Constraints (2007) 12:179–205
DOI 10.1007/s10601-006-9014-4

Design of Financial CDO Squared Transactions Using Constraint Programming

Pierre Flener · Justin Pearson · Luis G. Reyna · Olof Sivertsson

Published online: 12 January 2007

Abstract We give an approximate and often extremely fast method of building a particular kind of portfolio in finance, here called a portfolio design (PD), with applications in the credit derivatives market, for example when designing collateralised debt obligations squared (CDO^2) transactions. A PD generalises a balanced incomplete block design (BIBD) and is usually harder to build. Worse, typical financial PDs are an order of magnitude larger than the largest BIBDs built so far by constraint programs, and in practice an optimisation version of the problem of building PDs has to be solved. Our method is based on embedding small designs, whose determination is itself a constraint satisfaction problem, into the original large design. Together with the detection of when a PD might be a BIBD, symmetry breaking, extended reuse of previously built PDs, and admissibility checking during search, the performance of the method becomes good enough for designing (near-)optimal CDO^2 transactions, with sizes common in the credit derivatives market, within minutes. For example, we optimally build a typical financial PD, which has over 10^{746} symmetries, in just a few minutes. The high quality of our approximate designs can be assessed by comparison with a lower bound on the optimum. Our designs sufficiently improve the currently best ones so as often to make the difference

P. Flener (✉) · J. Pearson · O. Sivertsson
Department of Information Technology, Uppsala University,
Box 337, SE-751 05 Uppsala, Sweden
e-mail: PierreF@it.uu.se

J. Pearson
e-mail: Justin@it.uu.se

O. Sivertsson
e-mail: OSivertsson@gmail.com

L. G. Reyna
Swiss Re Financial Products, Swiss Re, Park Avenue Plaza,
New York, NY 10055, USA
e-mail: Luis_Reyna@SwissRe.com

between having and not having a feasible CDO^2 transaction due to investor and rating-agency constraints.

Keywords Financial mathematics · Credit derivatives · Collateralised debt obligation (CDO) · CDO squared · Portfolio design (PD) · Optimal portfolio design (OPD) · Balanced incomplete block design (BIBD) · Constraint programming · Embedding

1 Introduction

The structured credit market has seen two important new products over the last decade: credit derivatives and collateralised debt obligations (CDOs). These new products have created the ability to leverage and transform credit risk in ways not possible through the traditional bond and loan markets.

CDOs typically consist of a special-purpose vehicle that has credit exposure to around one hundred different issuers. Such vehicles purchase bonds and loans and other financial assets through the issuance of notes or obligations with varying levels of risk. In a typical structure, credit losses in the underlying pool are allocated to the most subordinated obligations or notes first. A natural progression of the market has been to use notes from existing CDOs as assets into a new generation of CDOs, called CDO Squared (CDO^2) or CDO of CDO [12].

The credit derivatives market has allowed a more efficient mechanism for creating CDO^2. The idea is to use tranches of credit default swaps instead of notes. The tranches are chosen from a collection of credits with the level of liquidity and risk adequate to the potential investors. These transactions are sometimes labelled synthetic CDO^2.

In the creation of a synthetic CDO^2, the natural question arises on how to maximise the diversification of the tranches given a limited universe of previously chosen credits. In a typical CDO^2, the number of available credits ranges from 250 to 500 and the number of tranches from four to as many as 25. The investment banker arranging for a CDO^2 usually seeks to maximise the return of the subordinated notes under the constraints imposed by the rating agencies and the investors. This is a challenge that typically is only partially addressed, in part due to the difficulty of pricing the underlying assets [6].[1]

In this paper, we analyse the financially relevant abstracted problem of selecting the credits comprising each of the tranches with a minimal overlap, or maximum diversification. The minimisation of the overlap usually results in better ratings for the notes, typically resulting in more efficient structures. The contributions and significance of this paper are as follows:

– We introduce a *new method of building portfolio designs* to the finance world, with practical applications in the credit derivatives market, such as the design of CDO^2 transactions. The method is fully automated, often extremely fast, and

[1]There are very few publicly accessible papers we can cite in this introduction, as most are confidential due to the potential financial value of their results.

builds designs that are as close to the optimum as one is willing to wait for. It improves on the usual method of ad hoc manual permutations.

- We introduce portfolio designs (PDs) as a *new benchmark problem* and a *new successful technology transfer* to the constraint programming community.
- We present the *new concept of design embeddings* by generalising the well-known notion of design multiples, and successfully apply it to solve large PD instances (near-)optimally.
- We significantly *improve the run-times and quality of our previous results* in [11], where the discussed problem was actually originally introduced, using the new theoretical and modelling results of [16].

The remainder of this paper is organised as follows. In Section 2, we present PDs first in their financial domain and then in an abstracted way, while introducing necessary theoretical background and results. Next, in Section 3, we present a sophisticated method, implemented as a constraint program, for exactly building a PD by global search. Since this method does not scale for the solution of typical financial instances of the optimisation version of the problem of building PDs, we introduce in Section 4 a method of approximately building such larger designs, using a notion of embedding occurrences of smaller designs in a larger one. The determination of the small designs is itself a constraint satisfaction problem, and they are built using the method of Section 3. Finally, in Section 5, we conclude, discuss related work, and outline future work.

All tests were done using SICStus Prolog 3.12.3 on Debian GNU/Linux running Linux 2.4.18 on an AMD AthlonXP 2400+ CPU with 256MB RAM.

2 Portfolio Designs

After giving a brief introduction to the financial background of our portfolio designs (PDs) in Section 2.1, we recall balanced incomplete block designs (BIBDs) in Section 2.2. BIBDs are a special case of PDs and we will sometimes take advantage of this when building PDs. We present admissibility conditions for the BIBD parameters in Section 2.3 and discuss the symmetries of the usual model for BIBDs in Section 2.4. With this background, we can then introduce PDs in Section 2.5, present an admissibility condition for their parameters in Section 2.6, and discuss the symmetries and other difficulties of a straightforward model for PDs in Section 2.7. Finally, we explain the rest of the journey of this paper in Section 2.8.

2.1 Brief Financial Background

Credit derivatives are used in finance to transfer the risk of a specified financial event happening to a credit asset without transferring the asset itself. A *credit default swap* (CDS) is a type of insurance for the holder of a financial asset against some specified financial event: the protection buyer does regular payments to the protection seller in exchange for a payment if that event occurs. Let us illustrate this with an example.

Example 1 Consider a small bank that six months ago issued a 3-year-bond of \$1,000,000 to a new IT business, which pays 10% interest per year. The bank has now become worried that the IT business may not be able to pay back its debt. Therefore the bank decides to buy protection against this in the form of a CDS from a derivative issuer. The bank pays 8% of \$1,000,000, i.e., \$80,000, divided into quarterly payments of \$20,000 to the derivative issuer. In exchange for this, the derivative issuer refunds the small bank for its \$1,000,000 should the IT business not be able to redeem the bond (perhaps because of bankruptcy) and the quarterly payments stop. Otherwise the bank is redeemed by the IT business as if the CDS had not existed, but the derivative issuer has also made some money at the expense of the bank.

CDSs are the most common credit derivative and naturally the issuers of CDSs want to minimise their risk by selling on their CDSs. Just like ordinary stock shares can be lumped together into mutual funds, CDSs can be lumped together into baskets consisting of different CDSs (with varying risk).

Collateralised debt obligations (CDOs) are credit derivatives that consist of a large number of other credit derivatives. There are *cashflow CDOs*, where the underlying credit derivatives are bonds or loans, and *synthetic CDOs*, where the underlying credit derivatives are CDSs.

A CDO is divided into tranches (baskets), each consisting of a fixed-sized subset of the underlying credit derivatives. Since different tranches consist of different bonds and loans or CDSs, they have different risks (and potential returns) associated with them. There are also *CDO-squared* (CDO^2), where the underlying credit derivatives are also CDO tranches. They are hard to risk-analyse because the same underlying credit derivative may be available in many tranches.

An investor can choose to invest in a tranche from a CDO that matches the wanted risk. But sometimes an investor may want to put a part of his money into a high-risk tranche and another part into a low- or medium-risk tranche. The issuer of a CDO still wants the investor to invest as much as possible in the issuer's CDO instead of looking for another issuer's CDO.

To be able to convince the investor of this, no two tranches should overlap more than absolutely necessary, because the investor wants to spread his risks, otherwise he could invest in just one tranche instead. Hence the question arises of how a CDO issuer should design a CDO to minimise the overlap between any two tranches. This is the problem we address in this paper.

2.2 Balanced Incomplete Block Designs (BIBDs)

Balanced incomplete block designs (BIBDs) [3, 13] are extensively studied in combinatorial design theory and are used within statistical-experiment design theory, the study of finite geometries, as well as the construction of error-correcting codes. We study BIBDs because a portfolio design is a generalisation of a BIBD.

Definition 1 Let $V = \{1, \dots, v\}$ be any set of v elements, called *varieties*. Let $B = \{1, \dots, b\}$. A *balanced incomplete block design (BIBD)* $\langle v, b, r, k, \lambda\rangle$ consists of b sets $B_1, \dots, B_b$, called *blocks*, each being a k-element subset of V, with $2 \leq k < v$,[2] such that each pair of distinct varieties occurs together in exactly λ blocks. This is called the *balancing condition* and we here call λ the *overlap*.

[2]If $k = v$, then it is a *complete* block design.

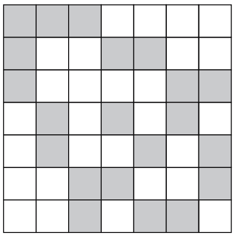

Fig. 1 Incidence matrix for the BIBD $\langle 7, 7, 3, 3, 1\rangle$. *Columns* represent blocks, *rows* represent co-blocks, *grey cells* represent $M_{ij} = 1$, and *white cells* represent $M_{ij} = 0$

Let V_i be the set of the indices of the blocks in which variety i occurs: $V_i = \{j \in B \mid i \in B_j\}$. The V_i are here called *co-blocks* and all are r-element subsets of B. The balancing condition can then be reformulated by requiring that any two co-blocks with distinct indices share exactly λ elements. Formally:

$$\forall j \in B : B_j \subseteq V \tag{1}$$

$$\forall i \in V : V_i \subseteq B \tag{2}$$

$$\forall j \in B : |B_j| = k \tag{3}$$

$$\forall i \in V : |V_i| = r \tag{4}$$

$$\forall i \neq j \in V : |V_i \cap V_j| = \lambda \tag{5}$$

One way of modelling a BIBD is in terms of its *incidence matrix*, which is a $v \times b$ matrix M, such that the entry M_{ij} at the intersection of row i and column j is 1 if $i \in B_j$ (that is $j \in V_i$) and 0 otherwise. Hence blocks are represented by columns and co-blocks by rows. The constraints (1) and (2) are then unnecessary to state. The constraints (3), (4) and (5) are then modelled by requiring, respectively, that there are exactly k ones (that is a sum of k) for each column, exactly r ones (that is a sum of r) for each row, and a scalar product of exactly λ for any pair of rows with distinct indices. Figure 1 shows an incidence matrix for the BIBD $\langle 7, 7, 3, 3, 1\rangle$.

2.3 Admissibility Conditions for BIBDs

There are some interesting properties of BIBDs that we will find useful later in this paper. Indeed, not all values of the parameters v, b, r, k, and λ result in BIBDs; actually very few do. The following three conditions are necessary for the parameters to be able to represent a BIBD:

$$\lambda < r \tag{6}$$

$$vr = bk \tag{7}$$

$$r(k-1) = \lambda(v-1) \tag{8}$$

Condition (6) says that none of the co-blocks can be equal, while condition (7) is an application of double counting: the co-blocks and blocks have together the same number of elements. Condition (8) is also an application of double counting since the sum of the cardinalities of the intersections between any co-block V_i and all other co-blocks can be computed in two ways: either by stating that V_i intersects with cardinality λ with each of the other $v - 1$ co-blocks, or by summing the cardinalities

k of the r blocks that contain elements of V_i but subtracting the r elements of V_i itself. When the parameters satisfy these conditions, they are said to be *admissible*. A consequence of the BIBD conditions and admissibility conditions is that any three of the five BIBD parameters are independent.

But just because the parameters are admissible does unfortunately not always result in a BIBD. An example of this is $v = 22 = b, r = 7 = k, \lambda = 2$, which are admissible values, and yet there does not exist a BIBD with these parameters. The problem is that the two equalities (7) and (8) do not capture the fixed overlap between any two co-blocks with distinct indices, but only the *sum* of the overlaps between a particular co-block and *all* other co-blocks. The pairwise overlap between any two co-blocks with distinct indices is captured by the product of the incidence matrix M and its transpose M^T:

$$MM^T = \begin{pmatrix} r & \lambda & \lambda & \cdots & \lambda \\ \lambda & r & \lambda & \cdots & \lambda \\ \lambda & \lambda & r & \cdots & \lambda \\ \cdot & \cdot & \cdot & r & \cdot \\ \lambda & \lambda & \lambda & \cdots & r \end{pmatrix}$$

From this matrix, Fisher's inequality for BIBDs can be derived:

Proposition 1 (*Fisher* [13]) *For any BIBD $\langle v, b, r, k, \lambda \rangle$ we must have*

$$b \geq v \tag{9}$$

Also, when the incidence matrix M is square, that is when $v = b$, the determinant of MM^T and Lagrange's four-squares theorem can be used to prove the following proposition:

Proposition 2 (*Bruck-Ryser-Chowla* [13]) *For any BIBD $\langle v, b, r, k, \lambda \rangle$ with $v = b$, we must have*

$$\text{for } v \text{ even: } \exists x \in \mathbb{Z} : x^2 = k - \lambda \tag{10}$$

$$\text{for } v \text{ odd: } \exists x, y, z \in \mathbb{Z} : z^2 = (k-1)x^2 + (-1)^{(v-1)/2}\lambda y^2, \text{ with } x \neq 0 \tag{11}$$

Using this last proposition, we see that no BIBD $\langle 22, 22, 7, 7, 2 \rangle$ can exist. Still, all these conditions are only necessary and not sufficient, so we can use them only to refute parameters that cannot result in a BIBD, but when all these conditions hold we still do not know if the BIBD exists or not. An example of this is the BIBD $\langle 111, 111, 11, 11, 1 \rangle$, whose parameters fulfil all of the conditions above, including (11) (with $x = 1 = y$ and $z = 3$), but no BIBD with these parameters exists.

Sufficient conditions for the existence of a BIBD are unfortunately not known. All information from the matrix MM^T needs to be captured in conditions with v, b, r, k, and λ to get sufficient conditions for the existence of BIBDs, not just projections, such as the determinant, which is used in the propositions above.

In the remainder of this paper, the three conditions of the previous two propositions are also referred to as admissibility conditions.

2.4 Symmetries of BIBDs

Since the varieties and block identifiers are indistinguishable, any two rows or columns of the incidence matrix can be freely permuted. Breaking all the resulting $v!b!$ symmetries can in theory be performed, for instance by posting $v!b! - 1$ ordering constraints [5]. In practice, breaking just some of those symmetries by posting just some of those ordering constraints works quite fine [9]. By simultaneously performing symmetry-breaking during search [8], but augmenting it with group-theoretical insights and some heuristics, improvements of another order of magnitude can be achieved, but only when computing all the designs [15], whereas we are here interested only in the first design. The designs built in [15] with $4 \leq v \leq 25$, which is the range of interest to us, have values of b up to 50, which is an order of magnitude below our range of interest.

2.5 Portfolio Designs (PDs)

In financial terms, a portfolio design (PD) is a CDO or CDO^2 transaction with v tranches, each consisting of r credit assets (CDSs or bonds and loans) out of a set of b credit assets, such that any two tranches with distinct indices share at most λ credit assets [11]. The choice of symbols comes from the close relationship to BIBDs. There is a universe of about $250 \leq b \leq 500$ credit assets. A typical portfolio contains about $4 \leq v \leq 25$ tranches, each of size $r \approx 100$. We now define PDs in abstract terms.

Definition 2 Let $V = \{1, \ldots, v\}$ be any set of v elements, called *varieties*. Let $B = \{1, \ldots, b\}$. A *portfolio design (PD)* $\langle v, b, r, \lambda \rangle$ consists of v sets $V_1, \ldots, V_v$, called *co-blocks*, each being an r-element subset of B, such that any two co-blocks with distinct indices share at most λ elements. This is called the *balancing condition* and we call λ the *maximum overlap*.

Let B_j be the set of varieties such that block index j occurs in co-block V_i: $B_j = \{i \in V \mid j \in V_i\}$. The B_j are called *blocks* and are all arbitrary-sized subsets of B. Formally:

$$\forall j \in B : B_j \subseteq V \tag{12}$$

$$\forall i \in V : V_i \subseteq B \tag{13}$$

$$\forall j \in B : |B_j| \leq v \tag{14}$$

$$\forall i \in V : |V_i| = r \tag{15}$$

$$\forall i \neq j \in V : |V_i \cap V_j| \leq \lambda \tag{16}$$

The differences with a BIBD are that any two co-blocks with distinct indices share *at most*, rather than exactly, λ elements, and that the cardinalities of the blocks can be *anything* in the range $0, \ldots, v$, rather than exactly some value k. This means that the admissibility conditions of BIBDs are in general not applicable to PDs. In particular, unlike in (6), the co-blocks can be equal, hence $\lambda \leq r$. PDs with $\lambda = r$ are trivial to build, as it suffices to make all the co-blocks equal. Being thus a generalisation of BIBDs, some PDs are actually BIBDs. This occurs when b divides rv and any two co-blocks with distinct indices share exactly λ elements. We will exploit this very successfully (in Section 3.3).

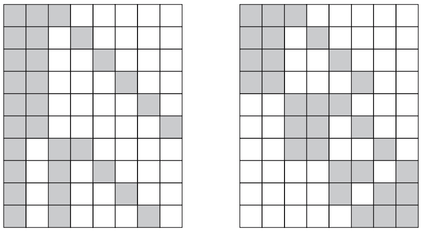
Fig. 2 Two incidence matrices for the PD $\langle 10, 8, 3, 2\rangle$. *Columns* represent blocks, *rows* represent co-blocks, *grey cells* represent $M_{ij} = 1$, and *white cells* represent $M_{ij} = 0$

Finding a PD $\langle v, b, r, \lambda\rangle$ is a constraint *satisfaction* problem. In practice, only the parameters v, b, and r are known, and a PD with a *minimal* value for λ is to be found, which is a constraint *optimisation* problem.

Definition 3 Let $V = \{1, \ldots, v\}$ be any set of v elements, called *varieties*. Let $B = \{1, \ldots, b\}$. An *optimal portfolio design* (*OPD*) $\langle v, b, r, \lambda\rangle$ consists of v sets $V_1, \ldots, V_v$, called *co-blocks*, each being an r-element subset of B, such that any two co-blocks with distinct indices share at most λ elements, where λ is minimal.

Just like a BIBD, an (O)PD is naturally modelled using its $v \times b$ incidence matrix, which has $v!b!$ symmetries. The constraints (12), (13) and (14) are then unnecessary to state. The constraints (15) and (16) are then modelled by requiring, respectively, that there are exactly r ones (that is a sum of r) for each row, and a scalar product of at most λ for any pair of rows with distinct indices. Figure 2 shows two incidence matrices for the PD $\langle 10, 8, 3, 2\rangle$.

2.6 Admissibility Conditions for PDs

We give a lower bound on λ, the maximum number of shared elements between any two co-blocks with distinct indices in a PD. This lower bound is also applicable to BIBDs, where its specialisation when b divides rv can be derived using the BIBD admissibility conditions (7) and (8).

Theorem 1 *Let $V_1, \ldots, V_v$ be r-element sets and B be their union, with $b = |B|$. If $|V_i \cap V_j| \leq \lambda$ for all $i \neq j$, then*[3]

$$\lambda \geq \frac{\left\lceil \frac{rv}{b} \right\rceil^2 \bmod (rv, b) + \left\lfloor \frac{rv}{b} \right\rfloor^2 (b - \bmod(rv, b)) - rv}{v(v-1)} \tag{17}$$

The proof is long and beyond the scope of this paper, but can be found in [16]. This lower bound is tighter than the one we used in [11] and is equal to it only when b divides rv. Furthermore, the expression of this new lower bound is never negative, unlike the previous one, which is negative when $b > rv$, that is when more elements are available than needed. This is what suggested that a tighter lower bound ought to exist. It is an open question whether there is a tighter lower bound that is easy to compute.

[3]The result of $\bmod(a, b)$ is the integer leftover when doing the integer division a/b.

The tighter lower bound means that we can be sure that no design exists for some PDs that were an open question previously.

Example 2 Consider the OPD $\langle 10, 350, 100\rangle$: Theorem 1 gives $\lambda \geq 21.\bar{1}$ (and a design does exist with $\lambda = 22$, see Section 4) whereas the bound of [11] only gives $\lambda \geq 20.63$. Hence we can now be sure no design with $\lambda = 21$ exists, which is a claim that required a separate proof previously (I.P. Gent and N. Wilson, October 2004, personal communications to J. Pearson).

However, even this tighter lower bound is not always exact.

Example 3 Consider the OPD $\langle 10, 8, 3\rangle$: using the bound of [11] we get $\lambda \geq 0.91\bar{6}$ while Theorem 1 gives $\lambda \geq 0.9\bar{3}$. Consider also the OPD $\langle 9, 8, 3\rangle$: using the bound of [11] we get $\lambda \geq 0.890625$ while Theorem 1 gives $\lambda \geq 0.91\bar{6}$. However, it is not difficult to show (with the method to be presented in Section 3) that there are no ten or even nine subsets of size 3 in an eight-element set that such that any two of them share at most $\lambda = 1$ element. In fact, these two OPDs are at best built with $\lambda = 2$; some of the co-blocks of such optimal designs share only one element, as can be seen in Fig. 2. (This example will be continued in Example 4.)

It is tempting to think that bounds can be similarly obtained on the block sizes. Indeed, a PD $\langle v, b, r, \lambda\rangle$ becomes a BIBD if b divides rv and any two co-blocks with distinct indices share *exactly* (rather than at most) λ elements: the integer value $k = \frac{rv}{b}$ is then obtained via the BIBD admissibility constraint (7). In case b does not divide vr, no PD constraint forces the credit assets to spread in some manner over the co-blocks, so that we do not necessarily have, as in the right-hand incidence matrix of Fig. 2, that $\lfloor \frac{rv}{b} \rfloor \leq k \leq \lceil \frac{rv}{b} \rceil$ for every block size k. Indeed, we have built PDs where the block sizes are distributed over the entire $1, \ldots, v$ range: see the left-hand incidence matrix of Fig. 2.

It is also tempting to think that it is sufficient to find co-blocks whose pairwise overlaps are all exactly λ, rather than at most λ. However, there is no PD $\langle 10, 8, 3, 2\rangle$ where the pairwise overlaps are all equal to 2, whereas Fig. 2 establishes the existence of designs where the pairwise overlaps are at most 2.

2.7 Symmetries and Other Difficulties of PDs

The tranches are indistinguishable, and we assume (in a first approximation) that all the credit assets are indistinguishable. Hence any two rows or columns of the incidence matrix can be freely permuted, which results in $v!b!$ symmetries.

PDs do not exhibit optimal sub-structure, in the sense that an optimal design does not necessarily contain optimal sub-designs, as shown next.

Example 4 (Continuation of Example 3.) Recall the PD $\langle 10, 8, 3, 2\rangle$ on the left-hand side in Fig. 2. Note that the block sizes are distributed over the entire $1, \ldots, v$ range, namely one block each of sizes 1, 5, 6, 10, and four blocks of size 2. Now, for the PD $\langle 8, 8, 3, 1\rangle$, it turns out that there *are* eight subsets of size 3 in an eight-element set such that the maximum overlap is 1. We can now see why PDs do not enjoy the optimal sub-structure property: the discussed design $\langle 10, 8, 3, 2\rangle$ contains no eight subsets of size 3 in the eight-element set such that the maximum overlap is 1. Note that the last four sets each have pairwise overlaps of 1 with four of the first six sets, while all other pairwise overlaps are 2.

The absence of a constraint on the block sizes—compare the vacuous PD condition (14) with the BIBD condition (3)—makes the PD $\langle v, b, r, \lambda\rangle$ much harder to build than the BIBD $\langle v, b, r, k, \lambda\rangle$, if such a k exists.

2.8 Roadmap for the Rest of the Paper

The lower bound on λ of Theorem 1 suggests a (naive) method of exactly building (small) OPDs: set λ to that lower bound and increase it by one each time no corresponding PD is found (within a reasonable amount of time). However, as stated above, this method will only work fast enough for OPDs that are one order of magnitude smaller than typical financial-scale OPDs. In Section 3, we develop a sophisticated exact method for building (small) PDs, without regard to their optimality. Using this method, we then devise in Section 4 an approximate method for building even large PDs that are (near-)optimal, and thereby address the building of financial-scale OPDs.

3 Exact Building of PDs

We now present a sophisticated method, implemented as a constraint program, for exactly building a (small) PD by global search, without regard to its optimality. A first model is introduced in Section 3.1, basically stating the PD constraints (12), (13), (14), (15), and (16) on the incidence matrix. For a briefer read, proceed directly to Section 4, skipping the elaborate optimisations described in the remaining subsections. First, static symmetry-breaking constraints and a static variable and value order are added in Section 3.2 to great effect. Another optimisation concerns BIBDs, which are a special case of PDs and thus easier to build (though still very hard): we show how to exploit very successfully this idea in Section 3.3 and refine this in Section 3.4 by checking against a published list of BIBD-admissible parameters that are known *not* to admit BIBDs. A smaller optimisation, discussed in Section 3.5, fixes the first two rows and the first column of the incidence matrix. Spectacular improvements are sometimes achieved, as reported in Section 3.6, by checking with very low overhead whether the incidence sub-matrix that remains to be labelled, as a PD, satisfies the PD admissibility condition (17). Every PD has a complement PD, which might be easier to build: we demonstrate the usefulness of this observation in Section 3.7, but failed to find an automatable heuristic for deciding when to try this. Finally, in Section 3.8, we show how to exploit the caching of previous results, not just by exact matches, but also by reusing cached designs for harder instances and by failing if there is a cached failure for an easier instance.

3.1 Basic Method

As suggested in Sections 2.5 and 2.7, our most basic PD method represents a PD $\langle v, b, r, \lambda\rangle$ by a $v \times b$ incidence matrix of zeroes and ones, whose rows are constrained

Table 1 Performance comparison of limiting the overlaps in different ways

PD				Results			
v	b	r	λ	Backtracks	Time$_{\text{old}}$	Time$_{\text{scalar\_product}}$	Time$_{\text{scalar\_product\_latest}}$
9	37	12	3	746,750	39.58	24.56	24.42
10	15	6	2	96,822	14.12	4.17	4.33
10	25	8	2	2,492	0.22	0.10	0.09
10	37	14	6	6,932	1.28	0.62	0.63
10	38	10	2	85,238	10.00	5.07	4.95

to have sum r. Three approaches were tried for the constraints that the scalar products of any two rows with distinct indices be at most λ. The first approach, used in [11], statically posts reified conjunction constraints between pairs of elements from the same positions on each row, and constrains the sum of the reified variables to be at most λ. The second and third approaches dynamically post `scalar_product` global constraints [2] during search, either between a newly labelled row and all not yet labelled rows, or between a row about to be labelled and all previously labelled rows.

Table 1 shows the performance (in number of backtracks and CPU seconds until the first design) of the three approaches for a few PDs. Using `scalar_product` constraints roughly halves the run-time, probably because this leads to a lot fewer constraints and hence to less overhead in the finite-domain solver. Delaying constraint posting seems to make no difference, probably because propagation for a constraint is not done until the domain of a variable involved changes.

3.2 Labelling Order and Static Symmetry Breaking

Breaking all the $v!b!$ symmetries can in theory be performed, for instance by posting $v!b! - 1$ global constraints [2] enforcing the (anti-)lexicographical ordering of vectors extracted from the incidence matrix ([5, 10]). In practice, as for BIBDs, breaking just some of those symmetries by just anti-lexicographically ordering the rows (since PD co-blocks can be repeated in the extreme case where $\lambda = r$) as well as anti-lexicographically ordering the columns (since blocks can be repeated) works quite fine [9] for values of b up to about 36, especially when labelling in a row-wise fashion and trying the value 1 before the value 0. However, this is one order of magnitude below the typical value for b in a financial (O)PD.

In PDs with $\lambda = r$ the left-most r columns have only ones. They might not seem very interesting, but they can occur when trying to build OPDs using the method in Section 4.

3.3 BIBD Detection

As noted in Section 2.5, some PDs are actually BIBDs. This is interesting as BIBDs are easier to build than PDs because there are constraints on the columns and because the overlap constraint between co-blocks with distinct indices is an equality instead of an inequality.

Table 2 Performance comparison between the BIBD and PD methods for BIBDs

BIBD/PD					Building BIBD		Building PD	
v	b	r	k	λ	Backtracks	Time	Backtracks	Time
7	7	3	3	1	0	0.00	2	0.00
8	14	7	4	3	57	0.01	209	0.02
9	18	8	4	3	95	0.01	168,384	6.04
9	24	8	3	2	9	0.01	51,487	1.99
10	30	9	3	2	23	0.02	72,795,534	2,984.00

Hence our PD method first tests whether the desired PD could possibly be a BIBD. If the PD parameters satisfy all the BIBD admissibility conditions of Section 2.3, then our PD method first tries to build the given PD as a BIBD, and only if that fails will it try to build it as a PD. If, instead of failing to build a BIBD, the BIBD method times out, then we assume that the PD method will also time out when trying to build a PD.

The used BIBD method is the one outlined in Sections 2.2 and 2.4. The BIBD admissibility checks are more or less free in terms of run-time, but the second part of Proposition 2, where v is odd, was actually omitted in our implementation. The end result when the given PD can be a BIBD is a substantial performance increase (in the number of backtracks and CPU seconds until the first design), as can be seen in Table 2.

Another option would be to try and change the parameters of a PD $\langle v, b, r, \lambda\rangle$ into parameters for a BIBD $\langle v', b', r, k, \lambda\rangle$ with $v' \geq v$ co-blocks, $b' \leq b$ blocks, and k determined by (7), such that they satisfy the BIBD admissibility conditions. Indeed, any additional co-blocks of such a BIBD, if it exists, can just be discarded, and any missing blocks thereof can just be set to the empty sets. For a given PD, we call such a BIBD an *extending BIBD*. As can be seen from Table 3, the performance improvement (in the number of backtracks and CPU seconds the first design) can be huge, from not being able to build a PD in a CPU hour (hence the question marks for the numbers of backtracks in the table) to building it in just a few seconds.

3.4 Checking for Known Non-existing BIBDs

As noted in Section 2.3, the BIBD admissibility conditions are necessary but not sufficient. Therefore, before trying to build a BIBD if the PD parameters satisfy the BIBD admissibility conditions (as just described in Section 3.3), our PD method actually checks those parameters against a list of BIBD-admissible parameters that

Table 3 Performance comparison between building the given PD or an extending BIBD

PD				Building PD		Extending BIBD					Building extending-BIBD	
v	b	r	λ	Backtracks	Time	v	b	r	k	λ	Backtracks	Time
10	33	15	6	?	>3,600	11	33	15	5	6	29,552	2.79
19	20	9	4	?	>3,600	19	19	9	9	4	11,922	1.29

are known *not* to result in BIBDs. We use a sub-list of the parameters published in ([3, 7]), namely those in the ranges we are interested in. If the PD method is given BIBD-admissible parameters that cannot possibly result in a BIBD, then it directly tries to build a PD.

In practice, the cases where BIBD-admissible PD parameters are known not to result in a BIBD are rare. But it is a computationally very cheap check that will avoid an almost certain time-out, or at least a long computation.

3.5 First-intersection and First-column Optimisations

At least one pair of co-blocks with distinct indices in a PD $\langle v, b, r, \lambda\rangle$ can be made to share exactly λ elements.

Proposition 3 *A PD $\langle v, b, r, \lambda\rangle$ where any two co-blocks with distinct indices actually share less than λ elements can be turned into a PD $\langle v, b, r, \lambda\rangle$ where at least one pair of co-blocks with distinct indices shares exactly λ elements.*

Proof Consider a PD $\langle v, b, r, \lambda\rangle$ where any two co-blocks with distinct indices actually overlap over at most $\lambda' < \lambda$ elements. By the full interchangeability of the co-blocks, we can re-order them to have the observed maximum overlap λ' between the co-blocks V_1 and V_2. By replacing an element of V_1, but not of V_2, with an element of V_2, the overlap between these two co-blocks is increased by one. Since no other overlap is increased by more than one when doing this, and since we can repeat this at worst until $\lambda = r$, we have proved the result. □

To build a PD $\langle v, b, r, \lambda\rangle$, at least one pair of co-blocks with distinct indices can, by Proposition 3, share exactly λ elements. Because of the full interchangeability of the co-blocks, we can force them to be the first two co-blocks.

Now consider how our PD method chooses the first two rows initially. For the first row, it tries with r ones to the left and the remaining elements all zero. For the second row, it tries with λ ones to the left, resulting in an overlap of λ between the first two co-blocks, followed by $r - \lambda$ zeroes, followed by the remaining $r - \lambda$ ones, and the remaining elements all zero. See Fig. 3 for an illustration (and ignore the second sentence of its caption for the time being).

If no design exists with this choice of the first two rows, then the solver backtracks and chooses another second row. If all choices for the second row fail, then the solver backtracks and chooses another first row. But any other choice of the first two co-blocks that has an overlap of λ can always be transformed into the choice described above by permuting the columns, because of the full interchangeability of the blocks. Remember that at least one overlap will be λ and that we have this for the first intersection as described above, so this is no limitation. Hence using another

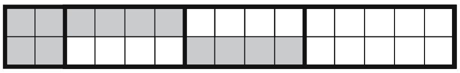

Fig. 3 The *first two rows* of a PD $\langle v, b, r, \lambda\rangle$, with $v \geq 2$, $b = 15$, $r = 6$, and $\lambda = 2$. The complement PD is $\langle v, b, b - r, b - 2r + \lambda\rangle$

Table 4 Performance comparison between plain building and building with the first intersection and first column fixed

BIBD					Plain building		Fix rows 1 and 2 and col. 1	
v	b	r	k	λ	Backtracks	Time	Backtracks	Time
8	14	7	4	3	57	0.01	57	0.00
9	18	8	4	3	96	0.01	95	0.01
9	24	8	3	2	9	0.01	9	0.01
10	30	9	3	2	23	0.01	23	0.02
15	21	7	5	2	111,421	12.63	111,400	12.68
19	19	9	9	4	12,294	1.58	11,922	1.38
25	25	9	9	3	46,105	7.16	46,015	6.98

choice for the elements of the first intersection will only lead to testing rows that are symmetric to rows that have already been tested unsuccessfully. Thus, the choice of the first two rows in the incidence matrix need never be reconsidered.

Only when there is no design, or when we enumerate all designs, will this technique give any performance gain, since only then will the solver backtrack all the way to the choice of the first and second rows. With this technique, fewer rows need to be tried before the solver can be sure no design exists. The number of available symmetries to a fully labelled incidence matrix is decreased from $v!b!$ to $(v-2)!b!$, that is by a factor of $v(v-1)$.

This technique applies to both PDs and BIBDs. But for BIBDs we can do even more. The constant column sum of BIBDs and the anti-lexicographical ordering of their rows imply that the first column must be k ones at the top and the remaining elements all zero. If this was not the case, then the anti-lexicographical ordering would not be satisfied. We cannot fix the second column, though, as there is no constraint between pairs of columns.

By fixing the first column for BIBDs we get a small performance improvement in some cases, not only when there is no design or when we enumerate all designs. Table 4 shows the performance impact (in the number of backtracks and CPU seconds) for finding the first, if any, BIBD. We see that using this optimisation leads to some small performance gains. For example, for the BIBD $\langle 15, 21, 7, 5, 2\rangle$, the number of backtracks decreases slightly. A decrease was to be expected as it is known that there exists no BIBD with these parameters even though they satisfy the BIBD admissibility conditions. Of course, we here switched off the checking for known non-existing BIBDs that was presented in Section 3.4, in order to allow the search to exhaust all the possibilities for the BIBD $\langle 15, 21, 7, 5, 2\rangle$ and see what impact the optimisation has on the performance.

Fixing the first column and the first row (but not the second row) has been done in [9], but there it was only evaluated as a technique for breaking symmetry in BIBDs but was not tested *together* with other symmetry-breaking techniques.

3.6 Checking Admissibility During Search

The anti-lexicographical ordering constraints on rows and columns are not only an efficient technique to break relatively much symmetry, but they also allow us to use

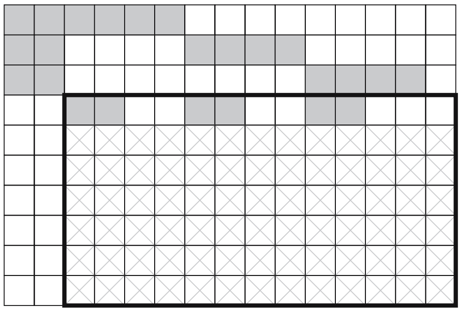
Fig. 4 Incidence matrix for a partially labelled PD$\langle 10, 15, 6, 2\rangle$. A *grey cell* represents a one, a *white cell* a zero, and a *crossed-out cell* an element that has not been labelled yet. The *bold rectangle* represents the PD $\langle 7, 13, 6, 2\rangle$, which does not satisfy the admissibility condition of Theorem 1, hence backtracking will occur

this knowledge about how a design must be formed in order to devise a more efficient method. The previous Section 3.5 showed one application thereof, and we now look at another, beginning with an example.

Example 5 Consider the partially labelled matrix for the PD $\langle 10, 15, 6, 2\rangle$ in Fig. 4. After labelling the first four rows, the solver has reached a state where the first two columns can no longer be used, since the rows are ordered anti-lexicographically. Hence any remaining row must consist of elements only in the singled out lower-right bold rectangle of the incidence matrix. But this part of the matrix *must be a PD on its own* and must therefore satisfy the admissibility condition of Theorem 1, with the same λ and r, but b and v decreased accordingly, giving the PD $\langle 7, 13, 6, 2\rangle$. In this case, that condition is not satisfied as we get $\lambda \geq \lceil \frac{16}{7} \rceil = 3$, but we are trying to build this design with $\lambda = 2$.

The technique above works for any PD or BIBD. After labelling a row, the solver looks in what column it has its first element, as any column to the left thereof is then unusable. For the resulting sub-design with the newly labelled row as the first row, and possibly fewer columns, the admissibility condition of Theorem 1 is checked. If is is not satisfied, then we can infer that this last labelled row will never lead to a design and force the solver to backtrack.

There are two special cases to consider. First, when the last labelled row has no first element, that is when it is empty, then we cannot do anything. This can only happen when $r = 0$, and hence the entire matrix will be empty. PDs with $r = 0$ occur sometimes when using the approximate method for OPDs in Section 4. Second, when

Table 5 Performance comparison between switching off and on the admissibility check during search for the remaining rows and columns

PD				Without check		With check during search	
v	b	r	λ	Backtracks	Time	Backtracks	Time
10	15	6	2	96,822	4.33	138	0.01
11	11	5	2	764	0.05	45	0.01
15	15	4	1	3,167,791	279.97	2,335,130	228.99
16	8	3	1	729	0.08	11	0.00

there is only one row left, then we need not check anything either, since the concept of overlap with all the remaining co-blocks becomes meaningless.

Table 5 shows the sometimes substantial performance gains (in number of backtracks and CPU seconds until the first design) we can get with this technique. Note that only PDs have been picked for which the technique is useful; many other PDs are not constrained enough for the admissibility check during search to fail, and hence for them this check is only a (small) overhead.

3.7 Using the Complement

When the solver has found an incidence matrix representing a PD or BIBD, then we have actually found another PD as well, namely the complement PD. To get the complement to a PD or BIBD represented as an incidence matrix, switch every 1 to a 0 and vice-versa.

For the complement of a PD $\langle v, b, r, \lambda\rangle$, the dimensions of the $v \times b$ incidence matrix are not changed, and the row sum r is replaced by $b - r$. In addition, the column sum k of a BIBD $\langle v, b, r, k, \lambda\rangle$ is replaced by $v - k$ in the complement BIBD. To realise what the maximum (or exact) overlap λ is in the complement PD (or complement BIBD), look at Fig. 3 again (and ignore the first sentence of its caption). The value of λ for the complement is the width of the last rectangle, which contains only non-filled elements. The width of this last rectangle is the total width b minus the width of each of the other rectangles, hence it is

$$b - \lambda - (r - \lambda) - (r - \lambda) = b - 2r + \lambda$$

To summarise, a PD $\langle v, b, r, \lambda\rangle$ (or BIBD $\langle v, b, r, k, \lambda\rangle$) also yields the complement PD $\langle v, b, b - r, b - 2r + \lambda\rangle$ (or the complement BIBD $\langle v, b, b - r, v - k, b - 2r + \lambda\rangle$).

Table 6 shows that the performance impact can be substantial for some PDs. They are listed in such a way that the complement has a smaller row sum than the original PD.

Unfortunately, no heuristic for deciding when to build the complement instead of the given design has been found. Therefore the complement is not tried by our method, but it can of course be manually fed to it instead of the given design.

However, in the next section, we show that the complement can be automatically used to increase the performance in other occasions as well.

Table 6 Performance comparison between building the given PD and its complement

PD				Building PD		Complement PD				Building complement PD	
v	b	r	λ	Backtracks	Time	v	b	r	λ	Backtracks	Time
10	38	28	20	1,360,590	91.24	10	38	10	2	85,238	5.01
10	31	22	15	992,711	52.09	10	31	9	15	4,044,363	152.22
10	15	6	2	3,251	0.16	10	15	9	5	138	0.02

3.8 Caching Results and Extended Cache Lookup

For each PD, our method will either succeed (find a design), or fail (establish that there is no design), or time out prior to succeeding or failing. Each of these results is cached, including the incidence matrix for any design, so that if the same design resurfaces, the method need not tackle it again, but can just look up the result.

This technique can even be extended to further increase the performance of our method, namely by looking for harder designs that have already been built with success and that can be adapted to become the wanted design. We assume that fewer columns always make a design harder to build, so that if a design is found, then so will the same design with more columns, but in at most as much time. We also assume that more rows always make a design harder to build, so that if a design is found, then so will the same design with fewer rows, but in at most as much time. There are thus two possibilities.

First, if we are to build the PD $\langle v, b, r, \lambda\rangle$ and it is not in the cache, but $\langle v, b', r, \lambda\rangle$ with $b' < b$ is there as a successfully built design, then the cached design $\langle v, b', r, \lambda\rangle$ provides a design $\langle v, b, r, \lambda\rangle$: take the first b' columns from $\langle v, b', r, \lambda\rangle$ and set the remaining $b - b'$ columns to all zeroes.

Second, if we are to build the PD $\langle v, b, r, \lambda\rangle$ and it is not in the cache, but $\langle v', b, r, \lambda\rangle$ with $v' > v$ is there as a successfully built design, then the cached design $\langle v', b, r, \lambda\rangle$ provides a design $\langle v, b, r, \lambda\rangle$: take the first v rows from $\langle v', b, r, \lambda\rangle$ and ignore the remaining $v' - v$ rows.

We can also do the converse, namely by looking for easier designs that have already been tried without success (with failure or time-out). We assume that more columns always make a design easier to build, so that if a design is not found prior to time-out, then so will the same design with fewer columns, but in at least as much time. We also assume that fewer rows always make a design easier to build, so that if a design is not found prior to time-out, then so will the same design with more rows, but in at least as much time. There are thus again two possibilities.

First, if we are to build the PD $\langle v, b, r, \lambda\rangle$ and it is not in the cache, but $\langle v, b', r, \lambda\rangle$ with $b' > b$ is there as a failed or timed-out design, then the solver will fail or time out on $\langle v, b, r, \lambda\rangle$ as well, under at least the same time-out limit.

Second, if we are to build the PD $\langle v, b, r, \lambda\rangle$ and it is not in the cache, but $\langle v', b, r, \lambda\rangle$ with $v' < v$ is there as a failed or timed-out design, then the solver will fail or time out on $\langle v, b, r, \lambda\rangle$ as well, under at least the same time-out limit.

We call this technique *extended cache lookup*. When approximately building large OPDs by embedding small PDs (see Section 4), it is common for the same PDs to appear several times, and also for PDs of roughly the same size to appear many times, so this technique will have a large impact on the performance of that method.

This extended cache lookup is implemented separately from the checking for known non-existing BIBDs described in Section 3.4, even though nothing would prevent us from merging the two. In our method, the parameters are first checked against the cache. If they are not found in the cache, then they are checked against the list of parameters that are known not to result in a BIBD. If they are not there either, then our method tries to build a PD, which will result in a new entry in the cache.

Finally, we can make use of the complement concept in Section 3.7 again. If the extended cache lookup cannot find a design, then an extended cache lookup is done for the complement instead, and only if this also fails does the method need to try and build a PD.

4 Approximate Building of OPDs

The method of Section 3 for exactly building PDs does not scale for the building of typical financial OPDs. In Section 4.3, we introduce a method of approximately building such large OPDs, using the exact PD method of Section 3. It rests on two key insights, explained in Sections 4.1 and 4.2.

4.1 Underconstrainedness

The first insight comes from observing that the typical values of v (the number of co-blocks) are quite small for the typical values of b (the number of blocks) and r (the size of the co-blocks), as shown in the following example.

Example 6 The first three columns of Table 7 chart how the lower bound on λ evolves with $v \geq 2$ according to the PD admissibility condition (17) when $b = 350$ and $r = 100$. The lower bound on λ initially grows from 0 for $v = 2$, to between 9 and 26 for the typical values of v (which are between 4 and 25), but does not grow much after that; in fact, it never exceeds 29, which it reaches for $v = 127$. This effect is exacerbated for smaller values of b and r, as shown in the fourth and fifth columns of Table 7.

While this example illustrates a prediction weakness of the lower bound (17) of Theorem 1 for large values of v, the main lesson is that there is a range for v in which the lower bound on λ does not change quickly for fixed values of b and r. For the ranges of values of v, b, and r that are of interest here, v is within that zone.

The consequence is that the PDs of interest here seem underconstrained in the sense that one may get (many) more than the intended v co-blocks of the same size r from the same universe of b credit assets, without seeing the maximum overlap λ of the co-blocks increase. Dually, one may draw the intended v co-blocks of the same size r from a (much) smaller universe than the available b credit assets, without seeing the maximum overlap λ of the co-blocks increase. For instance, Theorem 1 predicts that $v = 10$ co-blocks of $r = 100$ credit assets each may be drawn with a maximum overlap $\lambda = 22$ from a universe of $337 \leq b \leq 351$ credit assets. Again, this effect is exacerbated for smaller values of b and r. This underconstrainedness may lead to considerable combinatorial explosion. In fact, we have been unable to build any PDs of the magnitude considered here with the BIBD-style method outlined in Section 3, even when setting a quite high value for λ and allocating an entire CPU week. Labelling just one row of the incidence matrix already tends to take a lot of time after the first few rows.

Table 7 Unrounded and rounded lower bounds on the maximum overlap λ for $v \geq 2$ co-blocks and b blocks of size r, as given by the PD admissibility condition (17)

	$b = 350$ and $r = 100$		$b = 35$ and $r = 10$			
v	Unrounded lower bound on λ	Rounded lower bound on λ	Unrounded lower bound on λ	Rounded lower bound on λ	Time to first design	Backtracks to first design
2	0.000	0	0.0000	0	0.01	0
3	0.000	0	0.0000	0	0.00	0
4	8.333	9	0.8333	1	0.01	1
5	15.000	15	1.5000	2	0.02	184
6	16.667	17	1.6667	2	0.05	658
7	16.667	17	1.6667	2	0.08	921
8	19.643	20	1.9643	2	0.34	8,872
9	20.833	21	2.0833	3	0.04	566
10	21.111	22	2.1111	3	0.07	567
11	21.818	22	2.1818	3	0.08	567
12	22.727	23	2.2727	3	0.09	663
13	23.077	24	2.3077	3	0.14	1,878
14	23.077	24	2.3077	3	0.14	2,038
15	23.810	24	2.3810	3	0.19	2,245
16	24.167	25	2.4167	3	0.45	9,331
17	24.265	25	2.4265	3	0.52	10,221
…		25		3		
22	25.325	26	2.5325	3	1.26	16,078
…		26		3		
29	26.108	27	2.6108	3	3.33	35,305
…		27		3		
46	27.005	28	2.7005	3	?	?
…		28		3		
127	28.009	29	2.8009	3	?	?
…		29		3		

4.2 Embeddings

The second insight is that building optimal designs is not always practical. As shown below, we can often very efficiently build real-life financial PDs with values for λ that are within 5% of, if not identical to, the lower bound given by the PD admissibility condition (17). So we investigate the approximate building of real-life financial OPDs. The idea is to embed small PDs within a large one, as illustrated in the following example.

Example 7 A not necessarily optimal PD for the OPD $\langle 10, 350, 100 \rangle$ can be built by making ten copies of each column in any possibly optimal PD for the OPD $\langle 10, 35, 10 \rangle$. The fifth column of Table 7 gives $\lambda \geq 3$ for the OPD $\langle 10, 35, 10 \rangle$. Building the PD $\langle 10, 35, 10, 3 \rangle$ with the BIBD-style method in Section 3 takes about 0.07 CPU seconds and 567 backtracks to succeed. Since $10 \cdot 3 = 30$, this means that we can build from it a PD $\langle 10, 350, 100, 30 \rangle$. Since the third column of Table 7 gives $\lambda \geq 22$ for the

OPD $\langle 10, 350, 100\rangle$, the built PD with $\lambda = 30$ is quite far above that lower bound and may thus be sub-optimal. (This example will be continued in Example 9.)

This kind of embedding is a standard concept for BIBDs [3].

Definition 4 A BIBD $\langle v, b, r, k, \lambda\rangle$ is an *m-multiple BIBD* if $\langle v, \frac{b}{m}, \frac{r}{m}, k, \frac{\lambda}{m}\rangle$ parameterises a BIBD under the constraints (1), (2), (3), (4), and (5).

In other words, shrinking the number of blocks by a factor m shrinks the sizes of the co-blocks and their overlaps by the same factor m (provided they all divide m). The corresponding concept for PDs has a similar definition.

Definition 5 A PD $\langle v, b, r, \lambda\rangle$ is an *m-multiple PD* if $\langle v, \frac{b}{m}, \frac{r}{m}, \frac{\lambda}{m}\rangle$ parameterises a PD under the constraints (12), (13), (14), (15), and (16). We denote this by $\langle v, b, r, \lambda\rangle = m \cdot \langle v, \frac{b}{m}, \frac{r}{m}, \frac{\lambda}{m}\rangle$.

For OPDs, we can only compare the predicted lower bounds on their maximum overlaps, rather than the actual maximum overlaps as for PDs. The following property establishes that the same ratio holds between those lower bounds.

Observation 1 *The PD admissibility condition* (17) *predicts* $\lambda \geq \mu$ *for the OPD* $\langle v, b, r\rangle$ *if and only if it predicts* $\lambda \geq \frac{\mu}{m}$ *for the OPD* $\langle v, \frac{b}{m}, \frac{r}{m}\rangle$.

Example 8 Table 7 confirms the ratio of 10 between the unrounded lower bounds on λ for the OPDs $\langle v, 350, 100\rangle$ and $\langle v, 35, 10\rangle$, with $v \geq 2$.

However, a PD is not always an exact multiple of another PD. We advocate generalising the notion of multiples of a design and here do so for PDs. Let us first show the intuition on an example.

Example 9 (Continuation of Example 7.) Reconsider the $\langle 10, 350, 100, \lambda\rangle$ PDs. They are not 12-multiples of any PD as 12 does not divide both 350 and 100. Since $350 = 12 \cdot 27 + 26$ and $100 = 12 \cdot 8 + 4$, a not necessarily optimal PD $\langle 10, 350, 100, \lambda\rangle$ can be built by making 12 copies of each column in any possibly optimal PD $\langle 10, 27, 8, \lambda_1\rangle$ and appending any possibly optimal PD $\langle 10, 26, 4, \lambda_2\rangle$. The admissibility condition (17) gives $\lambda_1 \geq 2$ and $\lambda_2 \geq 1$. Building the PDs $\langle 10, 27, 8, 2\rangle$ and $\langle 10, 26, 4, 1\rangle$ with the BIBD-style method in Section 3 takes about 0.03 CPU seconds and 69 backtracks total to succeed. Since $12 \cdot 2 + 1 = 25$, this means that we can build from them a PD $\langle 10, 350, 100, 25\rangle$. Since the third column of Table 7 gives $\lambda \geq 22$ for the OPD $\langle 10, 350, 100\rangle$, the built PD with $\lambda = 25$ is still a bit above that lower bound and may be sub-optimal. (This example will be continued in Example 10.)

Let us now formalise all the intuitions from this example.

Definition 6 A PD $\langle v, b, r, \lambda\rangle$ *embeds* m occurrences of a PD $\langle v, b_1, r_1, \lambda_1\rangle$ and one occurrence of a PD $\langle v, b_2, r_2, \lambda_2\rangle$, which is denoted by $\langle v, b, r, \lambda\rangle = m \cdot \langle v, b_1, r_1, \lambda_1\rangle + \langle v, b_2, r_2, \lambda_2\rangle$, if it is built from m copies of each column of

$\langle v, b_1, r_1, \lambda_1 \rangle$ and one copy of each column of $\langle v, b_2, r_2, \lambda_2 \rangle$, modulo row and column swaps.

This definition implies that the following four conditions hold:

$$0 \leq r_i \leq b_i \geq 1 \quad \text{for } i = 1, 2 \tag{18}$$

$$b = mb_1 + b_2 \tag{19}$$

$$r = mr_1 + r_2 \tag{20}$$

$$\lambda \leq m\lambda_1 + \lambda_2 \tag{21}$$

The condition (18) ensures that the co-blocks can be subsets of B, for each of the two embedded PDs. It also eliminates the two cases ($b_i = 0$) where the PD admissibility condition (17) cannot be evaluated. The conditions (19) and (20) ensure that the embedding is exact. The reason why there is an inequality in condition (21) is that λ is the *maximum* overlap. Consider $v = 3$ and $m = 1$: the first embedded PD may have $1, 1, 2$ as overlaps, and the second embedded PD may have $1, 2, 1$ as overlaps, both with a maximum of 2, giving $1+1, 1+2, 2+1$ as overlaps for the embedding PD, with a maximum of 3, which is less than the upper bound $1 \cdot 2 + 2 = 4$ given by condition (21). For this reason, the calculated maximum overlap $\lambda = 25$ of the embedding PD in Example 9 is in fact a predicted upper bound, rather than necessarily the exact value as stated there. Hence it is in general better to use the actually *observed* value of λ of the embedding PD than the *predicted* upper bound given by condition (21). In that example, observation establishes that $\lambda = 25$ indeed.

Note that this embedding by vertical division of the incidence matrix is possible because of the currently assumed full column symmetry of the latter and because no PD constraint works against it. However, an embedding by horizontal division of the incidence matrix will lead to identical rows, that is worst-case designs ($\lambda = r$).

4.3 Approximate Building

Given a (financial-scale) OPD $\langle v, b, r \rangle$, the issue now becomes how to construct suitable PD embeddings, so that a PD $\langle v, b, r, \lambda \rangle$ with λ (near-)optimal can be built efficiently. An additional input is an admissible value Λ of λ that we are trying to match or undercut, say because it is one unit lower than the observed value of λ for the currently best PD, or one unit lower than the predicted upper bound on that value, as determined by condition (21).

The objective is to find values for m, b_1, r_1, b_2, and r_2 such that $\langle v, b, r, \lambda \rangle = m \cdot \langle v, b_1, r_1, \lambda_1 \rangle + \langle v, b_2, r_2, \lambda_2 \rangle$, where $\lambda \leq \Lambda$ and λ_i is the rounded lower bound given for v, b_i, r_i by the PD admissibility condition (17). Two heuristic constraints in addition to the four conditions (18) to (21) become necessary in order to make the method pragmatic.

First, we must restrict the focus to the pairs of embedded PDs that have a chance of leading to a PD whose maximum overlap does not exceed Λ:

$$m\lambda_1 + \lambda_2 \leq \Lambda \tag{22}$$

Indeed, the left-hand side is by condition (21) the predicted upper bound on the maximum overlap of the embedding PD built from the two embedded PDs $\langle v, b_i, r_i, \lambda_i \rangle$, if they exist. In practice, it is usually equal to the observed maximum overlap of such an embedding PD, hence this constraint. Note that this constraint implies that $m \leq \Lambda$.

Second, knowing that PDs with values of b up to a threshold $T \approx 36$ can often be built (quite quickly) using the BIBD-style method in Section 3, the objective in choosing the parameters of the embedding is to have *both* embedded PDs within that range for b:

$$b_i \leq T \quad \text{for } i = 1, 2 \tag{23}$$

where T is the last input to our OPD method. Good values of T are between 20 and 40 when $10 \leq v \leq 20$.

Note that the determination of candidate embeddings is thus itself a constraint satisfaction problem.

There is no guarantee that all PDs with $b \leq T$ can be built sufficiently quickly. For instance, the sixth and seventh columns of Table 7 chart the CPU times in seconds and backtracks for $\langle v, 35, 10, \lambda \rangle$ for $v \geq 2$ and λ equal to the rounded lower bound in the fifth column. The experiments were conducted using the BIBD-style method in Section 3. A question mark means that we stopped the method after one CPU hour. We observe that for any range of values of v where the rounded lower bound on λ remains the same, the runtimes increase with v. In other words, they increase when the rounding distance for the lower bound on λ decreases. This may not always be the case. The same pattern can be observed for the number of backtracks. The rounding distance seems to be a good indicator of the constrainedness of a PD. A good heuristic then seems to be that we should favour embeddings where both embedded PDs have not too small rounding distances. In our observation, for the typical values of v, PDs with rounding distances below 0.15 are often problematic. Hence we also advocate ordering the embedded PD pairs that satisfy the constraints (18), (19), (20), (21), (22), and (23) by decreasing rounding distance to the next integer for λ_1, so that the apparently easier PD pairs are attempted first. Setting a time limit on each attempt is another useful refinement. Let us now illustrate this method, summarised in Algorithm 1, which is non-deterministic.

Algorithm 1 Approximate building of OPDs

```
Require: $v, b, r, \Lambda, T$, time-out
  $m, b_1, r_1, \lambda_1, b_2, r_2, \lambda_2 := solve((18)–(23))$
  if $v, b_1, r_1, \lambda_1 := solve((12)–(16))$ and $v, b_2, r_2, \lambda_2 := solve((12)–(16))$ then
    return $m \cdot \langle v, b_1, r_1, \lambda_1 \rangle + \langle v, b_2, r_2, \lambda_2 \rangle$
  end if
```

Example 10 (Continuation of Example 9.) Let us try and improve on the possibly sub-optimal PD with $\lambda = 25 = \Lambda + 1$ previously obtained for the OPD $\langle 10, 350, 100 \rangle$. The embeddings satisfying the constraints (18), (19), (20), (21), (22), and (23) with $T = 36$ are given in Table 8, ordered by decreasing rounding distance to the next integer for λ_1. Setting a time limit of 5 CPU minutes, we now attempt to build the PDs in the second and fourth columns, proceeding row by row.

Table 8 Embeddings of $\langle 10, 350, 100\rangle$ satisfying the constraints (18), (19), (20), (21), (22), and (23) for $\Lambda = 24$ and $T = 36$, ordered by decreasing rounding distance to the next integer for λ_1

m	$\langle v, b_1, r_1, \lambda_1\rangle$	Unrounded λ_1	$\langle v, b_2, r_2, \lambda_2\rangle$	Unrounded λ_2	$m\lambda_1 + \lambda_2$
10	$\langle 10, 32, 09, 2\rangle$	1.867	$\langle 10, 30, 10, 3\rangle$	2.667	23
11	$\langle 10, 31, 09, 2\rangle$	1.933	$\langle 10, 09, 01, 1\rangle$	0.022	23
11	$\langle 10, 30, 09, 2\rangle$	2.000	$\langle 10, 20, 01, 0\rangle$	0.000	22

For the first embedding, it only takes about 0.76 CPU seconds and 13, 152 backtracks total to build its two PDs. Hence we can build a PD $\langle 10, 350, 100, \lambda\rangle$ from ten copies of the PD $\langle 10, 32, 9, 2\rangle$ and one copy of the PD $\langle 10, 30, 10, 3\rangle$; it has a predicted and observed maximum overlap $\lambda = 10 \cdot 2 + 3 = 23$, which is better than the PD in Example 9.

For the second embedding, it takes about 157 CPU seconds and about $4 \cdot 10^6$ backtracks (mostly because of the first embedded PD, as the second one has $\lambda_2 = r_2$ and is thus trivial to build). We get another design of predicted and observed maximum overlap $\lambda = 11 \cdot 2 + 1 = 23$.

The third embedding is very interesting. Building its first embedded PD can be tried as a BIBD with blocks of fixed size $k = 3 = \frac{r_1 v}{b}$ and pairwise overlaps of *exactly* λ_1 elements (rather than *at most* λ_1 elements), as the unrounded λ_1 is a natural number, namely 2, and as b_1 divides $r_1 v$. The additional constraint (3) on the block sizes and the requirement of exact rather than bounded overlaps give very good propagation. With this detection of potential BIBDs switched on (see Section 2.4), our PD method builds this PD in about 0.02 CPU seconds and 23 backtracks; with this detection switched off, it takes about 3, 000 CPU seconds and $73 \cdot 10^6$ backtracks on the corresponding PD, which does not have that constraint (see Table 2). The second embedded PD is trivial (in the sense that there are at least as many credit derivatives as in the union of the requested co-blocks) since $r_2 v \leq b_2$ and is built in about 0.01 CPU seconds and 0 backtracks. Hence we can build a PD $\langle 10, 350, 100, \lambda\rangle$, given in Fig. 5, from 11 copies of the PD $\langle 10, 30, 9, 2\rangle$ and one copy of the PD $\langle 10, 20, 1, 0\rangle$; it has a predicted and observed maximum overlap $\lambda = 11 \cdot 2 + 0 = 22$ and is optimal, as (17) gives $\lambda \geq 22$. Note that the last 10 blocks are empty in this PD.

Some of our experiments are summarised in Table 9, for typical financial OPDs. We always initially set Λ to the lower bound given by Theorem 1. For the three first OPDs we used $T = 36$, while for the remaining two OPDs we had to use a higher threshold, for instance $T = 72$ here, to get any embeddings at all. We used

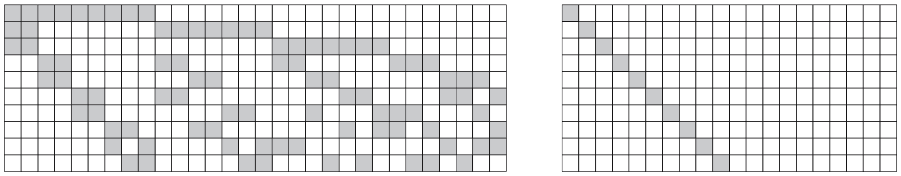

Fig. 5 Optimal portfolio design $\langle 10, 350, 100\rangle$, built from $11 \cdot \langle 10, 30, 9, 2\rangle + \langle 10, 20, 1, 0\rangle$, and of maximum overlap $11 \cdot 2 + 0 = 22$

Table 9 Some attempts at building optimal portfolio designs by embeddings

OPD $\langle v, b, r\rangle$	Bound (17)	Λ	T	$m \cdot \langle v, b_1, r_1, \lambda_1\rangle + \langle v, b_2, r_2, \lambda_2\rangle$	Observed overlap	Run time
$\langle 10, 350, 100\rangle$	22	22	36	$11 \cdot \langle 10, 30, 09, 2\rangle + \langle 10, 20, 01, 0\rangle$	22	0.02
$\langle 09, 300, 100\rangle$	25	25	36	$24 \cdot \langle 09, 12, 04, 1\rangle + \langle 09, 12, 04, 1\rangle$	25	0.00
$\langle 10, 325, 100\rangle$	24	24	36	$10 \cdot \langle 10, 30, 09, 2\rangle + \langle 10, 25, 10, 4\rangle$	24	0.17
$\langle 10, 360, 120\rangle$	32	32	72	$12 \cdot \langle 10, 25, 08, 2\rangle + \langle 10, 60, 24, 8\rangle$	32	68.91
$\langle 15, 350, 100\rangle$	24	24	72	$05 \cdot \langle 15, 60, 17, 4\rangle + \langle 15, 50, 15, 4\rangle$	–	$> 3,600.00$
		24	72	$09 \cdot \langle 15, 33, 09, 2\rangle + \langle 15, 53, 19, 6\rangle$	–	$> 3,600.00$
		25	72	$\cdots$	–	$> 3,600.00$
		26	72	$06 \cdot \langle 15, 50, 15, 4\rangle + \langle 15, 50, 10, 2\rangle$	26	864.00

a time-out of one CPU hour and report run times in CPU seconds until the first built design. As can be seen, an optimal design for the last instance eluded even the embedding-based approach under these parameters, even though it is known to exist [1]. Even setting $\Lambda = 25$ did not help under these settings. Fortunately, for $\Lambda = 26$, a candidate embedding led to success before time-out.

5 Conclusion

Summary We have given an approximate and often extremely fast method, implemented as a constraint program, of building (near-)optimal portfolio designs (OPDs), with a financial application in designing CDO^2 transactions in the credit derivatives market. Their corresponding satisfaction designs, namely portfolio designs (PDs), generalise balanced incomplete block designs (BIBDs). However, typical financial PDs are an order of magnitude larger than the largest BIBDs built so far by constraint programs, and PDs lack a counterpart of a crucial BIBD constraint. Hence current BIBD-style methods are not suitable for real-life financial OPDs. Our method is based on embedding (multiple copies of) independent designs into the original design. Their determination is itself a constraint satisfaction problem. The high quality of our approximate designs can be assessed by comparison with a lower bound on the maximum overlap.

Generalisation The generalisation of the main idea is as follows, in the context of a large instance of a constraint optimisation problem (COP) where a bound on the cost can be somehow calculated. One can then solve the corresponding constraint satisfaction problem (CSP) instances for costs satisfying that bound in order at least to get good feasible solutions to the original COP instance. The idea is to embed several independent small CSP instances P_i within a large CSP instance P corresponding to the given COP instance. This approximation amounts to restricting the search space to feasible solutions of a given structure. For an OPD $\langle v, b, r\rangle$, that structure is dictated by $\langle v, b, r, \lambda\rangle = m \cdot \langle v, b_1, r_1, \lambda_1\rangle + \langle v, b_2, r_2, \lambda_2\rangle$. A solution S to P can then be built from solutions S_i to the P_i. If there is a relationship between the costs of S and the S_i, then this relationship can be used to determine CSP instance candidates for the P_i, via another CSP, using as cost estimates the calculated bounds on the costs of the corresponding COP instances. For OPDs, this relationship is given by $\lambda \leq m\lambda_1 + \lambda_2$.

Related Work The idea of exploiting *independent* sub-problems also underlies Tree-based Russian Doll Search [14]. The idea of embedding (multiple copies of) sub-problem instances into a larger problem instance is related to the concept of abstract local search [4], where a concrete solution is built from a solution to an abstraction of the original problem instance and then analysed for flaws so as to infer a new abstract solution. This works well if the concretisation and analysis steps are tractable and if the abstraction is optimality preserving, in the sense that optimal concrete solutions can be built from abstract solutions. Our embedded problem instances can indeed be jointly seen as an *abstraction* of the original problem instance. For instance, entire bundles of credit assets are here abstracted into single super-credit-assets. We have been unable so far to prove optimality preservation of such portfolio abstractions, or to find conditions for it. As also observed in [4], this is not problematic for hard problem instances, such as the typical financial OPDs considered here, where the utility of abstractions can only be assessed by comparison with other techniques. In any case, we have seen that our portfolio abstractions lead to solutions that are extremely close to a lower bound on the maximal overlap.

Also, we have found only one paper taking a constraint programming approach to portfolio design [17], but the tackled problem there is actually different from ours and is limited to portfolios consisting of just one tranche.

Future Work The quality of our new, tighter lower bound in Theorem 1 on the maximal overlap λ of an OPD $\langle v, b, r\rangle$ is an open question. For instance, note that the bottom of Table 7 suggests that an arbitrary error might be achievable for disproportionately large values of v compared to b and r.

Our notion of embedding can be generalised to any linear combination of several designs. Indeed, Definition 6 is restricted to embeddings of always *two* designs, namely one quotient design and one remainder design, with coefficients m and 1, respectively. The price to pay for this structural restriction of the search space may be that a design of optimal maximum overlap eludes us sometimes.

Some additional abstraction may reduce the $0, \ldots, v$ range of observed block sizes. Indeed, a counterpart of the BIBD constraint (3) might enormously speed up the building process. The facts that some PDs (such as $\langle 9, 35, 10, 2\rangle$) take CPU days to fail while increasing their λ (to obtain $\langle 9, 35, 10, 3\rangle$ here) then leads to quasi instantaneous success, and that other PDs (such as $\langle 10, 30, 9, 2\rangle$) take many CPU minutes to build while the corresponding BIBDs, if any ($\langle 10, 30, 9, 3, 2\rangle$ here), are built quasi instantaneously, show that there is still much space for improving our method. For instance, it would be nice to prove that enforcing $\lfloor \frac{rv}{b} \rfloor \leq k \leq \lceil \frac{rv}{b} \rceil$ for every block size k does not prevent the existence of PDs. As can be seen in the two incidence matrices of Fig. 2, such a constraint leads to a better spread of the available credit assets over the tranches (which might be desirable from a financial point of view) and to a better usage of the admissibility check during search proposed in Section 3.6. This should also help overcome some of the hardness we observed for OPDs with $v \geq 15$ tranches.

Since PDs have overlaps that are *at most* λ, we can sometimes reorder the rows of embedded PDs before appending these embedded PDs into the embedding PD, such that its observed maximum overlap is decreased. Consider again $v = 3$ and $m = 1$: assume both embedded PDs have $1, 1, 2$ as overlaps, with a maximum of 2, giving $1 + 1, 1 + 1, 2 + 2$ as overlaps for the embedding PD, with a maximum of 4, which is the upper bound $1 \cdot 2 + 2 = 4$ given by condition (21). However, upon reordering the rows of the second embedded PD such that its overlaps are $1, 2, 1$, we get $1 + 1, 1 + 2, 2 + 1$ as overlaps for the embedding PD, with a maximum of 3, which is less than the upper bound $1 \cdot 2 + 2 = 4$ given by condition (21).

Our experiments so far with dynamic symmetry-breaking by dominance detection, using the STAB technique [15], are reported in [16]. The results have been inconclusive, but more work is needed.

Instead of posting for any two co-blocks with distinct indices a constraint on their intersection size, it would be more efficient to design a global constraint [2] that maintains the minimum intersection size for a set of co-blocks. A global constraint would not only reduce the computational overhead, but would possibly be able to do more propagation.

Finally, it would be interesting to try integer programming (IP) on the (optimal) portfolio design problem and to test how far it scales, both with and without the proposed embedding approach. Under standard IP modelling techniques, a total of bv^2 extra variables and $3bv^2$ channelling constraints have to be introduced to model in a linear fashion the essentially quadratic logical conjunction inherent in the co-block intersection constraint. In our preliminary experiments, such an IP model builds PDs slower (under CPLEX, via OPL 3.7) than our most basic constraint program (under ILOG Solver, via OPL 3.7) of Section 3.1 with the labelling order in Section 3.2, but without any of the improvements of Sections 3.2 to 3.8. Improving that initial IP model is best left to IP experts.

Conclusion and Financial Relevance Our OPD method has eliminated the need for ad hoc manual permutations when designing CDO^2 transactions. On average, we have found that the maximum overlap in a given financial PD can be decreased anywhere from 2 to 4% by using the new method. Even though this may not sound like a dramatic improvement, the ability to reduce the maximum overlap from 25 to 22%, say, may make the difference between having or not having a feasible transaction due to investor and rating-agency constraints. Issuers of certain types of financial instruments will benefit directly when their instruments can be made more attractive.

It should be pointed out that it is easy to reduce the overlap by increasing the number of available credits. However, such new credits tend to be less known and thus more difficult to analyse, resulting in less than efficient portfolios.

In practice, the credit assets are not all indistinguishable. A client might have personal preferences for or against some credit assets, declare some credit assets as mutually exclusive, and so on. The advantage of our deployment of constraint technology is that such specific needs can be neatly handled without having to devise new (optimal) portfolio design methods from scratch each time. However, such side constraints may break some of the full column symmetry, so piecewise symmetry breaking has to be deployed instead. Furthermore, the likely addition of many more side-constraints will make the problem less and less purely combinatorial, and this is the typical scenario where constraint programming is expected to be faster or to find better solutions than rival technologies, such as integer programming.

Acknowledgements We thank Ian Gent, Nic Wilson, and the referees for their useful comments.

References

1. Ågren M., Flener, P., & Pearson, J. (2005). Incremental algorithms for local search from existential second-order logic. In van Beek, P., ed., *Proceedings of CP'05*, vol. 3709, LNCS, pages 47–61. Springer, Berlin Heidelberg New York.
2. Beldiceanu N., Carlsson, M., & Rampon, J.-X. (November 2005). Global constraint catalog. Technical Report T2005-08, Swedish Institute of Computer Science.
3. Colbourn C. J., & Dinitz J. H., eds., (1996). *The CRC Handbook of Combinatorial Designs*. CRC, Boca Raton, FL .
4. Crawford J. M., Dalal M., & Walser J. P. (1998). Abstract local search. In *Proceedings of the AIPS'98 Workshop on Planning as Combinatorial Search, Pittsburg, PA*.
5. Crawford, J. M., Ginsberg, M., Luks, E., & Roy A. (1996). Symmetry-breaking predicates for search problems. In Aiello, L. C., Doyle, J. & Shapiro, S. C., eds., *Proceedings of KR'96*, pages 148–159. Morgan Kaufmann, San Mateo, CA.
6. Das, S. R., & Geng, G. (2004). Correlated default processes: A criterion-based copula approach. *Journal of Investment Management*, 2(2).
7. Dinitz J. (2005). Handbook of combinatorial designs, new results. Available at http://www.emba.uvm.edu/~dinitz/newresults.html. (Accessed July 28th, 2005).
8. Fahle T., Schamberger, S., & Sellmann, M. (2001). Symmetry breaking. In Walsh, T., ed., *Proceedings of CP'01*, vol. 2293, LNCS, pages 93–107. Springer, Berlin Heidelberg New York.
9. Flener P., Frisch, A. M., Hnich, B., Kızıltan, Z., Miguel, I., Pearson, J., & Walsh T. (2002). Breaking row and column symmetries in matrix models. In P. Van Hentenryck (Ed.), *Proceedings of CP'02*, vol. 2470, LNCS, pages 462–476. Springer, Berlin Heidelberg New York.
10. Flener, P., & Pearson, J. (2002). Breaking all the symmetries in matrix models: Results, conjectures, and directions. In Flener, P. & Pearson, J., eds., *Proceedings of SymCon'02*. (Available at http://www.it.uu.se/research/group/astra/SymCon02/).
11. Flener, P., Pearson, J., & Reyna, L. G. (2004). Financial portfolio optimisation. In Wallace, M., ed., *Proceedings of CP'04*, vol. 3258, LNCS, pages 227–241. Springer, Berlin Heidelberg New York.
12. Gilkes, K., & Drexler, M. (December 2003). Drill-down approach for synthetic CDO Squared transactions. Standard and Poor's, New York.
13. Hall Jr., M. (1964). Block designs. In Beckenbach, E. F., ed., *Applied Combinatorial Mathematics*, chapter 13, pages 369–405. Wiley, New York.
14. Meseguer, P., & Sànchez, M. (2000). Tree-based Russian doll search: Preliminary results. In Rossi, F., ed., *Proceedings of the CP'00 Workshop on Soft Constraints*.
15. Puget, J.-F. (2003). Symmetry breaking using stabilizers. In Rossi, F., ed., *Proceedings of CP'03*, vol. 2833, LNCS, pages 585–599. Springer, Berlin Heidelberg New York.
16. Sivertsson, O. (December 2005). Construction of synthetic CDO squared. Master's thesis, Computing Science, Department of Information Technology, Uppsala University, Sweden. (Available as Technical Report 2005-042 at http://www.it.uu.se/research/reports/2005-042/).
17. Wetzel, G., & Zabatta, F. (1998). A constraint programming approach to portfolio selection. In *Proceedings of ECAI'98*, pages 263–264.