This 'nasty' Sudoku is a modified version of an example due to Veit Elser [37], who found the first puzzle which could not be solved using Douglas–Rachford methods.

We decided to ask: What happens when we remove one entry from the 'nasty' Sudoku? From one hundred random initializations:

– Removing the top-left entry, a "7", the puzzle was still difficult for the Douglas–Rachford algorithm: we had a 24% success rate — comparable to the 'nasty' Sudoku without any entries removed.
– If any other single entry was removed, the problem could be solved *fairly* reliably: we had a 99% success rate.

For each of the puzzles with an entry removed, the number of distinct solution was determined using *SudokuSolver*,[17] and are reported in Table 4. Those with an

[16] Schaad's web-based solver: `https://people.ok.ubc.ca/bauschke/Jason/`

[17] SudokuSolver: `http://infohost.nmt.edu/tcc/help/lang/python/examples/sudoku/`

Table 3: Number of instances solved from 1000 replications.

	AI escargot	'Nasty'
DR	985	202
DR+Proj	975	172

entry removed, that could be reliably solved all have many solutions — anywhere from a few hundred to a few thousand; while the puzzle with the top-left entry removed has relatively few — only five.[18] It is possible that this structure that makes the 'nasty' Sudoku difficult to solve, with the Douglas–Rachford algorithm hindered by an abundance of 'near' solutions.

We then asked: What happens when entries from the solution are added to incomplete 'nasty' Sudoku? From one hundred random starts:

- If any single entry was added, the Sudoku could be solved more often, but not reliably: we had only a 54% success rate.

We also examined how the binary Douglas–Rachford method applied to this 'nasty' Sudoku behaves relative to its behaviour on other hard problems (see Table 3). Specially, we considered *AI escargot*, a Sudoku purposely designed by Arto Inkala to be really difficult. Our Douglas–Rachford implementation could solve AI escargot *fairly* reliably: we had a success rate of 99%. In contrast to the 'nasty' Sudoku, the number of solutions to AI escargot with one entry removed was no more than a few hundred; typically much less.

[18] For the five solutions: `http://carma.newcastle.edu.au/DRmethods/comb-opt/nasty_nonunique.txt`

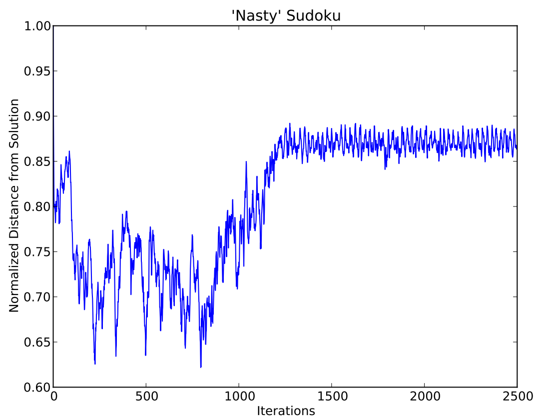


Fig. 12: Typical behaviour of the Douglas–Rachford algorithm applied to the ‘nasty’ Sudoku, modelled as a zero-one program.

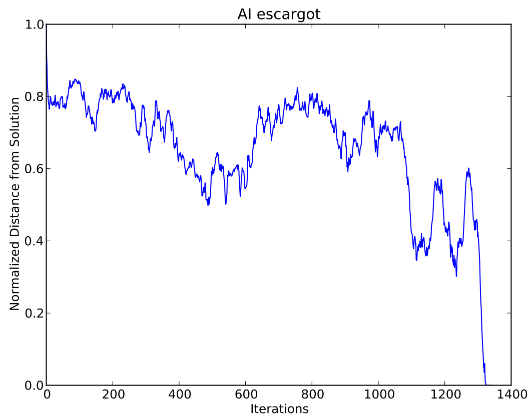


Fig. 13: Typical behaviour of the Douglas–Rachford algorithm for AI escargot, modelled as a zero-one program.

We then asked the question: How does the distances from the solution vary as a function of the number of iterations? This is plotted in Fig. 12 and 13, for

Table 4: Number of distinct solutions for the 'nasty' Sudoku with a single entry removed.

Entry removed	Distinct solutions	Entry removed	Distinct solutions
None	1	$S[5,1]$	216
$S[1,1]$	5	$S[5,7]$	2487
$S[1,6]$	571	$S[6,6]$	476
$S[1,8]$	2528	$S[6,7]$	1315
$S[2,2]$	874	$S[7,2]$	1905
$S[2,8]$	1504	$S[7,5]$	966
$S[3,3]$	2039	$S[8,2]$	711
$S[3,4]$	1984	$S[8,5]$	579
$S[3,7]$	182	$S[9,3]$	1278
$S[4,3]$	2019	$S[9,4]$	1368
$S[4,4]$	3799	$S[9,9]$	1640
$S[4,8]$	1263		

the 'nasty' Sudoku and AI escargot, respectively.[19] The same for each of the five solution to the 'nasty' Sudoku, with the top-left entry removed, is shown in Fig. 14.

In what follows, denote by $(\mathbf{x}_n)$ the sequence of iterates obtained from the Douglas–Rachford algorithm, and by $\mathbf{x}^*$ the Sudoku solution obtained from $(\mathbf{x}_n)$. In contrast to the convex setting, Fig. 12 and 13 show that the sequence $(\|\mathbf{x}_n - \mathbf{x}^*\|)$ need not be monotone decreasing.

[19] If $\mathbf{x}_n$ is the current iterate, $\mathbf{x}^*$ the solution, and $m = \max_n \|P_D\mathbf{x}_n - \mathbf{x}^*\|$, $\|P_D\mathbf{x}_n - \mathbf{x}^*\|/m$ is plotted against n.

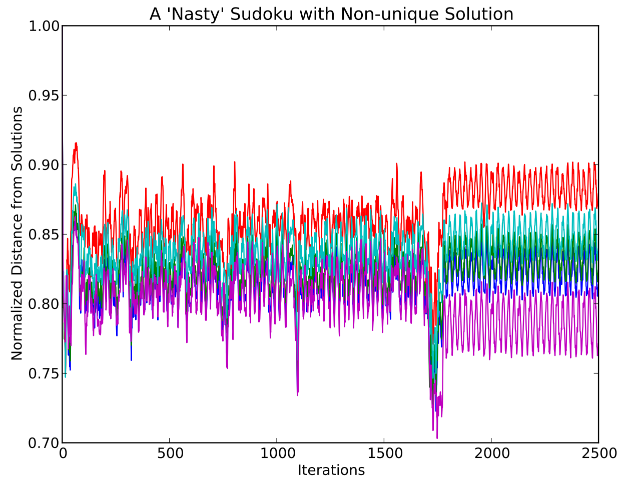


Fig. 14: Typical behaviour of Douglas–Rachford applied to 'nasty' Sudoku with top-left entry removed. The five colors represent the possible solutions.

In the convex setting, $(\mathbf{x}_n)$ is known to have the very useful property of being *Fejér monotone* with respect to $\operatorname{Fix} T_{D,C}$. That is,

$$\|\mathbf{x}_{n+1} - c\| \leq \|\mathbf{x}_n - c\| \text{ for any } c \in \operatorname{Fix} T_{D,C}.$$

When $(\mathbf{x}_n)$ converged to a solution, $\|\mathbf{x}_n - x^*\|$ decreased rapidly just before the solution was found (see Fig. 13). This seemed to occur regardless of the behaviour of earlier iterations. Perhaps this behaviour is due to the Douglas–Rachford iterate entering a local basin of attraction.

The methods Section 6.3.2, applied to the two difficult Sudoku puzzles, were also compared (see Table 5). While all solved AI escargot easily, applied to the 'nasty' Sudoku, YASS was significantly slower – the Douglas–Rachford method is not the only algorithm to find the puzzle difficult.

Table 5: Mean (Max) Time in second from 1000 replications.

	AI escargot	‘Nasty’
DR	1.232 (6.243)	4.840 (6.629)
DR+Proj	1.623 (6.074)	5.312 (7.689)
Gurobi (default)	0.157 (0.845)	0.111 (0.125)
Gurobi (pre-solve off)	0.094 (0.153)	0.253 (0.365)
YASS	0.162 (0.255)	12.370 (13.612)
DLX	0.020 (0.032)	0.110 (0.126)

7 Solving Nonograms

Recall that a nonogram puzzle consists of a blank $m \times n$ grid of pixels (the canvas) together with $(m+n)$ cluster-size sequences, one for each row and each column [38]. The goal is to paint the canvas with a picture that satisfies the following constraints:

– Each pixel must be black or white.
– If a row (resp. column) has cluster-size sequence $s_1, s_2, \ldots, s_k$ then it must contain k clusters of black pixels, separated by at least one white pixel, such that the ith leftmost (resp. uppermost) cluster contains s_i black pixels.

An example of a nonogram puzzle is given in Fig. 15. Its solution, found by the Douglas–Rachford algorithm, is shown in Fig. 17.

We model nonograms as a binary feasibility problem. The $m \times n$ grid is represented as a matrix $A \in \mathbb{R}^{m\times n}$. We define

$$A[i,j] = \begin{cases} 0 \text{ if the } (i,j)\text{-th entry of the grid is white,} \\ 1 \text{ if the } (i,j)\text{-th entry of the grid is black.} \end{cases}$$

								1			
					2			**4**	**1**	**2**	**2**
		2	**3**	**1**	**1**	**5**	**4**	**1**	**5**	**2**	**1**
1	**2**										
	2										
	1										
	1										
	2										
2	**4**										
2	**6**										
	8										
1	**1**										
2	**2**										

Fig. 15: A nonogram whose solution can be found by Douglas–Rachford (see Fig. 17). Cluster-size sequences for each row and column are given.

Let $\mathcal{R}_i \subset \mathbb{R}^m$ (resp. $\mathcal{C}_j \subset \mathbb{R}^n$) denote the set of vectors having cluster-size sequences matching row i (resp. column j).

$$C_1 = \{A : A[i,:] \in \mathcal{R}_i \text{ for } i = 1, \ldots, m\},$$

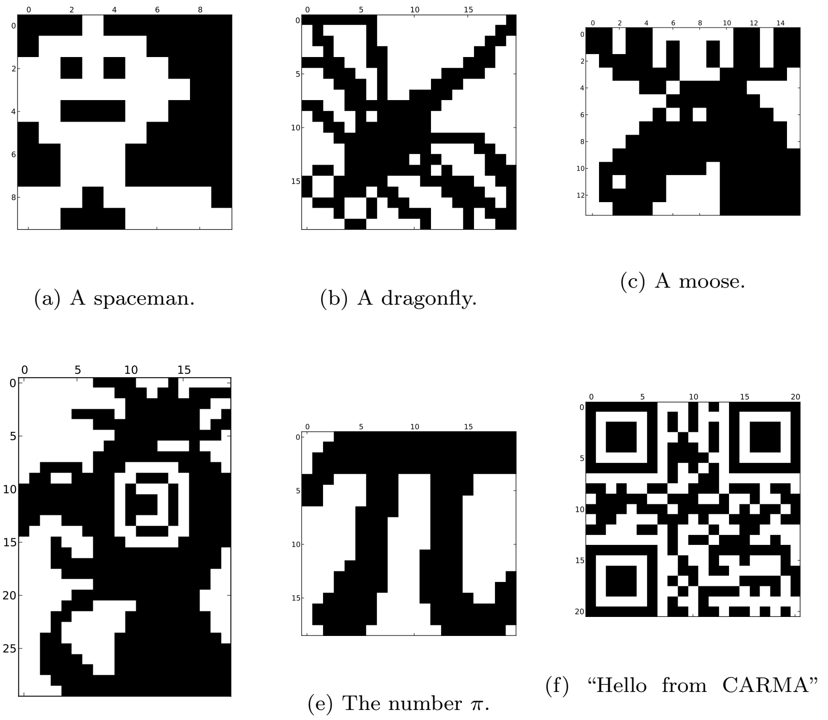
$$C_2 = \{A : A[:,j] \in \mathcal{C}_j \text{ for } j = 1, \ldots, n\}.$$

Given an incomplete nonogram puzzle, A is a solution if and only if

$$A \in C_1 \cap C_2.$$

We investigated the viability of the Douglas–Rachford method to solve nonogram puzzles, by testing the algorithm on seven puzzles: the puzzle in Fig. 15, and the six puzzles shown in Fig. 16. Our implementation, written in Python, is, appropriately modified, the same as the method of Section 6.3.

Applied to nonograms, the Douglas–Rachford algorithm is highly successful. From 1000 random initializations, all puzzles considered were solved with a 100% success rate.



(a) A spaceman.

(b) A dragonfly.

(c) A moose.

(d) A parrot.

(e) The number π.

(f) "Hello from CARMA" encoded as a QR code.[20]

Fig. 16: Solutions to six nonograms found by the Douglas–Rachford algorithm.

Within this model, a difficulty is that the projections onto C_1 and C_2 have no simple form. So far, our attempts to find an efficient method to do so have been unsuccessful. Our current implementation pre-computes $\mathcal{R}_i$ and $\mathcal{C}_j$, for all indices i, j, and at each iteration chooses the nearest point by computing the distance to each point in the appropriate set.

For nonograms with large canvases, the enumeration of $\mathcal{R}_i$ and $\mathcal{C}_j$ becomes intractable. However, the Douglas–Rachford iterations themselves are fast.

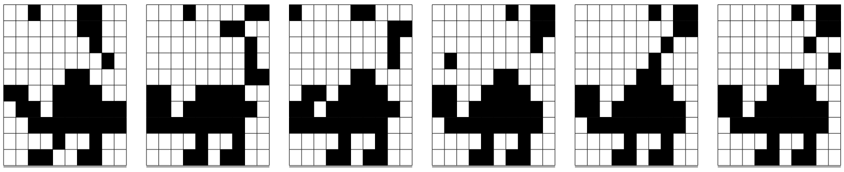


Fig. 17: Solution to the nonogram in Fig. 15 found by Douglas–Rachford in six iterations: showing the projection onto C_1 of these six iterations.

Remark 7.1 (Performance on NP-complete problems) We note that for Sudoku, the computation of projections is easy but the typical number of (easy) iterative steps large—as befits an NP complete problem. By contrast for nonograms, the number of steps is very small but an exponential amount of work is presumably buried in computing the projections. ♢

8 Conclusions

The message of the list in Section 4.2 and of the previous two sections is the following. When presented with a new combinatorial feasibility problem it is well worth seeing if Douglas–Rachford can deal with it—it is conceptually very simple and is usually relatively easy to implement. It would be interesting to apply Douglas–Rachford to various other classes of problems.

Moreover, this approach allows for the intuition developed in Euclidean space to be usefully repurposed. This lets one profitably consider non-expansive fixed point methods in the class of CAT(0) metric spaces — a far ranging concept

[20] *QR (quick response) codes* are two-dimensional bar codes originally designed for use in the Japanese automobile industry. Their data is typically encoded in either numerical, alphanumerical, or binary formats.

introduced twenty years ago in algebraic topology but now finding applications to optimization and fixed point algorithms. The convergence of various projection type algorithms to feasible points is under investigation by Searston and Sims among others in such spaces [39]: thereby broadening the constraint structures to which projection-type algorithms apply to include metrically rather than only algebraically convex sets.

Weak convergence of project-project-average has been established [39]. Reflections have been shown to be well defined in those CAT(0) spaces with extensible geodesics and curvature bounded below [40]. Examples have been constructed to show that unlike in Hilbert spaces they need not be nonexpansive unless the space has constant curvature [40]. None-the-less it appears that the basic Douglas–Rachford algorithm (reflect-reflect-average) may continue to converge in fair generality.

Many resources can be found at the paper's companion website:

http://carma.newcastle.edu.au/DRmethods/comb-opt/

Acknowledgements We wish to thank Heinz Bauschke, Russell Luke, Ian Searston and Brailey Sims for many useful insights. Example 2.1 was provided by Brailey Sims.

References

1. Douglas, J., Rachford, H.: On the numerical solution of heat conduction problems in two and three space variables. Trans. Am. Math. Soc. **82**(2), 421–439 (1956)
2. Lions, P., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. SIAM J. Numer. Anal. pp. 964–979 (1979)
3. Combettes, P.: Solving monotone inclusions via compositions of nonexpansive averaged operators. Optim. **53**(5–6), 475–504 (2004)

4. Borwein, J., Zhu, Q.: Techniques of variational analysis. CMS books in mathematics. Springer, New York (2005)
5. Borwein, J., Sims, B.: The Douglas–Rachford algorithm in the absence of convexity. In: Fixed-Point Algorithms for Inverse Problems in Science and Engineering, pp. 93–109. Springer (2011)
6. Bauschke, H., Combettes, P., Luke, D.: Finding best approximation pairs relative to two closed convex sets in Hilbert spaces. J. Approx. Theory **127**(2), 178–192 (2004)
7. Bauschke, H., Borwein, J.: On the convergence of von Neumann's alternating projection algorithm for two sets. Set-Valued Anal. **1**(2), 185–212 (1993)
8. Combettes, P., Pesquet, J.: A Douglas–Rachford splitting approach to nonsmooth convex variational signal recovery. IEEE J. Sel. Top. Signal Process. **1**(4), 564–574 (2007)
9. Gandy, S., Yamada, I.: Convex optimization techniques for the efficient recovery of a sparsely corrupted low-rank matrix. J. Math-for-Industry **2**(5), 147–156 (2010)
10. Steidl, G., Teuber, T.: Removing multiplicative noise by Douglas–Rachford splitting methods. J. Math. Imaging Vis. **36**(2), 168–184 (2010)
11. Svaiter, B.F.: On weak convergence of the Douglas-Rachford method. SIAM J. Control Optim. **49**(1), 280–287 (2011)
12. Yamada, I., Gandy, S., Yamagishi, M.: Sparsity-aware adaptive filtering based on a Douglas–Rachford splitting. In: Proc. EUSIPCO, pp. 1929–1933 (2011)
13. Bauschke, H., Combettes, P.: Convex Analysis and Monotone Operator Theory in Hilbert Spaces. Springer, New York (2011)
14. Opial, Z.: Weak convergence of the sequence of successive approximations for nonexpansive mappings. Bull. Amer. Math. Soc. **73**(4), 591–597 (1967)
15. Borwein, J., Tam, M.: A cyclic Douglas-Rachford iteration scheme. J. Optim. Theory Appl. (2013). Doi: 10.1007/s10957-013-0381-x
16. Bauschke, H., Matoušková, E., Reich, S.: Projection and proximal point methods: convergence results and counterexamples. Nonlinear Anal.: Theory, Methods, and Appl. **56**(5), 715–738 (2004)
17. Borwein, J., Vanderwerff, J.: Convex functions: constructions, characterizations and counterexamples. Encyclopedia of Mathematics and its Applications. Cambridge University

Press (2010)

18. Aragón Artacho, F., Borwein, J.: Global convergence of a non-convex Douglas–Rachford iteration. J. Glob. Optim. pp. 1–17 (2012). Doi: 10.1007/s10898-012-9958-4
19. Hesse, R., Luke, D.: Nonconvex notions of regularity and convergence of fundamental algorithms for feasibility problems. preprint http://arxiv.org/pdf/1205.0318v1 (2012)
20. Bauschke, H., Luke, D., Phan, H., Wang, X.: Restricted normal cones and sparsity optimization with affine constraints. Found. Comput. Math. pp. 1–21 (2012). Doi: 10.1007/s10208-013-9161-0
21. Babu, P., Pelckmans, K., Stoica, P., Li, J.: Linear systems, sparse solutions, and Sudoku. IEEE Signal Process. Lett. **17**(1), 40–42 (2010)
22. Aragón Artacho, F., Borwein, J., Tam, M.: Douglas–Rachford feasibility methods for matrix completion problems. preprint http://arxiv.org/abs/1308.4243 (2013)
23. Elser, V., Rankenburg, I.: Deconstructing the energy landscape: constraint-based algorithms for folding heteropolymers. Phys. Rev. E **73**(2), 026,702 (2006)
24. Elser, V., Rankenburg, I., Thibault, P.: Searching with iterated maps. Proc. Natl. Acad. Sci. **104**(2), 418–423 (2007)
25. Bauschke, H., Combettes, P., Luke, D.: Phase retrieval, error reduction algorithm, and Fienup variants: a view from convex optimization. J. Opt. Soc. Am. A **19**(7), 1334–1345 (2002)
26. Bauschke, H., Combettes, P., Luke, D.: Hybrid projection–reflection method for phase retrieval. J. Opt. Soc. Am. A **20**(6), 1025–1034 (2003)
27. Johnson, C.: Matirx completion problems: A survey. Proc. Sympos. Appl. Math. pp. 171–198 (1990)
28. Schaad, J.: Modeling the 8-queens problem and sudoku using an algorithm based on projections onto nonconvex sets. Master's thesis, Univ. British Columbia (2010)
29. Gravel, S., Elser, V.: Divide and concur: A general approach to constraint satisfaction. Phys. Rev. E **78**(3), 036,706 (2008)
30. Garey, R., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. A Series of Books in the Mathematical Sciences. W. H. Freeman (1979)
31. Takenaga, Y., Walsh, T.: Tetravex is NP-complete. Inf. Process. Lett. **99**, 171–174 (2006)

32. Bansal, P.: Code for solving Tetravex using Douglas–Rachford algorithm. http://people.ok.ubc.ca/bauschke/Pulkit/pulkitreport.pdf (2010)
33. Takayuki, Y., Takahiro, S.: Complexity and completeness of finding another solution and its application to puzzles. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. (Inst. Electron. Inf. Commun. En.) **86**(5), 1052–1060 (2003)
34. Nagao, T., Ueda, N.: NP-completeness results for nonogram via parsimonious reductions. Tech. Rep. TR96-0008, Depart. Computer Science, Tokyo Inst. Technol. (2012). CiteSeerX: doi=10.1.1.57.5277
35. van Rijn, J.: Playing games: The complexity of Klondike, Mahjong, nonograms and animal chess. Master's thesis, Leiden Inst. of Advanc. Computer Science, Leiden Univ. (2012)
36. Hardy, G., Littlewood, J., Pólya, G.: Inequalities. Cambridge Mathematical Library. Cambridge University Press (1952)
37. Elser, V.: private communication, August 27th (2012)
38. Bosch, R.: Painting by numbers. Optima **65**, 16–17 (2001)
39. Bačák, M., Searston, I., Sims, B.: Alternating projections in CAT(0) spaces. J. Math. Anal. Appl. **385**(2), 599–607 (2012)
40. Searston, I., Sims, B.: Nonlinear analysis in geodesic metric spaces, in particular CAT(0) spaces. preprint (2013)