SIAM Review
Vol. 59, No. 2, pp. 393–422

The Secret Life of Keys: On the Calculation of Mechanical Lock Systems*

Christof Vömel†
Flavio de Lorenzi†
Samuel Beer†
Erwin Fuchs‡

Abstract. Keys and locks are an omnipresent fixture in our daily life, limiting physical access to privileged resources or spaces. While most of us may have marveled at the intricate shape of a key, the usually hidden mechanical complexity within a cylinder lock is even more awe-inspiring, containing a multitude of tiny movable parts such as springs and pins that have been precision-manufactured from highly customized materials using specialized fabrication techniques.

It is perhaps unknown that mechanical cylinder locks possess a number of important design constraints that uniquely distinguish them from their electronic counterparts. Aside from manufacturing costs, a cylinder's most significant limitations are the upper bound on its outer dimensions as well as the lower bound on the size of its internal mechanical security features (pins). Cylinders cannot be very large so that they still fit into doors and avoid the need for large keys. Pins cannot be too small in order to withstand wear and tear and provide appropriate physical security.

Even more complex than a single cylinder is the design of an ensemble of "related" locks as may occur in an apartment, office building, factory, or hospital, for example. Instead of controlling access to one resource, the set of open/not open relationships between all the keys and locks of such a *lock system* may encode a complex and diverse hierarchy of privileges for each individual key, from the benign opening of the front entrance by all staff to heavily restricted access to confidential documents or hazardous materials by selected personnel only.

For the mathematician or computer scientist, lock system design poses a fascinating array of theoretical and computational challenges. Abstraction via an algebraic model shows that cylinders and keys of a lock system can be represented within an upper semilattice, where the induced partial ordering establishes the must open/must not open functions. Finding an "equivalent" sub-semilattice within the semilattice over the set of those cylinders that are mechanically realizable then embodies the heart of the lock system calculation problem.

From the graph-theory point of view, finding such a sub-semilattice is exactly the famous graph subgraph isomorphism problem, which is known to be NP hard. As problem size increases, the solution via deterministic algorithms from combinatorial optimization quickly becomes intractable. In particular, conventional branch-and-bound strategies such as pruning, designed to limit systematic examination of the full search space, are ineffective for the graphs arising from lock systems.

For this reason, a pragmatic approach has to resort to randomized search space exploration via probabilistic heuristics that are by their very nature not guaranteed to work in all cases, but do yield results quickly enough to be useful for many relevant practical cases.

*Received by the editors July 9, 2015; accepted for publication (in revised form) August 1, 2016; published electronically May 5, 2017.
http://www.siam.org/journals/sirev/59-2/M103005.html
Funding: This research was funded by the Swiss Commission for Technology and Innovation, grant 13132.1 PFFLR-IW.

†Institute of Applied Mathematics and Physics, School of Engineering, ZHAW, Switzerland (christof.voemel@.zhaw.ch, flavio.delorenzi@.zhaw.ch, samuel.beer@.zhaw.ch).
‡KESO AG, ASSA ABLOY (Switzerland) Ltd., Richterswil, Switzerland (erwin.fuchs@keso.com).

Among the class of randomized optimization algorithms, simulated annealing proves to be particularly relevant for our setting. However, the vast available search space makes a vanilla algorithm impractical and adaptations are necessary. Parameters such as the cooling strategy have to be carefully tuned and the design of the error function for evaluating the quality of the tentative isomorphisms must weigh the tradeoff between expressiveness and runtime costs. Finally, in order to steer the algorithm toward promising choices in the search space, we combine annealing with a prefilter that guides selection based on an encoding computed in a preprocessing step.

The research described within this article was conducted in a research partnership between academia and industry; its goal is to develop a formalized model and an automated lock system calculation approach suitable for commercial use. Moreover, we wish to spark interest in the broader mathematics and computer science community in the challenging questions arising from this exciting industrial application.

Key words. cylinder lock, lock system, algebraic model, semilattice, combinatorial optimization, NP completeness, directed acyclic graph, subgraph isomorphism, global optimization, heuristic, randomized algorithm, simulated annealing, mathematics in industry, artificial intelligence

AMS subject classifications. 68R10, 90C27, 90C35, 06A07, 06A12, 03G10

DOI. 10.1137/15M1030054

1. Introduction.

The door slams shut, and almost instantly there arises that gnawing feeling of having forgotten something. But what? Oh no!

At some point in our daily lives, almost all of us have found ourselves wondering how to open a lock without ready access to a suitable key. Whether we have forgotten or lost that key, or indeed never possessed it, opening that lock can range from a curiosity-piquing mystery to a vital necessity. Ubiquitous and yet exuding secrecy, locks indeed have sparked the interest of many people, including celebrated magicians [30] and Nobel Prize–winning physicists [42]. Unsurprisingly, there is a host of literature teaching amateurs and professionals about the internals of locks and how to bypass their mechanical security mechanisms; see, for example, [46, 50, 53].

While electronic locks are becoming more available, mechanical locks remain the de-facto standard in many environments [34, 40]. Facing the crowd of amateur and professional lock openers, a locksmith has to invent and carefully design a combination of mechanical security mechanisms. These establish the desired control features in a *cylinder* lock to allow authorized *keys* access and deny unauthorized ones, as well as to withstand break-ins. Having evolved significantly over time [53], an "ordinary" door lock nowadays may contain a multitude of tiny mechanical devices precision-manufactured at tolerances of micrometer scale; see Figure 1.1.

Mechanical cylinder locks possess a number of challenging design constraints against which security requirements have to be carefully weighed. Besides the obvious manufacturing costs, a cylinder's most limiting constraints are its dimensions (height, width, and depth), which are determined by its environment (for example, a door) as well as people's aversion to keys that are too large or too heavy. Furthermore, to protect against wear and tear (for example, from frequent use, soiling, or changing climate), any mechanical security feature has to be especially durable and cannot be too small. Thus, the maximum space available inside the cylinder and the minimum size of the mechanical devices limit the number of security features that can be put inside a cylinder.

Of great importance for this paper is another somewhat less obvious yet equally challenging complication, namely, that locks usually do not exist in isolation. Most

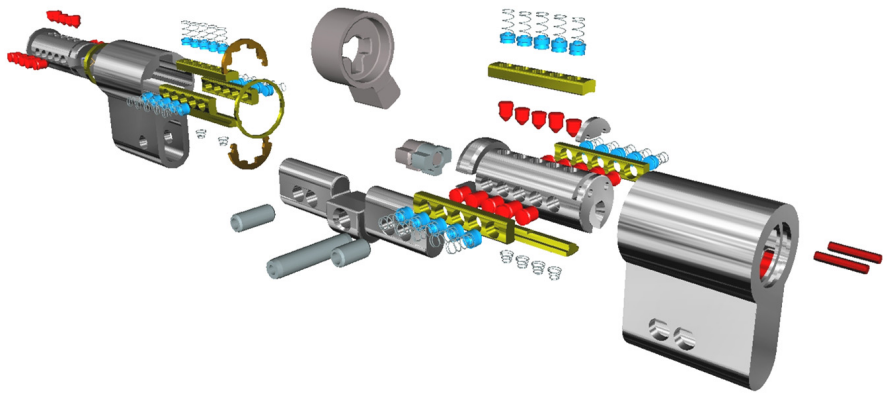

Fig. 1.1 *Precision mechanics inside a modern door lock. There is a locking bolt centered between two cylinders; the security mechanism is based on spring-loaded arrays of tumbler pins. More details on lock technology are provided in section 2. Illustration courtesy of ASSA ABLOY (Switzerland) Ltd.*

locks occur as part of *lock systems*, that is, ensembles of locks that relate to each other conceptually and functionally. As a simple "toy" example, consider a two-family duplex house where each of its two apartment units has its own door lock. Each lock *must* be opened by the inhabitant's key, but *must not* open to the other unit's key. Housekeeping and management, however, need to be able to open both doors.

With the duplex house example at hand, we can immediately recognize an abundance of more diverse and complicated lock systems occurring in apartment or office buildings, hotels, hospitals, or universities, for example. Their defining characteristics are two complementary sets, cylinders and keys, where each key *must* open one or more cylinders but *must not* open the others, and where each cylinder *must* be unlocked by some of the keys, but *must not* be unlocked by the others.

The typical abstract specification of a lock system thus consists of a rectangular table A, where entry $A(i, j)$ indicates whether cylinder i must or must not be unlocked by key j. For an electronic lock system, it would suffice to store this table in binary form and simply check the access privileges due to entry $A(i, j)$ when key j is presented to the electronic lock i. However, for mechanical locks, table A must be *implicitly* encoded in the selection of the security features for each cylinder and each key, because its size typically far exceeds available space. Moreover, the encoding should be as *compact* as possible to make good use of the limited cylinder space as well as save resources and thereby reduce costs.

For a first glance, consider Table 1.1 and the previous duplex house example.

Not wanting to delve into the specifics of mechanics here, we can think of a *challenge-response protocol* where we model a security feature as a part of a "challenge" posed to any key. A number of such features are placed in various locations in the cylinder, and if a key "responds" to all of them correctly, that is, the key possesses a valid "matching twin" feature at each of the necessary locations, the cylinder *unlocks* or *opens*. Note that abstract modeling of the mechanics not only allows us to reason effectively, but also accommodates the constant evolution of the mechanical design

Table 1.1 *Specification tables for two apartment door cylinders C1 and C2 as well as the two inhabitants' keys I1 (opening C1 but not C2) and I2 (opening C2 but not C1) and the management key M (opening both C1 and C2). "1" signifies must open; "0" signifies must not open.*

	I1	I2	M
C1	1	0	1
C2	0	1	1

due to improved manufacturing tolerances and materials as well as new challenges such as that posed by 3D printing to anticopy protections.

The duplex house example is easy enough to use to reason about the smallest set of security features necessary to implement the specification in Table 1.1. If cylinder C1 has a feature at a position A but lacks one at another position B, i.e., "[A B] := [1 0]," and cylinder C2 is designed complementarily with a feature in position B but not A, i.e., "[A B] := [0 1]," then key I1 with twin features "[1 0]" opens only C1 but not C2, and key I2 with "[0 1]" only opens C2 but not C1. The management key M will be "[1 1]" and open both C1 and C2. (Note that in order to guarantee these relations, each of the two cylinders would only require its own *distinguishing* feature to be present and would not check for others.)

Assigning each cylinder its own distinctive feature in the above fashion is always *valid*, albeit *inefficient*, as the limited space inside the cylinders is exhausted very quickly. Fortunately, it is possible to avoid this scenario in many practically relevant cases. For example, the most compact approach for a similar "quinplex" house with five apartments, each accessible by its own key and the management key, requires only four, not five, security features. The curious reader may pause here briefly, find the solution, and also ponder related problems such as the general n-plex house. In general, the larger a lock system and the more elaborate its must open/must not open relations, the harder it becomes to make optimal use of resources; a complexity analysis shows that the optimal *encoding problem* is NP hard; see [12, 28, 29, 38, 54].

In order to physically realize the actual lock system, we must ultimately return to the question of mechanical feasibility. Due to the aforementioned minimum dimension requirements as well as their specific shapes, mechanical security features cannot be arbitrarily placed inside a cylinder. For example, the presence of a feature in one spot precludes the placement of others too close by. Thus, we would have to find mechanically suitable places for all the necessary security features within the cylinders. For practical and logistical reasons, however, manufacturers typically start from another angle and first consider standardized sets of physically possible cylinder configurations. In this setting, we then have to find a subset of cylinders that is rich enough to meet the needs of the lock system at hand; that is, we must correctly select mechanical matches for each specified cylinder such that the mechanical keys exactly fulfill the required must open/must not open relations. This *matching problem* is also NP hard; it is in fact a graph subgraph isomorphism problem [24, 39]; see section 3.

With increasing problem size, deterministic search algorithms [61, 62, 67, 68] quickly become intractable. In particular, graph pruning strategies designed to limit exhaustive examination of the full search space, as, for example, is introduced in the VF2 algorithm [15, 16, 65], do not yield significant gains for lock system graphs, due to high degrees of symmetry in the mechanics. As the only viable alternative, *heuristic* search space exploration [2, 10, 48, 52, 57, 63] offers a successful paradigm that tries

to benefit from *random* variations to systematic search. Readers unfamiliar with or unconvinced by the idea may be swayed by thinking of trading in (quasi) certainty of a *very* long search for a hopeful (albeit ultimately uncertain) chance at reasonably quick success. Typically, one starts from an initial *guess*, inspects the *local* neighborhood for a solution, and then restarts if necessary with another guess. Randomized "guessing" crucially drives innovation, or evolution, in the *global* search for a solution and can be done either uninformed or guided by insights gained from previous experience or simplified models, say. Moreover, an *error function* measures the degree of deviation from an optimum. It can serve as an objective function for optimization techniques such as "hill-climbing" type search [57, 63] in local neighborhoods or at least help determine which global search regions are more promising than others.

While any search methodology based on probabilistic heuristics is not guaranteed to be uniformly reliable or efficient [72], a carefully tailored algorithm can still yield good results quickly enough to be useful in many relevant practical cases. We show that among the large class of stochastic optimization algorithms [10, 37, 48, 59, 63], simulated annealing [1, 2, 10, 13, 20, 35] can be enhanced by encoding techniques [12, 28, 29, 38, 54] to successfully master the challenges in our setting; see sections 4 and 5.

We now invite the reader to accompany us into the universe of lock systems. That key in our pocket also unlocks a world at the crossroad of algebraic structure, computational complexity, combinatorial optimization, and randomized algorithms.

2. Physical Design Aspects of Mechanical Lock Systems. This section gives the reader a more precise idea of the "inner life" of locks and lock systems, a deeper look behind the scenes, beyond what has been merely sketched in the introduction. We first inspect the interior of pin-tumbler cylinder locks in section 2.1. Section 2.2 showcases an annotated commercial specification chart of a small lock system. Finally, section 2.3 places the discussion into the larger context of adjacent topics that are related but not relevant to the core of this article. All illustrations in this section appear courtesy of ASSA ABLOY (Switzerland) Ltd.

2.1. Overview of Mechanical Cylinder Lock Technology. Figures 2.1 and 2.2 illustrate the opening mechanism in a cylinder lock based on the common pin-tumbler technology. As mentioned in the introduction, keys and locks implement a challenge-response protocol in which the cylinder presents a "challenge" to be met with a valid "response" by a key. This is now exhibited in more detail.

The longitudinal key profile (see Figure 2.1) is cut out in several concave parts, with a shape reminiscent of an "inverted mountain range." The specific shape of the profile corresponds to the key's "challenge response" when inserted into a cylinder.

The mechanical cylinder (see Figure 2.2) consists of an inner rotor part housed within an outer stator casing. When unlocked, the rotor is allowed to turn and open the lock, typically by removing a locking bolt. The locking mechanism consists of spring-loaded pins (stacks of thin metal plates) that each sit within vertical holes. Without the right key, the springs push the pins so that they protrude without alignment into the rotor, thereby preventing it from turning. Adjusting the pins in just the right way so that they all exactly align at the shear point between stator and rotor represents the "challenge" posed by the cylinder to an opening key.

Note that the situation depicted in Figure 2.2 is typical in that in a single row, there is space for lining up only five to six spring-loaded pins. To increase the number of available opening functions for larger lock systems, however, each pin may be allowed to align at several places. This can be achieved by stacking several smaller movable plates on top of each other instead of using a single larger contiguous pin;

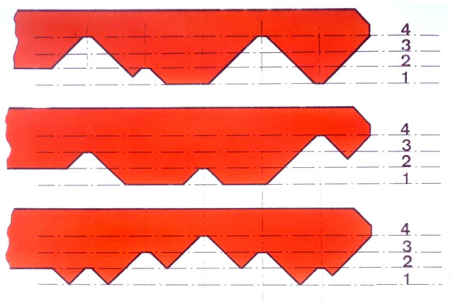

Fig. 2.1 *Key profile variants. A key possesses multiple concave cuts of specified discrete levels.*

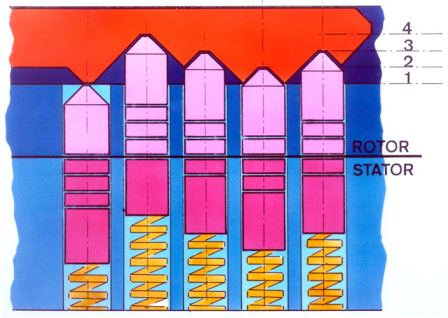

Fig. 2.2 *Opening mechanism of a pin-tumbler lock. Vertical springs push each pin upwards. By inserting the key, the pins push into the concave sections and thereby may adjust their height, counterbalancing the spring tension.*

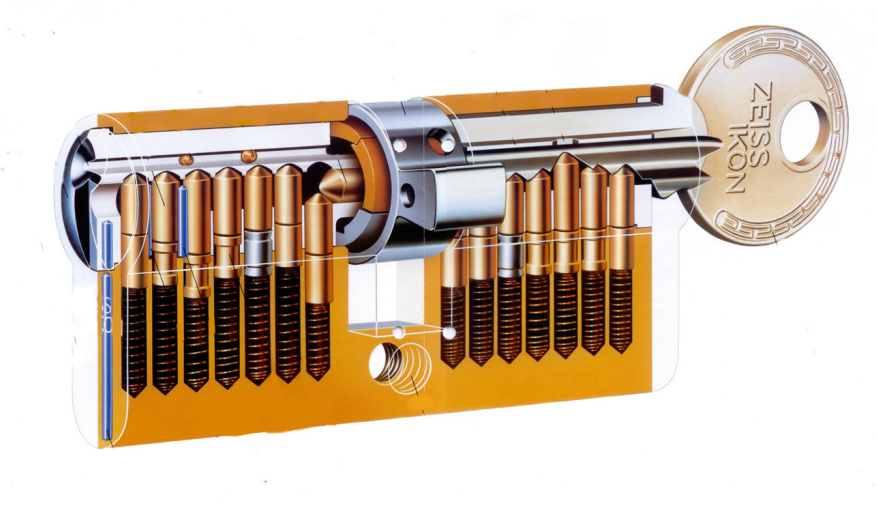

Fig. 2.3 *Two-sided door lock with two cylinders (the left one locked with unaligned pins, the right one unlocked with pins aligned at the shear point) and a locking bolt in the center.*

see Figure 2.2 where, for illustration, multiple alignments at four levels are possible for each pin.

For a comparison of locked and unlocked pin configurations, Figure 2.3 shows a two-sided assembly as one would find it in a door lock.

As an alternative to a single row of "stacked" vertical pins, we may also consider arranging simple pins at multiple angles inside the cylinder; see Figures 2.4 and 2.5. In this configuration, a cylinder may contain pins at up to 15 places, each needing to be aligned for the rotor to turn.

For the remainder of this paper, we focus on the second type of mechanical cylinder consisting of three rows of simple pins, as these are of relevance for our industrial

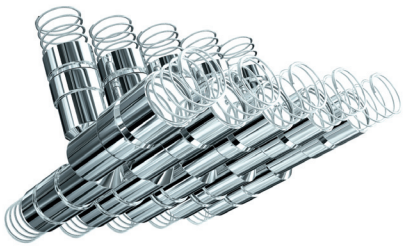

Fig. 2.4 *Three-directionally angled pin placement.*

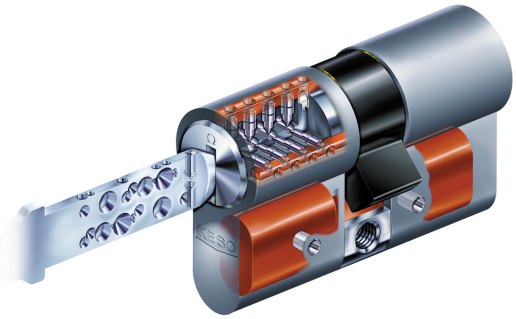

Fig. 2.5 *Cylinder and corresponding key. The key has a* 180° *rotation symmetry along its longitudinal axis, mostly for sake of convenience; there are three unique rows of pins.*

partner. However, we see no reason why the theory and algorithms developed in what follows would not apply to the first kind; see [41] for a previous investigation of some of the aspects.

Note that with several rows of pins having to fit into the available mechanical cylinder space, typically there is not enough room for stacked pins. To provide a greater selection of different cylinders for large lock systems, we can trade some security for variability: instead of a single cylinder of ultimate "strength" with the maximum number of 15 possible pins, we can reduce the number of pins per cylinder and create more of them by combinatorics. In practice, one typically chooses between seven and nine pins (from 15) per cylinder, corresponding to the highest binomial coefficients. The mechanical cylinders are then produced from the, say, $\binom{15}{7}$ combinatoric variations. Interestingly though, not all variations will be manufactured; instead, a few of the pins are fixed to separate subcontractors, distributors, and regional markets to prevent unauthorized reimports or price cutting. Lastly, by varying placement locations and changing the distance between pins, we can create additional pin configurations and corresponding sets of mechanical cylinders; see section 5.4.2.

2.2. Specification of Locking Functionalities. To give the reader a glimpse of the commercial specification and administration of lock systems, Figure 2.6 shows an annotated commercial lock system specification chart. Many of the details available in this chart are not of immediate relevance for this paper. However, they represent necessary information for the manufacturer to maintain the lock system for the client and nicely illustrate the complexity of the overall framework. Section **G** represents the core of the chart, stating in "sparse" notation the rectangular table of open/not open relations between the cylinders and keys, as described in the introduction.

A slightly closer look at the details of sections **C, D, G** in Figure 2.6 reveals a lock system consisting of seven cylinders and four keys. Cylinders 1–4 belong to the entrance doors of four individual apartments, each to be opened by its own key. Moreover, cylinders 5–7 belong to communally shared doors at the building entrance and in the cellar, all to be opened by each of the four keys.

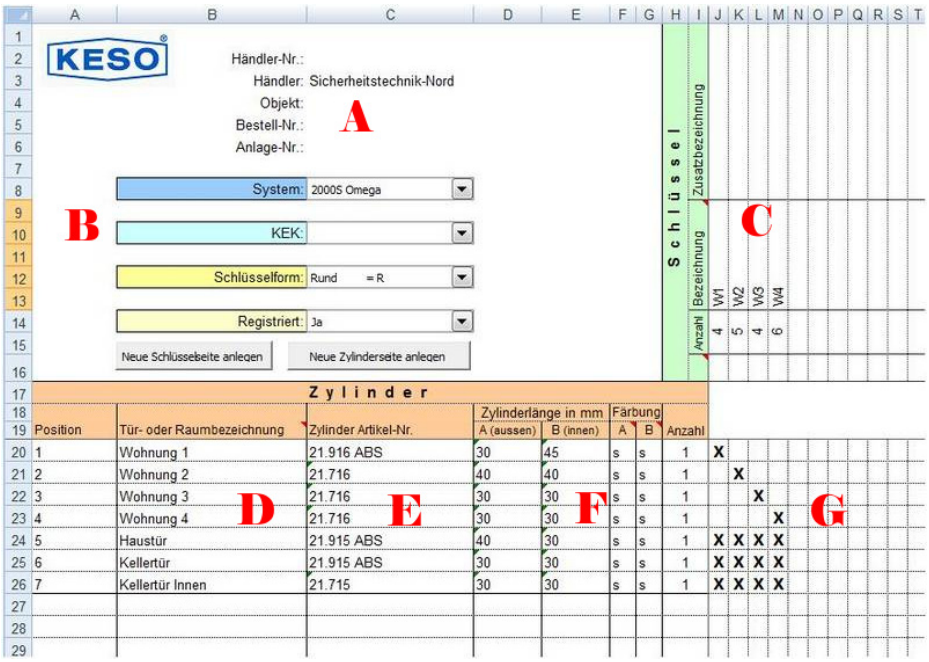

Fig. 2.6 *Commercial lock system specification chart. Sections have been annotated in bold letters for readers who cannot understand German:* ***A*** *refers to details of the subcontractor responsible for installation and maintenance;* ***B*** *gives details about the lock system category (which influences price, features, and complexity);* ***C*** *and* ***D*** *list the keys (in this case four) and door locks (seven in this example), respectively;* ***E*** *states the manufacturer codes of those mechanical cylinders that have been installed; and* ***F*** *adds further details such as measurements. Finally,* ***G*** *specifies with "**X**" what key must open which door; a missing "**X**" indicates "must not open."*

2.3. Related Subjects Nonessential to this Paper. To conclude, we briefly discuss related topics that are of only peripheral interest to the core of this paper and hence will be omitted from the following discussion. Curious readers inspired by this short summary are encouraged to follow up at their own leisure.

The technologies used for precision manufacturing, metal forming, and materials engineering directly constrain the choices of pin layout, placement, and complexity inside a cylinder. However, this information is proprietary and protected as a trade secret. Therefore, we will leave the discussion at the general level of section 2.1.

Second, a security infrastructure might be based solely on electronics in the form of identification badges or key cards based on radio-frequency identification (RFID) [26, 66], offering great flexibility. When a key gets lost, its electronic component can be deactivated; key privileges can also be easily reconfigured without physical access or modifications. Furthermore, key use can be monitored and logged.

One can also leverage these benefits within classical lock systems by upgrading with mechatronic components. For example, mechanical keys can be additionally equipped with an RFID transponder chip that provides a secondary authentication and access control factor by a matching RFID reader integrated into the cylinder; see Figure 2.7. (Alas, this can increase costs by a factor of up to four; for this reason, about 75–80% of mechanical locks have no mechatronic components.)

Fig. 2.7 *RFID-enhanced cylinder lock.*

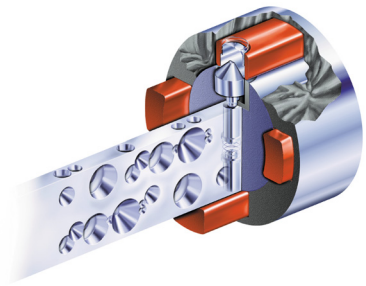

Fig. 2.8 *Anticopy protection with a spring-loaded pin installed on a key.*

Another topic that is beyond our scope concerns mechanical key modifications, for example, for anticopy protection. While chip technology can be an effective means to that end, it is also possible to include miniature dynamic elements like a spring-loaded pin on an otherwise completely static key; see Figure 2.8.

Last but not least, we do not consider special high-security features, some of which are briefly discussed in, for example, [53]. In any case, security experts [7, 22, 23] would remind us that locks in general are only one of potentially many components of a security infrastructure whose overall level of protection depends proverbially on the "weakest link in the chain." These reservations notwithstanding, we are now ready to proceed to the main part of the article and address all the fascinating theoretical and computational challenges that are offered by lock systems.

3. Algebraic Model and Structure. With a more detailed picture of mechanical cylinders and keys in hand, we are now ready to approach the lock system calculation and marry the demands of the specification chart with the restrictions imposed by the mechanics. We first introduce a binary vector model for cylinders and keys in section 3.1. Section 3.2 then discusses partial ordering aspects and the upper semilattice structure of the key hierarchy; section 3.3 presents the visualization by directed acyclic graphs. Section 3.4 formally states the combinatorial optimization problems at the core of lock system calculation.

3.1. Correct Encodings with Binary Vectors. To begin, let us return to the simple lock system presented in the introduction, consisting of two apartments with their individual keys and a manager key able to access both. Table 3.1 states the corresponding specification chart in the sparse notation introduced in section 2.2.

Table 3.1 *Lock system specification chart in sparse notation for two apartment door cylinders $C1$ and $C2$ as well as the two inhabitants' keys $I1$ (opening $C1$ but not $C2$) and $I2$ (opening $C2$ but not $C1$) and the management key M (opening both $C1$ and $C2$). An "X" means "must open," its absence signifies "must not open."*

	Key I1	Key I2	Key M
Cylinder C1	X		X
Cylinder C2		X	X

As sketched in the introduction, we can then ponder what set of mechanical features those cylinders and keys *at least* must have to implement these demands. Clearly, there must be at least one pin in cylinder C1 that is not aligned when key I2 is presented, but aligned for I1. Conversely, there is another pin in cylinder C2 that is unaligned for key I1 but aligned with key I2. Moreover, no more than these two pins are needed since key M, as master key, combines features of I1 and I2.

As an algebraic model, let us consider a set of binary (bit) vectors. For a cylinder, a "1" at position i indicates the presence of a simple pin (a "challenge" to a key), and a "0" indicates its absence. Correspondingly, for a key, a "1" at position i indicates a valid response to the pin at that position (mechanical alignment at the shear point), while "0" means a missing response (no pin alignment). An *encoding* e is a injective mapping of the two sets of cylinders and keys of a lock system into a set of binary vectors of fixed length, where the exact length varies and depends on the specification. As an example, consider the encoding of the lock system from Table 3.1 given in Table 3.2. For simplicity, the binary vectors are represented as binary numbers; the prefix "0b" is there for clarity but will be omitted in what follows.

Table 3.2 *Encoding e of the lock system specification chart from Table* 3.1 *by binary vectors (numbers). The prefix "0b" is used to indicate the binary numbe system. The encoding represents features that are present ("1") or missing ("0") and are used for the challenge-response protocol implementing the must open/must not open specifications. For details, see Theorem* 1.

Cylinder $\mathbf{x}$	Encoding $\mathbf{e(x)}$	Key $\mathbf{x}$	Encoding $\mathbf{e(x)}$
Cylinder C1	0b10	Key I1	0b10
Cylinder C2	0b01	Key I2	0b01
		Key M	0b11

We call an encoding *correct* if the fundamental relations in the following theorem hold for all pairs of keys and cylinders in accordance with the lock system specification. We will be particularly interested in finding a correct encoding with bit vectors with length as short as possible; see section 3.4.

THEOREM 1.

1. *The must open relation is stated as follows:*

$$\text{key } \mathbf{k} \text{ opens cylinder } \mathbf{c} \quad \Leftrightarrow \quad \mathbf{e}(\mathbf{k}) \wedge \mathbf{e}(\mathbf{c}) = \mathbf{e}(\mathbf{c}), \tag{1}$$

$$\Leftrightarrow \quad \mathbf{e}(\mathbf{k}) \vee \mathbf{e}(\mathbf{c}) = \mathbf{e}(\mathbf{k}), \tag{2}$$

   where $\wedge$ *and* $\vee$ *denote the logical* AND *and* OR*, executed place-by-place (bitwise) for the encodings* $e(k)$ *of key* k *and* $e(c)$ *of cylinder* c*.*

2. *The must not open relation is stated as follows:*

$$\text{key } \mathbf{k} \text{ does not open cylinder } \mathbf{c} \quad \Leftrightarrow \quad \mathbf{e}(\mathbf{k}) \wedge \mathbf{e}(\mathbf{c}) < \mathbf{e}(\mathbf{c}), \tag{3}$$

$$\Leftrightarrow \quad \mathbf{e}(\mathbf{k}) \vee \mathbf{e}(\mathbf{c}) > \mathbf{e}(\mathbf{k}). \tag{4}$$

   Here, $<$ *and* $>$ *denote lexicographical comparison* [37] *of the encodings, or the linear order of the equivalent binary numbers.*

Relation (1) and its dual (2) formalize the intuitive notion presented in the introduction that in order for a key to open a cylinder, it must possess matching security features at all the places where the cylinder requires them. (Other features may be present on the key but are not checked by the cylinder.) Relations (3) and (4) say

that in order for a key to not open a cylinder, it must lack a security feature in at least one of the places where the cylinder requires them.

It can be seen that each cylinder c partitions the set of keys into those that open c and those that do not. Conversely, each key k induces an analogous dual partition on the set of cylinders. Assuming that keys do not contain redundant information, we can deduce an important generative principle based on the encoding of cylinders.

THEOREM 2. *For any key k, let $C(k)$ denote the set of cylinders opened by k. Then*

$$\mathbf{e(k)} = \bigvee_{\mathbf{c} \in \mathbf{C(k)}} \mathbf{e(c)}. \tag{5}$$

Observe that a lock system with n different cylinders can always be encoded using bit vectors of length n, and there can exist up to $2^n - 1$ different keys. Depending on which of these keys are specified and which are unwanted, shorter encodings may be possible. To see this, let us return to a question posed in section 1: what if there had been five instead of two apartments in the example from Table 3.1? Once again, combinatorics in the form of binomial coefficients comes to our aid; see Table 3.3: instead of a single feature, permute a pair of features within a group of four possible locations, resulting in a total of $\binom{4}{2} = 6$ cylinder encodings. (Thus, there is even a spare one for future use, say, when a key gets lost and the lock needs to be replaced.)

Table 3.3 *Extension of the example in Table* 3.2. *Encoding five cylinders with four instead of five bits. Rather than a single unique feature, each cylinder has a unique combination of two out of a group of four.*

Entity $\mathbf{x}$	Encoding $\mathbf{e(x)}$
Cylinder C1, Key I1	0b0011
Cylinder C2, Key I2	0b0101
⋮	⋮
Cylinder C5, Key I5	0b1100
Key M	0b1111

However, there is a subtle yet vital restriction to be aware of when using combinations of features to encode a group. For example, what if another key was introduced that had to open cylinders C1 and C5, but not C2–C4? With the encoding from Table 3.3, this would be infeasible. By Theorem 2, the resulting key would be equal to the master key M and thus would open all five cylinders, not just the desired two! Hence, the encoding would not be correct. In conclusion, the optimization problem of finding a shortest correct encoding is critically constrained by the set of must open/must not open requirements for the specified keys.

3.2. Partial Ordering and Upper Semilattice Structure. In the model introduced in section 3.1, there is no difference in the encoding of a cylinder and a "singleton" key that opens this cylinder but no others: obviously, such a key needs all the features present in that cylinder and does not require any further ones present in other cylinders. As a consequence, each cylinder c can be equivalently represented by its singleton key $k =: k_c$ satisfying

$$e(c) = e(k_c) \wedge e(c) = e(k_c). \tag{6}$$

This simplification then allows us to condense the lock system representation into a hierarchy consisting solely of keys and a binary relation $\geqq$:

$$\text{key } \mathbf{k} \text{ opens cylinder } \mathbf{c} \quad \Leftrightarrow \quad \mathbf{k} \geqq \mathbf{k_c}, \tag{7}$$

$$\text{key } \mathbf{k} \text{ does not open cylinder } \mathbf{c} \quad \Leftrightarrow \quad \mathbf{k} \ngeqq \mathbf{k_c}, \tag{8}$$

$$\text{key } \mathbf{k1} \text{ opens all cylinders opened by key } \mathbf{k2} \quad \Leftrightarrow \quad \mathbf{k1} \geqq \mathbf{k2}. \tag{9}$$

(For the lock system encoded in Table 3.2, we thus have $\mathbf{M} \geqq \mathbf{I1}$ and $\mathbf{M} \geqq \mathbf{I2}$. Note, however, that $\mathbf{I1} \ngeqq \mathbf{I2}$ and $\mathbf{I2} \ngeqq \mathbf{I1}$.) It can be readily verified that this relation satisfies the following three important properties for all keys k_i, k_j, k_k:

- Reflexivity: $k_i \geqq k_i$.
- Antisymmetry: $k_i \geqq k_j, k_j \geqq k_i \Rightarrow k_i = k_j$.
- Transitivity: $k_i \geqq k_j, k_j \geqq k_k \Rightarrow k_i \geqq k_k$.

(For the useful antisymmetry property to hold, we make the reasonable assumption that any redundant duplicates have been eliminated from the lock system. Although not strictly necessary, this considerably simplifies the discussion by considering partially ordered instead of merely preordered sets [60].)

Without loss of generality, in what follows we further assume the existence of a master key opening all locks. In this case, any nonempty finite subset of keys has a least upper bound, or *join* [9, 19]. However, $k(c_1) \wedge k(c_2)$ does not exist for cylinders $c_1 \neq c_2$. These findings are summarized as follows.

THEOREM 3. *The must open/must not open relations among cylinders and keys, as defined in a lock system specification, induce a partial order* (9) *on the partially ordered set (poset) of keys of the lock system. When a master key exists, the set of keys has an upper semilattice structure.*

It is important to note that the structure described in Theorem 3 is inherent to the original problem and not an artifact introduced by our particular model. We thus carefully distinguish between the abstract partial order $\geqq$ of keys and any of the operators $\vee, \wedge, <, >$ tied to the specific encoding from section 3.1. However, the encoding can be viewed as a partial order preserving *embedding* into some complete lattice 2^S.

3.3. Visualization as Directed Acyclic Graphs. Among the many equivalent representations of partial orders (see, for example, [9]), we prefer the partial order by a directed graph $G = (V, E)$. Each vertex $v_i \in V$ represents a unique key, and each directed edge $e = (v_j \to v_i) \in E$ represents the order relation $v_i \geqq v_j$ between two associated keys $v_i \neq v_j$. We call v_i an *ancestor* of v_j, and v_j a *descendant* of v_i. An immediate ancestor is called a *parent*, and an immediate descendant is a *child*.

The antisymmetry from section 3.2 guarantees that G cannot have (directed) cycles [17, 65]. Moreover, a bijective mapping f between the poset of keys and the vertices of a directed graph is called an *isomorphism* when $k_i \geqq k_j$ if and only if there is a directed edge from $f(k_j)$ to $f(k_i)$. (Ancestors thus correspond to stronger keys whose opening functions include those of the weaker keys that are their descendants.)

THEOREM 4. *The poset of keys from a lock system specification is isomorphic to a connected directed acyclic graph (DAG). The master key corresponds to the unique root node; each leaf corresponds to the singleton key k_c of a unique cylinder c.*

In order to decrease visual complexity, we prefer a "sparse" representation of the partial order called a *transitive reduction* [4]: we omit all edges which are implicit through transitivity and which can be reconstructed via *transitive closure* [65]. Furthermore, we may elect to not show singleton keys at all if there are too many.

Figure 3.1 shows the DAG for the lock system of Table 3.1. Each node but the root has a unique parent; the DAG is a rooted *tree*. (The precise term is *arborescence* [37], a DAG whose underlying undirected graph is a tree and where the single root node has a unique path to every other node. For easier reading, we prefer to use the term "tree.")

The DAG for a slightly augmented system is shown in Figure 3.2. Here, the inhabitants' keys I1 and I2 both open the entrance E, in addition to their respective apartment doors A1 and A2. In this case, the DAG has lost the tree property; node E has more than one parent and is called *cross-locked*. While it is benign in this case, cross-locking can cause severe problems for calculations; see sections 4 and 5.

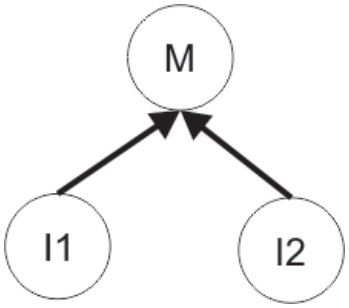

Fig. 3.1 *DAG for the lock system from Table* 3.1.

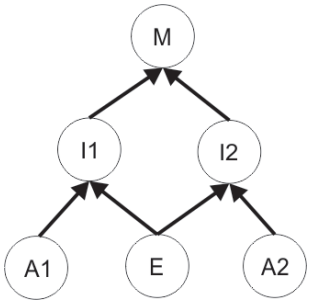

Fig. 3.2 *DAG for this lock system augmented by a shared entrance E.*

Comparison of Figures 3.1 and 3.2 reveals a particularity of lock system graphs: any node that is not a leaf must have at least two children. The directed edge $f(k_j) \to f(k_i)$ implies that key k_i is stronger than k_j; that is, $C(k_j) \subsetneq C(k_i)$. The additional cylinder(s) in the nonempty set $C(k_i) \backslash C(k_j)$ will be represented by at least one more directed edge incoming to v_i. This is why in Figure 3.1 keys I1 and I2 are identical with the singleton keys (instead of forming short chains), whereas in Figure 3.2 they become parents of A1, E and A2, E, respectively. (With omission of the singletons, the lock system in Figure 3.2 would *look* the same as in Figure 3.1, where I1, I2 are "group" keys opening a group of two cylinders each.)

Finally, note that the arrow direction itself is nothing but a convention; as long as it is consistent, we are free to choose it to point either way. Having all arrows point up toward the root (our choice) reminds us that stronger keys *inherit* all the features of their descendants. Conversely, the alternative of arrows pointing away from the root downwards indicates that *missing* features are inherited from ancestors.

3.4. The Fundamental Encoding and Matching Problems in Lock System Calculation. We are finally ready to state precisely the fundamental encoding and matching problems from combinatorial optimization arising in lock system calculation. Both are computationally difficult to handle as they can be shown to belong to the class of NP hard problems. (For background on NP complete decision and NP hard optimization problems, see, for example, [17, 24, 37, 49].)

DEFINITION 5 (encoding problem). *For a given lock system specification, find a correct encoding of shortest bit vector length.*

The encoding reflects what might be regarded as the structure and complexity of the lock system specification. Despite superficial similarities, we do not see any

useful connections to optimal encoding problems from information theory [17, 18, 43, 56]. Theorem 6 categorizes the complexity, and we will elaborate on the result in section 4.1.

THEOREM 6 (see [12, 28]). *The encoding problem is* NP *hard.*

The DAG can provide some estimates of the encoding length. The total number of leaves is an upper bound that in practice, typically and fortunately, is a gross overestimate. (Recall that equipping each cylinder with its own distinctive feature is usually unnecessary and inefficient.) A lower bound is the number of nodes on the longest directed path from the root to any leaf. (This bound is sharp for a binary tree. General DAGs are more complicated, however; see section 4.)

For the definition of the matching problem, recall that mechanical cylinders consist of a fixed number of simple pins, say, seven, placed at some of 15 total possible places. A *representation* r is an injective mapping of a set of mechanical cylinders into the set of binary vectors of length 15, where pin locations have been linearly ordered according to a predefined, fixed scheme. To distinguish the mechanical cylinders from those abstractly defined in the lock system specification, we denote the former by z_i (and the latter by c_i). Also note the difference between specification-dependent encoding (e.g., Table 3.3) and mechanics-tied representation (see Table 3.4).

Table 3.4 *Example of a representation of two mechanical cylinders with* 7 *pins among* 15 *places. As a visual aid, the* 15 *places are partitioned into groups of five, corresponding to the three rows of* 5 *pins, each in a mechanical cylinder.*

Cylinder $\mathbf{z}$	Representation $\mathbf{r(z)}$
Z1	01010 00011 10101
Z2	01010 11000 10110

Analogous to Theorem 2, we can find the corresponding representation $r(k)$ of a mechanical key k from its opening functions.

THEOREM 7. *For any mechanical key k, let $Z(k)$ denote the set of mechanical cylinders opened by k. Then the key's representation is*

$$\mathbf{r}(\mathbf{k}) = \bigvee_{\mathbf{z}\in\mathbf{Z}(\mathbf{k})} \mathbf{r}(\mathbf{z}). \tag{10}$$

A *matching* is an injective mapping of the cylinders $\{c_i\}$ from the specification into a (possibly much larger) set of mechanical cylinders $\{z_i\}$. The matching is *correct* if the following analogue of Theorem 1 holds for all mechanical cylinder and key representations, for the must open/must not open relations from the specification.

THEOREM 8.

1. *The must open relation is stated as follows:*

$$\text{key } \mathbf{k} \text{ opens mechanical cylinder } \mathbf{z} \quad \Leftrightarrow \quad \mathbf{r}(\mathbf{k}) \wedge \mathbf{r}(\mathbf{z}) = \mathbf{r}(\mathbf{z}), \tag{11}$$

$$\Leftrightarrow \quad \mathbf{r}(\mathbf{k}) \vee \mathbf{r}(\mathbf{z}) = \mathbf{r}(\mathbf{k}). \tag{12}$$

2. *The must not open relation is stated as follows:*

$$\text{key } \mathbf{k} \text{ does not open cylinder } \mathbf{z} \quad \Leftrightarrow \quad \mathbf{r}(\mathbf{k}) \wedge \mathbf{r}(\mathbf{z}) < \mathbf{r}(\mathbf{z}), \tag{13}$$

$$\Leftrightarrow \quad \mathbf{r}(\mathbf{k}) \vee \mathbf{r}(\mathbf{z}) > \mathbf{r}(\mathbf{k}). \tag{14}$$

Note that by deriving the key representations from those of the mechanical cylinders via Theorem 7, the must open relations (11) and (12) are automatically fulfilled. The correctness of the matching depends on guaranteeing the must not open relations (13) and (14), since a caveat similar to the one at the end of section 3.1 applies. We are now ready to define the matching problem.

DEFINITION 9 (matching problem). *Find a correct matching for the cylinders* $\{c_i\}$ *from a given lock system specification within a set of mechanical cylinders* $\{z_i\}$.

To classify the complexity, let us revisit the connection between the lock specification and its DAG discussed in section 3. Note that there is another, much larger DAG corresponding to the mechanical keys over the set of all mechanical cylinders. Now, for two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, a bijective mapping $f : V_1 \to V_2$ is a *graph isomorphism* when there is an edge $e_1 = v_i \to v_j$ in E_1 between vertices v_i and v_j if and only if there is an edge $e_2 = f(v_i) \to f(v_j)$ in E_2 between $f(v_i)$ and $f(v_j)$. (For example, two different correct matchings may still yield, via (10), exactly the same representations for all mechanical keys but the singletons; in this case, the two matchings are *equivalent* and "their" DAGs are canonically isomorphic.) The *subgraph isomorphism* problem consists of finding a subgraph $G_2' = (V_2', E_2') \subset G_2$ such that G_1 is isomorphic to G_2'. Here $|V_2'| = |V_1|$ and $e = v_i \to v_j \in E_2'$ if and only if $e \in E_2$ and $v_i, v_j \in V_2'$ (that is, exactly those edges "within" G_2' are considered). By matching specified cylinders $\{c_i\}$ to mechanical counterparts $\{z_i\}$, we implicitly define a subgraph of the "mechanical DAG," and then checking the matching for "correctness" in the sense of Theorem 8 is nothing other than testing for subgraph isomorphism.

THEOREM 10 (see [24]). *The matching problem is equivalent to the graph subgraph isomorphism decision problem and is* NP *complete.*

While section 4 discusses algorithmic aspects in greater detail, the reader should at least by now have an intuitive grasp of the practical implications of matching complexity. Imagine that we have selected an initial tentative mapping $\{c_i\} \to \{z_i\}$ from specified to mechanical cylinders. We then derive mechanical key representations according to Theorem 7 and subsequently use Theorem 8 to check correctness. In the unfortunate yet likely case that one or more of the derived mechanical keys open too many cylinders, we need to retract, modify the mapping, and start over.

Finally, take careful note of how the encoding problem seeks an optimal compact abstraction of the lock system specification disregarding mechanical constraints, while, on the other hand, the matching problem embodies the core issue of representing the requirements of the specification within the limitations imposed by the mechanics.

4. Algorithms.

This section discusses algorithms for encoding and matching problems. Some details are considered sensitive by our industrial partner and are only elaborated upon sparingly; we ask the reader's forgiveness.

4.1. Algorithms for the Encoding Problem.

4.1.1. Hierarchical Encoding and Obstacles for Optimality. For the sake of simplification, let us first consider a lock system whose DAG, after transitive reduction, is a tree. Starting from the top downwards, the children of the root node *partition* the set of cylinders into a disjoint union of subsets [9]; see the example depicted in Figure 4.1 where the cylinders consist of five mutually disjoint subsets induced by the must open/must not open relations of the children of the root (master key).

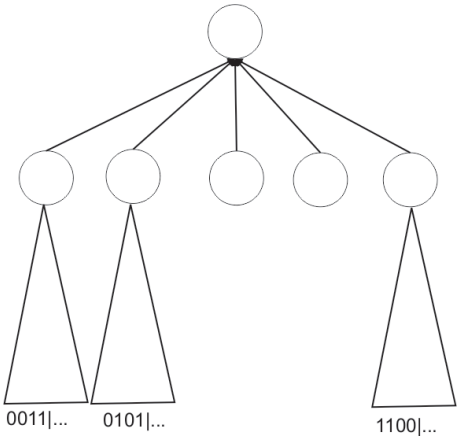

Fig. 4.1 *Hierarchical encoding of disjoint subtrees. All cylinders and keys in one subtree share a unique subencoding (shown at the bottom of each subtree; in this case a prefix of four) that separates them from all cylinders and keys in other subtrees.*

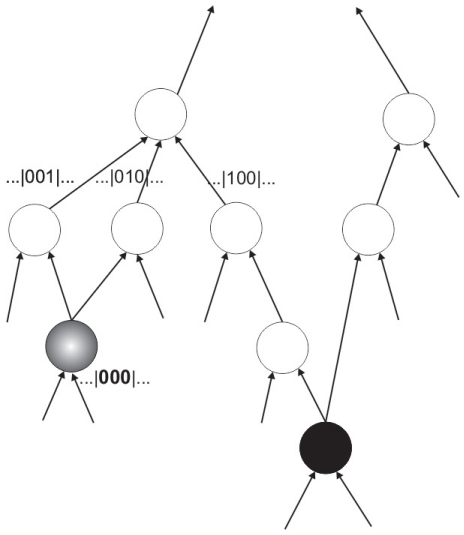

Fig. 4.2 *Unstructured DAG with cross-locking (shaded and filled nodes).*

Then assigning all cylinders (and keys) in one subtree a unique common "subencoding" (in this example a prefix of four) ensures that no key from another subtree will open them, thereby enforcing a *maximum separation* between subtrees. (Note that just like in Table 3.3 from section 3, we use binomial coefficients to achieve compact encodings at this stage.) This intuitive idea can of course be applied recursively, by proceeding to the next "level" of the children's children, which *refines* the previous *coarser* partition of cylinders into a *finer* one [9, 21, 51]. The final encoding then is composed of a sequence of several subvectors, each reflecting another level of recursive partitioning. In this simple form, hierarchical encoding thus is an instance of the *divide and conquer* paradigm using *breadth-first search (BFS)* for the graph traversal [17, 36].

However, our enthusiasm for the maximum separation principle is quickly dampened once we recognize its limitations as only a *locally optimal* heuristic. In particular, the cylinder sets in a partition may vary greatly in cardinality, as may the keys from one subtree to another. Such *imbalances*, when they occur, cannot be avoided as they are a direct consequence of the lock system specification (however, see section 4.1.2 for an at least partially mitigating technique based on introducing artificial nodes via *splitting*). Enforcing strictly hierarchical BFS processing thus implies foregoing optimality; the largest subgroup of nodes encountered at that recursion level determines the space needed for a subencoding.

Further serious complications arise from cross-locked nodes; see the examples in Figure 4.2. Cross-locking joins what would otherwise be independent subtrees; this implies that the cylinder sets at the bottom of two or more subtrees overlap. Note that whenever present, a cross-locked node is explicitly prescribed in the lock system specification. Therefore, all its parents with the corresponding must open relations are of essential importance; it is not correct to further reduce the problem to, say, a spanning tree [17, 37].

One of the complications that cross-locking causes for hierarchical encoding is illustrated by the shaded node in the left lower corner of Figure 4.2. The hierarchical separation at the parent level, separating ("001") and ("010") from their sibling ("100"), is overturned by the imperative ("000") = ("001") ∧ ("010") of the cross-locked node. To guarantee the must not open relation of the sibling, one thus needs *another* feature on the cylinders below the cross-locked key. Depending on the separation at coarser levels, it may still be possible to use the same feature location for separation in different independent subtrees, thereby mitigating the risk of having to introduce a new feature for each cross-locked node. This more frugal approach, however, comes at the price of significant overhead in book-keeping.

Second, there are cross-lockings like the solid filled black node in the lower right of Figure 4.2 which do not reside at one unique recurrence level. These nodes require special attention as they threaten the *topological order* in which parents are processed before children. With considerable additional care, one may elect to delay their processing until their very last occurrence. Alternatively, one may identify such nodes beforehand and assign the concerned cylinders their own individual feature, presupposing that there are just a few of these nodes requiring such special treatment.

At this point in our discussion, we hope to have convinced the reader of the severe obstacles standing in the way of finding an optimal encoding for a general DAG. In section 4.1.2, we approach the subject from another angle originating from object-oriented programming. Section 5.4.1 will present an illustrative case study exhibiting the sophisticated encoding of a practically relevant test case.

4.1.2. The Graph Coloring Connection. This section studies automatic encoding techniques that have been derived in the conceptually related setting of multiple-inheritance type hierarchies in object-oriented programming. Moreover, it provides some insight into the encoding complexity as characterized in Theorem 6 from section 3.4. Note that our focus is on algorithms for general DAGs; we do not consider techniques that are limited to trees [14, 29].

Our presentation adopts the algorithmic framework for hierarchical encodings suggested in [12] that encapsulates a sequence of related works [5, 11, 12, 38]. Two of these algorithms, a hierarchical encoding based on Caseau et al. [11, 12] and another one by Krall et al. [38], will feature in the comparative studies presented in section 5.

All these techniques are based on the fundamental observation that the number of so-called *join-irreducible* elements in the semilattice establishes an upper bound for minimum encoding length [44, 45]. (A join-irreducible element j has the property that for any nonempty subset S with $j = \bigvee S$ implies $j \in S$.) The join-irreducibles feature as vertices of an associated *conflict graph* [12, 38] whose (undirected) edges indicate which of the join-irreducibles necessarily need to be separated. Finally, a vertex *coloring* determines the *chromatic number* (minimum number of colors) or an approximation thereof, where the colors can be directly translated into the bits of a correct encoding of the whole upper semilattice. It is this connection to the *graph coloring* problem (which is known to be NP complete [24]) that makes the encoding problem NP hard; see Theorem 6.

Nodes with many children represent an intrinsic challenge as all these children may appear as a *clique* (a completely connected subgraph) in the associated conflict graph. Heuristically *splitting* the children into subgroups by inserting artificial intermediate nodes [12, 29, 38] provides a remedy that decreases branching factor and clique sizes; it therefore is a default preprocessor [12, 38] in all these algorithms. (As

an alternative, one may also consider multicoloring the vertices of the conflict graph; however, in the absence of an efficient multicoloring algorithm and with reported performance [69] comparable to Krall, Vitek, and Horspool [38], we did not pursue this idea any further.)

4.2. Algorithms for the Matching Problem.

4.2.1. Navigating the Landscape of Combinatorial Optimization Algorithms. Combinatorial optimization is a well-established and active area of research with a host of specialized subdomains too vast to discuss; instead, we refer the reader to textbooks [10, 37, 48, 63] that also contain further references. What we do want to provide here is our rationale for choosing a particular algorithm among the many available techniques. For the following discussion, it is important to remember that we search for a *globally optimal* solution of the matching problem; an inexact *approximate* solution is ultimately of no practical value (even though one still may make some use of it, for example, as a starting guess to initialize or restart an algorithm). Furthermore, a single instance, a "point," in the search space refers to the entire key DAG induced over a candidate set of cylinders whose correct matching has to be assessed.

Two useful (for us) classification categories govern the distinction between (1) *deterministic* algorithms and those that rely on *randomness* as well as (2) algorithms with *memory* and essentially *memoryless* ones.

A deterministic algorithm with memory [61, 62, 67, 68] *systematically searches* for a matching while keeping track of already explored or nonpromising candidates. A memoryless algorithm tries to exploit patterns and properties of the search space or the error (objective) function to get by (essentially) without memory; the best-known example is the *greedy algorithm* that is optimal for matroids [17, 37, 52].

As a powerful alternative to determinism, randomized search heuristics [8, 48] allow a nonsystematic exploration of the global search space as well as a convenient way of escaping from local optima that are not globally optimal [48, 57]. *Simulated annealing* (SA) [1, 2, 10, 13, 20, 35, 58] is an essentially memoryless approach that seeks an optimum of the objective function without insisting on continuously improving solutions. Instead, degradations are accepted albeit of smaller degree and with decreasing probability over time, governed by a schedule borrowed from a physical model for the cooling of matter. An example of a randomized algorithm with memory is *tabu search* [27] that maintains a list of recently inspected suboptimal solutions in order to avoid revisiting them in the short term. *Genetic algorithms* and *genetic programming* [10, 47, 48] maintain a set of promising approximate solutions from which new ones are generated mimicking the biological principles of recombination, mutation, and selection of the fittest.

Given the complexity of the matching problem, it comes as no surprise that experiments with state-of-the-art graph algorithms such as [15, 16, 61, 62, 65] quickly expose the intractability of deterministic systematic search space exploration. With no theoretical or practical evidence for the usefulness of greedy algorithms, randomized heuristics thus are a natural choice. Deciding among the many available techniques, however, is not straightforward. One of our guiding principles was a small memory footprint in order to allow multiple potential use cases for our software, including stand-alone use cases, seamless-workflow-integrated use cases, and as back-end application for web services. This line of reasoning speaks against genetic algorithms and instead advocates for SA as our main vehicle. Additionally, the final decision in its favor was supported by its success in a related previous investigation [41] and our own pilot feasibility study. We do not believe SA to be the only reasonable choice of

an algorithm [10, 57, 72]. On the contrary, experiments [10] suggest that its success depends as much on problem-specific tailoring as on the (meta-)heuristic.

4.2.2. Simulated Annealing and Refinements. The purpose of SA [1, 2, 10, 13, 20, 35, 48, 58] for our matching problem consists of finding a zero of an appropriate nonnegative *error function* f measuring the grade of deviation from a correct matching. Refining the idea of local search in a constraint satisfaction problem (CSP) [57], we successively improve the matching of cylinders such that all keys finally have their correct must open/must not open functions. We will say more about error function design later on; for now we can think of simply counting the incorrect edges in the induced key graph of a tentative matching. Such edges correspond to key representations that violate specified must not open relations; see section 3.4 and also [31].

Ideally, one would like to find a sequence of tentative matchings such that their associated errors strictly decrease, $f_1 > f_2 > \cdots$, thereby ensuring convergence to a *global* optimum. However, as explained in section 4.2.1, there is a very high chance of getting stuck in a *local* minimum from which the algorithm needs to be able to *escape* [48] in order to further explore the search space. SA provides a mechanism based on randomization for this purpose: let f_c denote the current value of the error function and $f_n > f_c$ a new value corresponding to a new yet "worse" tentative matching; the new matching is nonetheless accepted as the next iterate with probability $p = e^{\frac{f_c - f_n}{T}} < 1$, where $T > 0$ is a control parameter called the *temperature*. Observe that for fixed values f_c and f_n, p is strictly monotonically increasing in T. To achieve convergence, SA uses a *cooling schedule* that lowers T over time, thereby gradually accepting only smaller degradations, with decreasing probability. The theory of Markov chains and Monte Carlo methods offers a perspective on the probability of convergence to a global optimum; for details see [10, 55] and the references therein. In practice, the cooling schedule benefits from careful tuning of our application; see also [1, 10, 58].

The generation of new tentative candidate mappings is a key ingredient of SA. Plain SA merely considers random exchanges of mechanical cylinders, but this approach is often too inefficient due to the vastness of the cylinder sets. Instead, one can *guide* SA by restricting its choices at runtime to "reasonable" candidates from a promising *neighborhood* [1, 10]. In our case, candidates can be determined ahead of time by assigning the "0" positions from a precomputed Caseau [11, 12]-type encoding as filters to the mechanics. The idea of using precomputed models to guide dynamic choices was originally introduced for improved dynamic task scheduling in parallel computing [6], but it shows equally great potential in our SA framework; see section 5. Moreover, matching entire *groups* of cylinders instead of individual ones yields additional efficiency gains.

Another point concerns the error function design. Intuitively, it seems desirable to keep track of very poorly matched cylinders that make significant contributions to the error function, in order to prioritize and expedite improvements to their matching. Such *auxiliary objectives* [10], however, are very costly to evaluate and have a significant impact on runtime. While computationally cheaper *surrogate error functions* [10] can be formulated for this task, we did not find worthwhile benefits in practice and chose to keep using the less expressive yet simpler and cheaper error function outlined earlier.

5. Sample Test Results and Case Studies.

5.1. Evaluation Overview. Table 5.1 gives a summary of the test data. Our test set consists of two parts. The first eight cases constitute a cross-section of represen-

Table 5.1 *Evaluation dataset. Data was recorded on a workstation with AMD Phenom II X6 1055T processor 2.80GHz and 4.00 GB RAM. The leftmost columns denote the lock system reference and numbers of cylinders, keys, and cross-locked nodes. As an additional measure of complexity, the following column gives the encoding length from a hierarchical encoding based on Caseau [11, 12]; see section 4.1. The final column specifies the runtime of our own filtered SA (fSA) from section 4.2 that computes a mechanically feasible solution to both the encoding and the matching problems. Results are discussed in section 5.3.*

Lock system	# cylinders	# keys	# x-locks	Caseau	Runtime fSA
TS 001 298	110	175	54	28	30 sec.
TS 001 200	564	649	414	57	12.5 min.
TS 001 168	3114	3688	31	33	38 min.
TS 100 250	52	75	11	15	1 sec.
TS 100 264	181	204	27	23	1 sec.
TS 100 255	129	163	130	48	45 sec.
TS 100 298	58	91	48	31	20 sec.
TS 100 299	364	383	42	26	1 sec.
TS 100 300	38	57	22	22	< 10 sec.
TS 100 301	106	169	3	21	< 10 sec.
TS 100 302	100	105	0	14	1 sec.
TS 100 305	6	10	0	8	1 sec.
TS 100 306	128	156	16	22	< 10 sec.
TS 100 307	56	65	4	22	1 sec.
TS 100 308	165	199	0	18	1 sec.
TS 100 309	32	41	18	21	< 10 sec.
TS 100 310	37	52	27	25	< 10 sec.
TS 100 311	387	495	81	28	20 sec.
TS 100 312	15	27	5	9	1 sec.
TS 100 314	23	41	18	19	1 sec.
TS 100 315	48	58	36	26	1 sec.
TS 100 316	22	26	4	12	1 sec.

tative core problems that were selected for an initial pilot feasibility study. This data was later supplemented by a selection of problems collected during one working day at the KESO, ASSA ABLOY facility.

Comments on the dataset in Table 5.1 follow in section 5.2. A discussion of the results is given in section 5.3 and is supplemented by two in-depth case studies presented in section 5.4.

5.2. Remarks on the Dataset. Table 5.1 indicates that cross-locked nodes occur very frequently in practice; section 5.4.1 will investigate their encoding in one test case, TS 100250, in greater detail. Moreover, we observe that encoding length almost always exceeds the 15 feature places provided by a single cylinder set; section 5.4.2 shows how we match with multiple cylinder sets.

Because of the nondeterministic nature of our own algorithm, SA with encoding-based filters (fSA), its performance is subject to random variations; timing results in Table 5.1 report typical performance (reproducible within meaningful error margin) on the test data.

5.3. Discussion. Prompted by one referee's comments, this section provides a discussion of the results in Table 5.1 with the goal of making our algorithmic contributions more transparent.

Previously existing algorithms for the encoding problem stem from application domains other than our own. For example, the techniques by Caseau et al. [11, 12] and by Krall, Vitek, and Horspool [38] described in section 4.1.2 were developed

for compactly describing multiple-inheritance hierarchies in object-oriented programming. Thus, it should not surprise the reader to learn about their shortcomings for lock system calculations which exhibit additional restrictions imposed by mechanical feasibility.

First, "traditional" encoding assumes the availability of an entire set of binary vectors. However, only a subset of those vectors might be mechanically feasible and thus useful for the subsequent matching problem. Case Study 2 in section 5.4.2 highlights this aspect in detail, contrasting 124 million theoretically possible combinations with 620,000 mechanically feasible ones.

Second, previous encodings might be simply too long to be matched. With the mechanical design currently being limited to at most 37 places, an encoding of length 57 as in the case of TS001200 cannot be used.

A third and subtle issue concerns the use of features for encoding the cylinders at the bottom of the upper semilattice. In previous settings, encodings did not need to mind their number; for example, in the case of TS001200, we encountered a cylinder encoding with 43 features. However, as discussed in section 2.1, mechanical cylinders currently contain 7 to 9 pins, with a strict feasible maximum of 15.

None of these remarks, of course, mean to disparage the classical techniques. Indeed, one way to think of our algorithm is as a nondeterministic enhancement of existing deterministic approaches that takes account of the additional restrictions in the lock system setting. Using our prefiltered SA, we can compute within reasonable additional time a solution that does respect all the mechanical constraints.

Finally, it is worth noting that in contrast to most "conventional" applications of SA, our algorithm is required to find a *global* optimum of the error function in every case, representing a mechanically feasible solution to both the encoding and the matching problems. An only locally optimal solution of the matching problem, no matter how close to the global optimum, would correspond to an incomplete mechanical realization of the lock system specification chart and is therefore deemed unacceptable.

5.4. In-Depth Case Studies. To gain some additional understanding of the previous discussion, this section considers two test cases in greater depth. We first study a compact encoding of TS100250, shown in Figure 5.1. Then we take a closer look at a correct matching of TS100255 from Figure 5.2.

5.4.1. Case Study 1: Encoding of TS100250. Figure 5.3 shows an excerpt of an efficient, correct encoding of the TS100250 lock system, focusing on the cross-locked nodes and their ancestors. The length of the complete encoding is 15, with node 1 representing the master key.

In the light of the discussion in section 4.1.1 on the complications caused by cross-locking, it is remarkable that the lock system can be encoded with 15 bits, given that there are 11 cross-locked nodes among 75 nodes in total. This example impressively illustrates the sophistication in encoding solutions which an algorithm can produce when permitted by the lock system specification.

Recalling Theorems 1 and 3, we can verify in detail how the upper semilattice structure from the partial ordering of the keys is reflected in the encoding

$$\text{key } \mathbf{k1} \text{ opens all cylinders opened by key } \mathbf{k2} \quad \Leftrightarrow \quad \mathbf{e}(\mathbf{k1}) > \mathbf{e}(\mathbf{k2}). \tag{15}$$

The symbol F in Figure 5.3 denotes free positions that are redundant, since we may not always need the full length to encode all of a key's must open/must not open

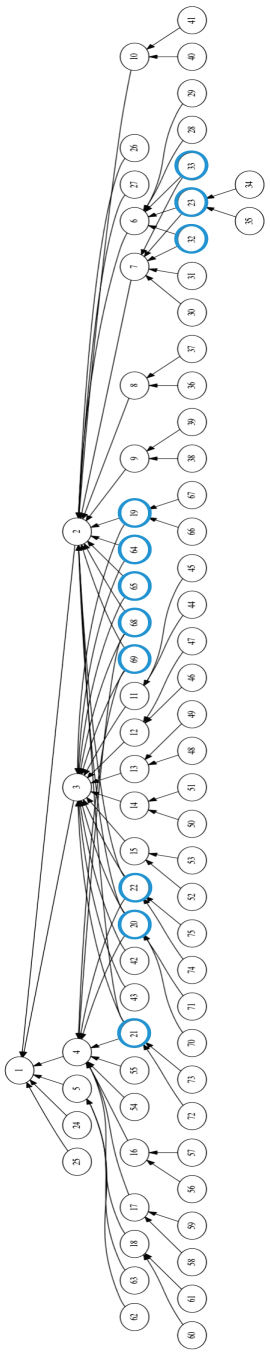

Fig. 5.1 *TS*100250: *Semistructured lock system with* 52 *cylinders and* 75 *keys,* 11 *of them cross-lockings. Cross-locked nodes have been highlighted by thickened borders.*

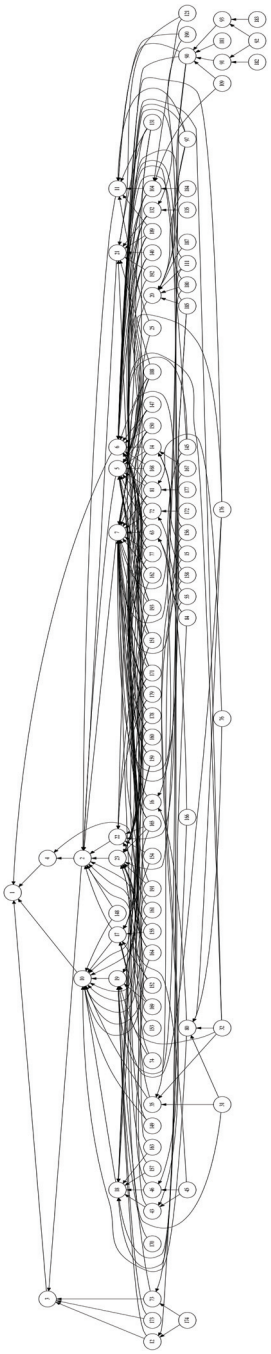

Fig. 5.2 *TS*100255: *Highly unstructured lock system with* 129 *cylinders and* 163 *keys,* 130 *of them cross-lockings. Singleton keys have been omitted to improve legibility.*

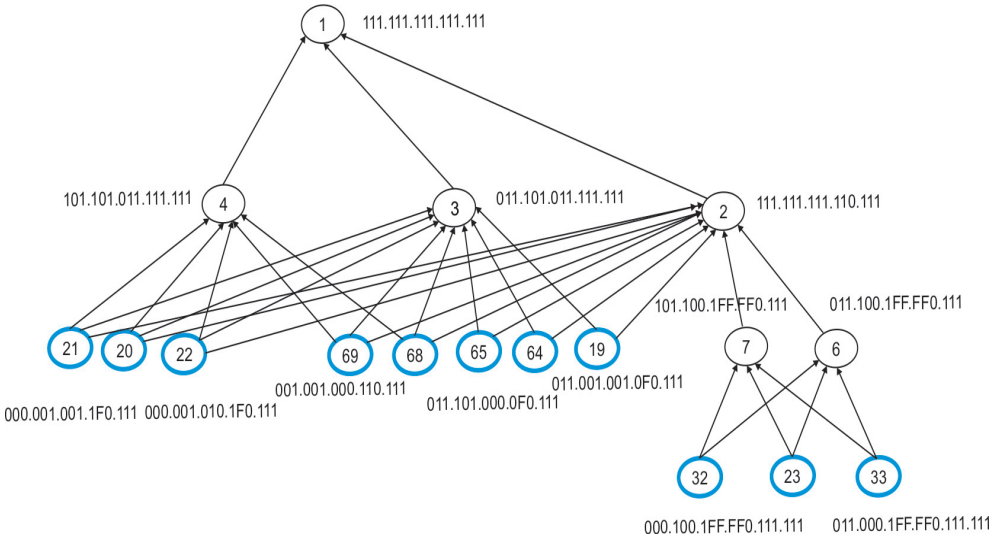

Fig. 5.3 *Encoding of TS*100250 *(excerpt; compare with Figure* 5.1*). Encoding length is* 15*. For better legibility, the encoding has been partitioned into parts of size three. "*1*" denotes presence of a feature, "*0*" its absence; "F" denotes free (undetermined) feature positions.*

relations. An F position may be chosen to be either "1" or "0" on the cylinders; both choices yield a correct encoding, as long as Theorem 2 holds.

Free positions are ultimately a consequence of imbalances in the DAG arising from the uneven distribution of cylinders and keys or, alternatively, from must open/must not open relations of varying complexity. It is instructive to study the effects of such imbalances on encoding. Note how two children of the root, 3 and 4, have fewer "1"-features than their sibling, 2. This implies that nodes 3 and 4 have smaller or less constrained ancestor DAGS than node 2. Moreover, with node 2 possessing just one feature less than node 1, its ancestors and their connecting edges critically determine the total encoding length.

Finally, note that any *permutation*, when consistently applied to all the vectors of a correct encoding, maintains the partial ordering of keys and cylinders and thereby preserves correctness. Thus, we must not be fazed by a computer algorithm that produces an encoding not akin to intuitive human interpretation. All that matters is correctness; we are free to rearrange the bit order a posteriori to a "friendlier" format.

5.4.2. Case Study 2: Matching of TS100255. Figure 5.4 shows an excerpt of a correct matching of the TS100255 lock system. Even though this lock system is not particularly large, it represents an extremely challenging example due to its lack of structure and its high number of cross-lockings, 130, out of 163 keys and 129 cylinders.

The complexity of TS100255 makes it impossible to find a correct matching within a single set of mechanical cylinders with just 15 feature places. Instead, as indicated at the end of section 2.1, we have to combine several sets of cylinders with varying feature positions to allow for a matching to exist. As in each cylinder set, its 15 representation bits correspond to their own unique feature positions, and we can embed several sets into longer vectors where each bit again represents a unique position that may occur in one set but not necessarily in others. In the example in Figure 5.4, we embedded multiple cylinder sets into a combined representation of length 37, long enough to

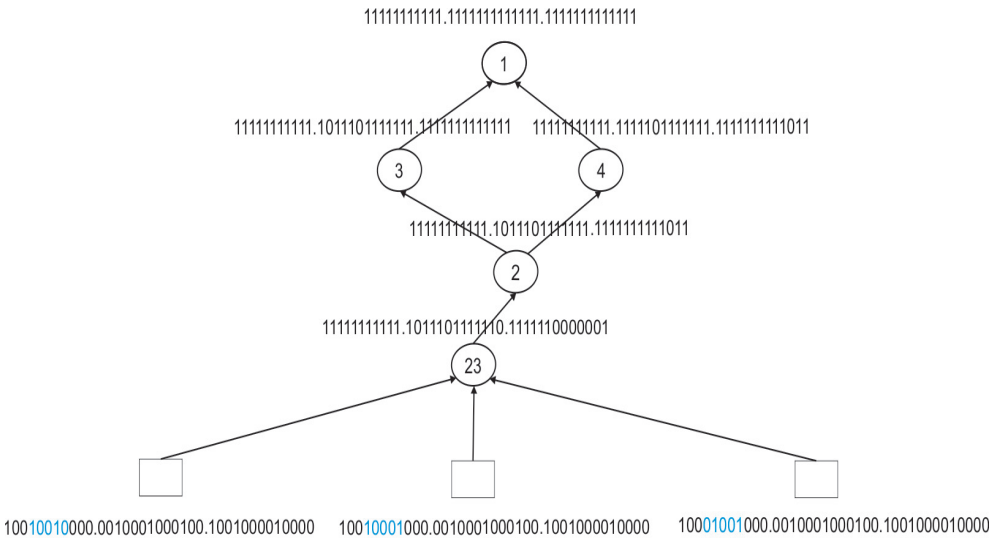

Fig. 5.4 *Matching of TS*100255 *(excerpt; compare with Figure* 5.2*). To clearly distinguish them, the leaves at the bottom of the graph corresponding to singleton keys (cylinders) are represented by rectangular boxes; each one has nine mechanical features. The representation of the union of cylinder sets has length* 37 *and is partitioned into three parts, corresponding to features on the top, left, and right sides of each key. (Just like in the encoding problem discussed in section* 5.4.1*, the bit order in the representation vectors can be changed with a permutation without jeopardizing the matching correctness.)*

find a correct matching by SA. (Interestingly, the encodings by both Caseau [11, 12] and Krall [38] are longer, although this is hardly a fair comparison, due to SA's much greater computational effort.) In this case, the union of cylinder sets contains about 620,000 mechanical cylinders among which a correct matching has to be found, guaranteeing that the mechanical key representations thus found satisfy the partial ordering equations

$$\text{key } \mathbf{k1} \text{ opens all cylinders opened by key } \mathbf{k2} \quad \Leftrightarrow \quad \mathbf{r}(\mathbf{k1}) > \mathbf{r}(\mathbf{k2}). \tag{16}$$

The upper semilattice of mechancial keys over 620,000 cylinders is giant and too large to ever store in memory; nevertheless, note how strongly constrained the cylinders are by mechanical restrictions as the theoretically possible number of $\binom{37}{9} \approx 124$ million combinations is far from being realized.

The intricacy of this example illustrates nicely the complexity that human professionals performing "manual" lock system calculations have to handle and the sophistication required to solve the lock system calculation problem. It thus makes a strong case for the importance and impact of the research and the computational techniques described in this paper.

5.5. A Note on Default Behavior and Exception Handling. When approaching an unknown lock system, we risk facing an all too familiar dilemma if the randomized matching algorithm does not find a solution quickly. Should we be patient and wait longer, or abort and start over? Without a universally valid answer, one typically chooses an a priori runtime limit, say, two hours, that is sufficiently long for handling almost all cases and short enough to still allow rush processing of an order [41].

If repeated runs of the black-box algorithm do not produce a solution, an *exception* has occurred that needs to be handled under the guidance of a human expert; on the algorithmic level this can be done, for example, by increasing the runtime limit or modifying default parameters and strategies. In the very rare case that a lock system still cannot be matched, the expert enters in a feedback dialogue with the customer to revise parts of the lock system specification.

6. Conclusion and Perspective. Approaching the end of this paper, we hope we have conveyed some of the mystery and magic of modern mechanical lock systems which offer a surprising diversity of interesting challenges not only to burglars and safecrackers, but also to mathematicians and computer scientists. If, by sheer curiosity or in the noble name of science, you the reader now feel inclined to tinker with locks and keys, we wish you a lot of fun—and the understanding and forgiveness of family and neighbors. In what follows, we briefly discuss interesting problems that are of a more theoretical or computational nature.

A combinatorial optimization problem of great practical importance concerns the development of algorithms for the *optimal extension* of a previously calculated lock system. We can distinguish three types of such extensions: first, the search for new keys with a novel set of opening functionalities for existing cylinders; second, the selection of new cylinders to be opened by existing keys; and third, a combination of the previous two cases. Using sort and search algorithms, one can quickly check if an extended solution exists that includes the previous one. Otherwise, the task becomes harder and we have to find a solution that requires the *smallest modification* of the previous calculation, that is, the exchange of as few of the existing cylinders and keys as possible.

A second issue concerns modifications to the mechanical design of the cylinders in order to allow for simplified models and more efficient computations. As pointed out in [41], automorphisms of the poset of mechanical keys can be exploited algorithmically by working with equivalence classes, thereby reducing the complexity of the calculations. Presently, as described in section 2, the pin assignments in the mechanical cylinders do not uniformly explore the combinatorial space and thereby subvert the existence of automorphisms. We expect a redesign to result in substantially better runtime performance and a simplified matching of groups of singleton keys to the same equivalence class of mechanical cylinders.

Moreover, connections to algebraic coding theory [56, 70, 71] may provide deep insights into intriguing mechanical design questions. For example, while in this paper we merely distinguish between presence and absence of security features, the use of pin shape variations within the same feature location is equivalent to encoding and matching with more a expressive alphabet than just a binary one, thereby producing cylinder sets of greater cardinality. But how much larger can we expect these sets to become, depending on how much we want individual cylinders to differ from each other? Instructively, when phrased in terms of a Hamming distance between code words, algebraic coding theory relates this particular question to so-called singleton bounds and maximum distance separable codes [64].

Last but not least, one may try to extend alternative models of knowledge representation and reasoning from artificial intelligence to lock system calculations. It would be very desirable to create an automated expert system [25, 32, 57] that implicitly encodes human insights and experience gained from decades of "manual" calculations. The computational challenges aside, one practical difficulty in that regard, however, is the transmission of knowledge from the experts who, for somewhat

understandable reasons, might only reluctantly share their skills. Nonetheless, we do see promising opportunities for researchers interested in such collaborations and hope to have sparked some interest in this exciting industrial application.

Appendix A. Educational Activities for Students. One referee's comments suggested that it would be helpful to outline educational activities for students in order to make the topics covered here more engaging and relatable. In this appendix, we provide a few such pointers.

A.1. A Seminar on Mathematical Consulting. The research described in this paper sprang from the very general and unfocused idea of creating fruitful and mutually beneficial partnerships between industry and academia. Before actually working on the science part, we had to identify an interesting—and interested—industrial partner, define a research topic that offered challenges as well as potentially significant pay-offs, and come up with a strategic plan that was promising enough to be successful in terms of project goals and funding. As a topic for a seminar, we propose that students share this experience and try out the role of a mathematical consultant for industry.

As a first step, research local companies and think about the products and services they provide. Imagine challenges that such a company could face, and how to phrase these questions in scientific terms. As a second step, try to make contact, for example at a fair, or by inviting a representative to a meeting or to give a talk. (If there are not enough suitable companies, one may have to resort to other academic departments within the university.)

Validate your ideas. Listen carefully, and pay attention not only to the topic of the conversation but also to the vocabulary used, to the line of thought, and to the metaphors and mental models employed. Try to translate into your own language, to create abstractions, and to relate to familiar scientific concepts or ideas. Ask many questions. Finally, identify potential contributions you could make and suggest ideas for a collaboration. Repeat.

Going through this process will equip you with a sense of the challenges of interdisciplinary work. It is good preparation for future employment outside of academia, and is also quite similar in spirit to graduate academic work such as preparing or evaluating a research proposal. On yet another level, being consciously exposed to a (sub-)culture potentially very different from one's own can be an immensely enriching experience and provide broader perspective, motivation, and insight. For a colorful account of such impressions by an artist-engineer, see [3].

A.2. A Group Activity on Analysis of a Local Lock System. The omnipresence of keys presents a great opportunity for hands-on experience in data modeling, analysis, and prediction. We therefore propose that students, for example, by means of a group project, create and validate their own model of a selected lock system.

As a preliminary step, select a suitable system and make sure to get legal approval before commencing the actual research. For example, a safe choice could be the lock system in a set of discarded office furniture.

First, create a tentative lock system specification chart that models what you think are the requirements of the system. Second, collect different keys and compare their features to discover similarities and discrepancies, then map these observations to the requirements identified in the lock system model. Try to assign roles to key features. Finally, try to predict the features of unknown keys by thei role in the lock system. Validate these predictions and refine as necessary.

Furthermore, depending on the number of available objects to study, one could also supplement the activity by trying out algorithms from statistical machine learning [33].

A.3. An Experimental Comparison of Deterministic and Randomized Algorithms. The graph subgraph isomorphism problem offers an ideal framework for the applied mathematician to gain first-hand experience with computational complexity and the issue of intractability.

We propose that students first study a deterministic, systematic algorithm for finding a graph subgraph isomorphism and observe how quickly the runtime becomes untenable with only slightly increasing problem size. Such algorithms are freely available, for example, as part of the Boost library [61, 62].

As a second task, students can customize a randomized algorithm of their own choice, or pick an existing one from the literature, for the graph subgraph isomorphism problem and try it out on the graph examples from the first part of the study. Depending on the test case and the algorithm, randomization may prove to be surprisingly efficient.

However, as one could see in the study presented in this paper, a randomized heuristic most likely needs problem-specific tailoring to make "informed" [57] choices and thus be genuinely useful. Students can challenge themselves and compete with each other to come up with the most effective original approach. Finally, the activity can be extended to other NP hard problems such as those that may originate from the consulting activity suggested above.

Acknowledgments. The illustrations in sections 1 and 2 appear by courtesy of KESO, ASSA ABLOY (Switzerland) Ltd.

We are grateful for the anonymous referees' thoughtful criticism that helped improve the paper. Thanks a lot also to Hans Kaper for his encouragement and support, and to the editors Louis Rossi and Ricardo Cortez for helping us find a home for this paper in the Education section of *SIAM Review*.

This paper is respectfully dedicated to the memory of our colleague Peter Biller.

REFERENCES

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, Wiley, New York, 1988. (Cited on pp. 397, 410, 411)

[2] E. Aarts and J. K. Lenstra, *Local Search in Combinatorial Optimization*, Princeton University Press, Princeton, NJ, 2003. (Cited on pp. 396, 397, 410, 411)

[3] J. L. Adams, *Good Products, Bad Products*, McGraw-Hill, New York, 2012. (Cited on p. 418)

[4] A. V. Aho, M. R. Garey, and J. D. Ullman, *The transitive reduction of a directed graph*, SIAM J. Comput., 1 (1972), pp. 131–137, https://doi.org/10.1137/0201008. (Cited on p. 404)

[5] H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr, *Efficient implementation of lattice operations*, ACM Trans. Program. Lang. Syst., 11 (1989), pp. 115–146. (Cited on p. 409)

[6] P. R. Amestoy, I. S. Duff, and C. Vömel, *Task scheduling in an asynchronous distributed memory multifrontal solver*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 544–565, https://doi.org/10.1137/S0895479802419877. (Cited on p. 411)

[7] R. J. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley, New York, 2008. (Cited on p. 401)

[8] A. Auger and B. Doerr, *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, World Scientific, River Edge, NJ, 2011. (Cited on p. 410)

[9] G. Birkhoff, *Lattice Theory*, revised ed., AMS, New York, 1948. (Cited on pp. 404, 407, 408)

[10] E. K. Burke and G. Kendall, *Search Methodologies*, 2nd ed., Springer, New York, 2014. (Cited on pp. 396, 397, 410, 411)

[11] Y. Caseau, *Efficient handling of multiple inheritance hierarchies*, SIGPLAN Not., 28 (1993), pp. 271–287. (Cited on pp. 409, 411, 412, 416)
[12] Y. Caseau, M. Habib, L. Nourine, and O. Raynaud, *Encoding of multiple inheritance hierarchies and partial orders*, Comput. Intell., 15 (1999), pp. 50–62. (Cited on pp. 396, 397, 406, 409, 411, 412, 416)
[13] V. Černý, *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*, J. Optim. Theory Appl., 45 (1985), pp. 41–51. (Cited on pp. 397, 410, 411)
[14] P. Colomb, O. Raynaud, and E. Thierry, *Generalized polychotomic encoding: A very short bit-vector encoding of tree hierarchies*, in Modelling, Computation and Optimization in Information Systems and Management Sciences, Comm. Comput. Inf. Sci. 14, Springer, New York, 2008, pp. 77–86. (Cited on p. 409)
[15] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, *An improved algorithm for matching large graphs*, in Proceedings of the 3rd IAPR-TC15 Workshop on Graph-Based Representations in Pattern Recognition, Cuen, 2001, pp. 149–159. (Cited on pp. 396, 410)
[16] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, *A (sub)graph isomorphism algorithm for matching large graphs*, IEEE Trans. Patt. Anal. Mach. Int., 26 (2004), pp. 1367–1372. (Cited on pp. 396, 410)
[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, Cambridge, MA, 2009. (Cited on pp. 404, 405, 406, 408, 410)
[18] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., Wiley, New York, 2006. (Cited on p. 406)
[19] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, 2nd ed., Cambridge University Press, Cambridge, UK, 2002. (Cited on p. 404)
[20] A. Dekkers and E. Aarts, *Global optimization and simulated annealing*, Math. Program., 50 (1991), pp. 367–393. (Cited on pp. 397, 410, 411)
[21] D. Ellerman, *The logic of partitions: Introduction to the dual of the logic of subsets*, Rev. Symb. Log., 3 (2010), pp. 287–350. (Cited on p. 408)
[22] L. Fennelly, *Effective Physical Security*, 4th ed., Butterworth-Heinemann, London, 2012. (Cited on p. 401)
[23] N. Ferguson, B. Schneier, and T. Kohno, *Cryptography Engineering: Design Principles and Practical Applications*, Wiley, New York, 2010. (Cited on p. 401)
[24] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979. (Cited on pp. 396, 405, 407, 409)
[25] J. C. Giarratano and G. D. Riley, *Expert Systems: Principles and Programming*, 4th ed., Thomson Course Technology, 2005. (Cited on p. 417)
[26] B. Glover and H. Bhatt, *RFID Essentials*, O'Reilly Media, Sebastopol, CA, 2006. (Cited on p. 400)
[27] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic, Norwell, MA, 1997. (Cited on p. 410)
[28] M. Habib and L. Nourine, *Bit-vector encoding for partially ordered sets*, in Orders, Algorithms, and Applications, Lecture Notes in Comput. Sci. 831, Springer, New York, 1994, pp. 1–12. (Cited on pp. 396, 397, 406)
[29] M. Habib, L. Nourine, O. Raynaud, and E. Thierry, *Computational aspects of the 2-dimension of partially ordered sets*, Theoret. Comput. Sci., 312 (2004), pp. 401–431. (Cited on pp. 396, 397, 409)
[30] D. Henning and C. Reynolds, *Houdini: His Legend and His Magic*, Warner Books, New York, 1977. (Cited on p. 394)
[31] L. Hérault, R. Horaud, F. Veillon, and J.-J. Niez, *Symbolic image matching by simulated annealing*, in Proceedings of the 4th British Machine Vision Conference (BMVC '90), British Machine Vision Association (BMVA), 1990, pp. 319–324. (Cited on p. 411)
[32] P. Jackson, *Introduction To Expert Systems*, 3rd ed., Addison-Wesley, Reading, MA, 1998. (Cited on p. 417)
[33] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, Springer, New York, 2013. (Cited on p. 419)
[34] L. Jones and T. de Castella, *Is the traditional metal key becoming obsolete?* BBC News Mag., October 30, 2014, bbc.com/news/magazine-29817520. (Cited on p. 394)
[35] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science, 220 (1983), pp. 671–680. (Cited on pp. 397, 410, 411)
[36] J. Kleinberg and E. Tardos, *Algorithm Design*, 1st international ed., Pearson, London, 2013. (Cited on p. 408)
[37] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 5th ed., Springer, New York, 2012. (Cited on pp. 397, 402, 405, 408, 410)

[38] A. Krall, J. Vitek, and R. Horspool, *Near optimal hierarchical encoding of types*, in ECOOP'97—Object-Oriented Programming, Lecture Notes in Comput. Sci. 1241, Springer, New York, 1997, pp. 128–145. (Cited on pp. 396, 397, 409, 410, 412, 416)
[39] M. Kuramochi and G. Karypis, *Frequent subgraph discovery*, in Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM '01), IEEE Computer Society Press, 2001, pp. 313–320. (Cited on p. 396)
[40] S. Kurutz, *Losing the key*, The New York Times, June 11, 2014. (Cited on p. 394)
[41] A. Lawer, *Calculation of Lock Systems*, Master's degree project, trita-na-e04081, Royal Institute of Technology, Stockholm, Sweden, 2004. (Cited on pp. 399, 410, 416, 417)
[42] R. Leighton and R. P. Feynman, *Surely You're Joking, Mr. Feynman!*, Vintage, New York, 1992. (Cited on p. 394)
[43] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, New York, 2003. (Cited on p. 406)
[44] G. Markowsky, *The representation of posets and lattices by sets*, Algebra Universalis, 11 (1980), pp. 173–192. (Cited on p. 409)
[45] G. Markowsky, *An overview of the poset of irreducibles*, in Combinatorial and Computational Mathematics: Present and Future, World Scientific, River Edge, NJ, 2001, pp. 162–177. (Cited on p. 409)
[46] M. McCloud, G. de Santos, and M. Jugurdzija, *Visual Guide to Lock Picking*, 3rd ed., Standard Publications, Champaign, IL, 2007. (Cited on p. 394)
[47] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer, New York, 2008. (Cited on p. 410)
[48] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, 2nd ed., Springer, New York, 2004. (Cited on pp. 396, 397, 410, 411)
[49] C. Moore and S. Mertens, *The Nature of Computation*, Oxford University Press, Oxford, 2011. (Cited on p. 405)
[50] D. Ollam, *Practical Lock Picking, Second Edition: A Physical Penetration Tester's Training Guide*, 2nd ed., Syngress, Maryland Heights, MO, 2012. (Cited on p. 394)
[51] R. Paige and R. E. Tarjan, *Three partition refinement algorithms*, SIAM J. Comput., 16 (1987), pp. 973–989, https://doi.org/10.1137/0216062. (Cited on p. 408)
[52] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover, New York, 1998. (Cited on pp. 396, 410)
[53] B. Phillips, *The Complete Book of Locks and Locksmithing*, 6th ed., McGraw-Hill Professional, New York, 2005. (Cited on pp. 394, 401)
[54] O. Raynaud and E. Thierry, *The complexity of embedding orders into small products of chains*, Order, 27 (2010), pp. 365–381. (Cited on pp. 396, 397)
[55] C. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed., Springer, New York, 2010. (Cited on p. 411)
[56] S. Roman, *Coding and Information Theory*, Springer, New York, 1992. (Cited on pp. 406, 417)
[57] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., Pearson, London, 2010. (Cited on pp. 396, 397, 410, 411, 417, 419)
[58] P. Salamon, P. Sibani, and R. Frost, *Facts, Conjectures, and Improvements for Simulated Annealing*, SIAM, Philadelphia, 2002. (Cited on pp. 410, 411)
[59] J. J. Schneider and S. Kirkpatrick, *Stochastic Optimization*, Springer, New York, 2006. (Cited on p. 397)
[60] B. Schröder, *Ordered Sets*, Birkhäuser, Cambridge, MA, 2003. (Cited on p. 404)
[61] J. G. Siek, *An Implementation of Graph Isomorphism Testing*, 2001; available at boost.org/doc. (Cited on pp. 396, 410, 419)
[62] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library. User Guide and Reference Manual. (C++ in Depth)*, Addison-Wesley Longman, Amsterdam, 2002. (Cited on pp. 396, 410, 419)
[63] D. Simon, *Evolutionary Optimization Algorithms*, Wiley, New York, 2013. (Cited on pp. 396, 397, 410)
[64] R. Singleton, *Maximum distance q-nary codes*, IEEE Trans. Information Theory, 10 (1964), pp. 116–118. (Cited on p. 417)
[65] S. S. Skiena, *The Algorithm Design Manual*, 2nd ed., Springer, New York, 2008. (Cited on pp. 396, 404, 410)
[66] C. Swedberg, *ASSA ABLOY creates NFC solution that uses phones to open doors, grant computer access*, RFID J., September 12, 2012, https://www.rfidjournal.com/articles/view?9899. (Cited on p. 400)
[67] J. R. Ullmann, *An algorithm for subgraph isomorphism*, J. ACM, 23 (1976), pp. 31–42. (Cited on pp. 396, 410)

[68] J. R. Ullmann, *Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism*, ACM J. Exp. Algorithmics, 15 (2010), article 1.6. (Cited on pp. 396, 410)
[69] M. van Bommel and P. Wang, *Hierarchy encoding with multiple genes*, in Database and Expert Systems Applications, Lecture Notes in Comput. Sci. 5181, Springer, New York, 2008, pp. 761–769. (Cited on p. 410)
[70] J. H. van Lint, *Introduction to Coding Theory*, 3rd ed., Springer, New York, 1998. (Cited on p. 417)
[71] L. R. Vermani, *Elements of Algebraic Coding Theory*, Chapman Hall/CRC, Boca Raton, FL, 1996. (Cited on p. 417)
[72] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, IEEE Trans. Evol. Comput., 1 (1997), pp. 67–82. (Cited on pp. 397, 411)