17. Final remarks

In these final, partly retrospective remarks, which are intended neither as a summary nor as a conclusion, we shall highlight and comment some overlapping facets of what has been achieved for the pattern-based solution of the general finite Constraint Satisfaction Problem (with a few open questions). As for the practical applicability of the approach developed in this book, we merely refer to the many Sudoku examples and to the chapters dedicated to other logic puzzles.

17.1. About our approach to the finite CSP

17.1.1. About the general distinctive features of our approach

There are five main inter-related reasons why this book diverges radically from the current literature on the finite CSP[1]:

– almost everything in our approach, in particular all our definitions and theorems, is formulated in terms of *mathematical logic*, independently of any algorithmic implementation; (apart from the obvious logical re-formulation of a CSP, the current literature on CSPs is mainly about algorithms for solving them and comparisons of such algorithms); however, by effectively implementing them and applying them to various types of constraints, we have shown that these logical definitions are not mere abstractions and that they can be made fully operational;

– we systematically use redundant (but not overly redundant) sets of CSP variables; correlatively, we do not define labels as <variable, value> pairs but as equivalence classes of such pairs;

– we fix the main parameter defining the "size" of a CSP and we are not (or not directly) concerned with the usual theoretical perspectives of complexity, such as NP-completeness of a CSP with respect to its size;

– we nevertheless tackle questions of complexity, in terms of the statistical distribution of the *minimal instances* of a fixed size CSP; although all our resolution rules are valid for all the instances of a CSP, without any kind of restriction, we grant minimal instances a major role in all our statistical analyses and classification

[1] We are not suggesting that our approach is better than the usual ones; we are aware that our purposes are non-standard and they may be irrelevant when speed of resolution is the main criterion; this is why we have stated our motivations with some detail in the Foreword.

results; the thin layer of instances they define in the whole forest of possible instances (see chapter 6 for this view) allows to discard secondary problems that multi-solution or over-constrained instances would raise for statistics; (by contrast, the notion of minimality is almost unknown in the CSP world);

– last but not least, our *purposes* lie much beyond the usual ones of finding a solution or defining the fastest algorithms for this. Here, *instead of the solution as a result, we are interested in the solution as a proof of the result, i.e. in the resolution path*. Accordingly, we have concentrated on finding *no-guessing, constructive, pure logic, pattern-based, rule-based, understandable, meaningful* resolution paths – though these words did not have a clear pre-assigned meaning.

We have taken this purpose into account in Part I by interpreting the "pure logic" requirement literally – i.e. as a solution completely defined in terms of mathematical logic (with no reference to any algorithmic notions). Thus, we have introduced a general resolution paradigm based on progressive candidate elimination. This amounts to progressive domain restriction, a classical idea in the CSP community. But, in our approach, each of these eliminations is justified by a single pattern – more precisely by a well defined *resolution rule* of a given *resolution theory* – and is interpreted in modal (non algorithmic) terms. We have established a clear logical (intuitionistic) status for the notion of a candidate (a notion that does not *a priori* pertain to the CSP Theory). Moreover, we have shown that the modal operator that naturally appears when one tries to provide a formal definition of a candidate can be "forgotten" when we state resolution rules, provided that we work with intuitionistic (or constructive) instead of classical logic (which is not a restriction in practice).

Once this logical framework is set, a more precise purpose can be examined, not completely independent from the vague "understandable" and "meaningful" original ones: one may want the *simplest* pure logic (or "rule-based" or "pattern-based") solution. As is generally understood without saying when one speaks of the simplest solution to a mathematical problem, we mean neither easiest to discover for a human being nor computationally cheapest, but simplest to understand for the reader. Even with such precisions, we have shown that "simplest" may still have many different, all logically grounded, meanings, associated with different (purely logical) ratings of instances.

Taking for granted that hard minimal instances of most fixed size CSPs cannot be solved by elementary rules but they require some kind of chain rules (with the classical xy-chains of Sudoku as our initial inspiration), we have refined our general paradigm by defining families of resolution rules of increasing logical (and computational) complexity, valid for any CSP: some reversible (Bivalue-Chains, g-Bivalue-Chains, Reversible-Subset-Chains, Reversible-g-Subset-Chains) and some orientated, much more powerful ones (whips, g-whips, S_p-whips, gS_p-whips, W_p-whips and similar braid families).

The different resolution paths obtained with each of these families when the simplest-first strategy is adopted correspond to different legitimate meanings of "simplest solution" (when they lead to a solution) and, in spite of strong subsumption relationships, we have shown (in several chapters, by exemples of instances that have different ratings) that none of them can be completely reduced to another in a way that would preserve the ratings. Said otherwise: there does not seem to be any universal notion of (logical) simplicity for the resolution of a CSP.

17.1.2. About our resolution rules (whips, braids, …)

Regarding these new families of chain rules, now reversing the history of our theoretical developments, four main points should be recalled:

– We have introduced a formal definition of Trial-and-Error (T&E), a procedure that, in noticeable contrast with the well known structured search algorithms (breadth-first, depth-first, …) and with all their CSP specific variants implementing some form of constraint propagation (arc-consistency, path-consistency, MAC, …), allows no "guessing", in the sense that it accepts no solution found by sheer chance during the search process: a value for a CSP variable is accepted only if all its other possible values have been tested and each of them has been constructively proven to lead to a contradiction.

– With the "T&E vs braids" theorem and its "T&E(T) vs T-braids" extensions to various resolution theories T, we have proven that a solution obtained by the T&E(T) procedure can always be replaced by a "pure logic" solution based on T-braids, i.e. on sequential patterns with no OR-branching accepting simpler patterns taken from the rules in T as their building blocks.

– Because its importance could not be over-estimated, we have proven in great detail that all our generalised braid resolution theories (braids, g-braids, S_p-braids, gS_p-braids, B_p-braids, B*-braids, …) have the *confluence property*. Thanks to this property, we have justified the idea that these types of logical theories can be supplemented by a "simplest first" strategy, defined by assigning in a natural way a different priority to each of their rules. When one tries to compute the rating of an instance and to find the simplest, pure logic solution for it, in the sense that it has a resolution path with the shortest possible braids in the family (which the T&E procedure alone is unable to provide), this strategy allows to consider only one resolution path; without this property, all of them should *a priori* be examined, which would add an exponential factor to computational complexity[2]. Even if the

[2] The confluence property of a resolution theory T should not be interpreted beyond what it means. In particular, it does not allow to assign a rating to each candidate of an instance P: different resolution paths for P within T will always have the same rating of their hardest step,

goal of maximum simplicity is not retained, the property of stability for confluence of these T-braid resolution theories remains very useful in practice, because it guarantees that valid eliminations and assertions occasionally found by any other consistent opportunistic solving methods (or any application-specific heuristics or any other search strategy) cannot introduce any risk of missing a solution based on T-braids or of finding only ones with unnecessarily long braids.

– With the statistical results of chapter 6, we have also shown that, in spite of a major structural difference between whips and braids (the "continuity" condition), whips (even if restricted to the no-loop ones) are a very good approximation of braids[3], in the double sense that: 1) the associated W and B ratings are rarely different when the W rating is finite and 2) the same "simplest first" strategy, *a priori* justified for braids but not for whips, can be applied to whips, with the result that a good approximation of the W rating is obtained after considering only one resolution path (i.e. the concrete effects of non confluence of the whip resolution theories appear only rarely). This is the best situation one can desire for a restriction: it reduces structural (and computational) complexity but it entails little difference in classification results.[4] Of course, much work remains to be done to check whether this proximity of whips and braids is true for all the types of extended whips and braids defined in this book (it seems to be true for g-whips) and for CSPs other than Sudoku (it seems to be true also for Futoshiki, Kakuro, Map colouring, Numbrix® and Hidato®, as can be seen by the small number of occurrences of braids appearing in the resolution paths).

17.1.3. About human solving based on these rules

The four above-mentioned points have their correlates regarding a human trying to solve an instance of a CSP "manually" (or should we say "neuronally"?), as may be the "standard" situation for some CSPs, such as logic puzzles:

– It should first be noted that T&E is the most natural and universal resolution method for a human who is unaware of more complex possibilities and who does

but these hardest steps may correspond to the elimination of different candidates. This is not an abstract view; it happens very often.

[3] We have shown this in great detail for Sudoku, but the resolution paths we have obtained for most of the Futoshiki, Kakuro, Map colouring, Numbrix® and Hidato® examples confirm a similar behaviour.

[4] By contrast, the "reversibility" condition often imposed on chains in some Sudoku circles (never clearly formulated before *HLS*) is very restrictive and it leads some players to reject solutions based on non-reversible (or "orientated") chains (such as whips and braids) and to the (in our opinion, hopeless for hard instances) search for extremely complex patterns (such as all kinds of what we would call extended g-Fish patterns: finned, sashimi, chains of g-Fish, …). This said, we acknowledge that Reversible-Subset-Chains (Nice Loops, AICs) may have some appeal for moderately complex instances.

not accept guessing. This was initially only a vague intuition. But, with time, it has received very concrete confirmations from our experience in the Sudoku micro-world (with friends, students, contacts, or from questions of newcomers on forums), considering the way new players spontaneously re-invent it without even having to think of it consciously. Indeed, it does not seem that they reject guessing *a priori*; they start by using it and they feel unsatisfied about it after some time, as soon as they understand that it is an arbitrary step in their solution; "no-guessing" then appears as an additional *a posteriori* requirement. Websites dedicated to the other logic puzzles studied in this book are another source of confirmation: T&E (in various names and usually in informal guises, but always in a form compatible with our formal definition) always appears as the most widely used resolution method, except of course for the easiest puzzles.

– The "T&E vs braids" theorem means that the most natural T&E solving technique, in spite of being strongly anathemised by some Sudoku experts, is not so far from being compatible with the abstract "pure logic" requirement. Moreover, its proof shows that a human solver can always easily modify a T&E solution in order to present it as a braid solution. Thanks to the subsumption theorems or to the more general "T&E(T) vs T-braids" theorem, this remains true when he learns more elaborate techniques (such as Subset or g-Subset rules) and he starts to combine them with T&E.

– Finding the shortest braid solution is a much harder goal than finding any solution based on braids and this is where the main divergence with a solution obtained by mere T&E occurs. For the human solver who started with T&E, it is nevertheless a natural step to try to find a shorter (even if not the shortest) solution. An obvious possibility consists of excising the useless branches of what he has first found; but he can also look for alternative braids, either for the same elimination or for a different one.

– As for the fourth point, a human solver is very likely to have spontaneously the idea of using the continuity condition of whips to guide his search for a contradiction on some target Z: it means giving a preference to pushing further the last tried step rather than a previous one. It is so natural that he may even apply it without being aware of it.

Finally, for a human solver, the transition from the spontaneous T&E procedure to the search for whips can be considered as a very natural process. Learning about Subsets and g-Subsets and looking for them can also be considered as a natural, though different, evolution. And the two can be combined. Once more, there is no unique way of defining what "the best solution" may mean.

Of course, a human player can also follow a very different learning path, starting with application specific rules, such as xy-chains in Sudoku and progressively trying to spot patterns from the ascending sequence of more complex rules following a discovery path similar to that in *HLS*. But, unless he limits himself to moderately complex instances, he cannot avoid the kind of non-reversible chain patterns introduced in this book.

17.1.4. About a strategic level

We have used the confluence property to justify the definition of a "simplest-first" strategy for all the braid and generalised braid (and, by extension, all the whip and generalised whip) resolution theories. This strategy perfectly fits the goals of finding the simplest solution (keeping the above comments on "simplest" in mind) and of rating an instance.

What the "simplest-first" strategy guarantees should be clear: for a resolution theory T with the confluence property, it finds a solution with the smallest T-rating (if there is one); in any case, at each step in any resolution path within T, the available assertion or elimination with the lowest T-rating is applied (or, when there are several, one of the possible assertions or eliminations with this rating is randomly chosen and applied). One thing it does *not* guarantee is that all these steps are necessary for justifying the next ones or that there is no other resolution path with fewer eliminations (not counting Elementary Constraints Propagation).

Other systematic strategies can also be imagined. One of them consists of considering subsets of CSP variables of "same type" and defining special cases of all the rules by restricting them to such subsets of variables and by assigning these cases higher priorities than their initial full version. This is what we have done for Sudoku in *HLS1*, with the 2D rules. It is easy to see that, as the "2D" rules are the various 2D projections (on the rc-, rn-, cn- and bn- spaces) of the "3D" ones presented here, all the 2D-braid theories (in each of these four 2D spaces) are stable for confluence and have the confluence property; it is therefore also true of their union. In *HLS1*, we have shown that 97% of the puzzles in the random Sudogen0 collection can be solved by such 2D rules (the real percentage may be a little less for an unbiased sample). We still consider these rules as interesting special cases that have an obvious place in the "simplest-first" strategy and that may be easier to find and/or to understand for a human player.

Now, it is very unlikely that any human solver would proceed in such a systematic way as described in any of the above two strategies. He may prefer to concentrate on some aspect of the puzzle and try to eliminate a candidate from a chosen cell (or group of cells). As soon as he has found a pattern justifying an elimination, he applies it. This could be called the opportunistic "first-found-first-applied" strategy. And, thanks to stability for confluence, it is justified in all the generalised braid resolution theories defined in this book. In simple terms, there can be no "bad" move able to block the way to the solution. This conclusion is in strong opposition to claims often made in some Sudoku circles that adding a clue (or asserting a value) may make a puzzle harder; such views can only rely on forgetting a few facts: 1) such cases arise only when rules of uniqueness are involved; 2) they arise only when hardness is measured by the SER; 3) if added to a resolution theory with the confluence property, a rule for uniqueness destroys it, unless it is given higher priority than all the other rules; 4) there is a confusion in SER between the priority of a rule and its rating; 5) this confusion prevents rules for uniqueness to apply as soon as they should; 6) as a result, the SER rating of rules for uniqueness is inconsistent.

What may be missing however in our approach is more general "strategic" knowledge for orientating the search: when should one look for such or such pattern? This would be meta-knowledge about how to use the knowledge included in the resolution theories. It would very likely have to be application-specific[5].

But the fact is, we have no idea of which criteria could constitute a basis for such meta-knowledge. Worse, even in the most studied Sudoku CSP, whereas there is a plethora of literature on resolution techniques (sometimes misleadingly called strategies), nothing has ever been written on the ways they should be used, i.e. on what might legitimately be called strategies. In particular, one common prejudice is that one should first try to eliminate bivalue/bilocal candidates (i.e., in our vocabulary, candidates in bivalue rc, rn, cn or bn cells). Whereas this may work for simple puzzles, it is almost never possible for complex ones. This can easily be seen by examining the hard examples of this book (for any of the CSPs we have studied), with the long sequences of whip eliminations necessary before a Single is found: if any of these eliminations had occurred for a bivalue CSP variable, then it would have been immediately followed by a Single.

17.2. About minimal instances and uniqueness

17.2.1. Minimal instances and uniqueness

Considering that, most of the time, we restrict our attention to minimal instances that (by definition) have a unique solution, one may wonder why we do not introduce any "axiom" of uniqueness. Indeed, there are many reasons:

– it is true that we restrict all our statistical analyses of resolution rules to minimal instances, for reasons that have been explained in the Introduction; but it does not entail that validity of resolution rules should be limited *per se* to minimal instances; on the contrary, they should apply to any instance; in a few examples in this book, our rules have even been used to prove non-uniqueness or non-existence of solutions;

– as mentioned in the Introduction, from the point of view of Mathematical Logic, uniqueness cannot be an *axiom*, at least not an axiom that could impose

[5] [Laurière 1978] presents a different perspective, based on general-purpose heuristics.

uniqueness of a solution; for any instance, it can only be an *assumption*; moreover, when incorrectly applied to a multi-solution instance, the *assumption* of uniqueness can lead, via a vicious circle, to the erroneous *conclusion* that an instance has a unique solution; we have given an example in *HLS1*, section XXII.3.1 (section 3.1 of chapter "Miscellanea" in *HLS2*);

– uniqueness is not a constraint the CSP solver (be he human or machine) is expected or can choose to satisfy; in some CSPs or some situations (such as for statistical analyses or for logic puzzles like Sudoku), uniqueness may be a requirement to the provider of instances (he should provide only "well formed" instances, i.e. minimal instances or, at least, instances with a unique solution); the CSP solver can then decide to trust his provider or not; if he does and he uses rules based on it in his resolution paths, then uniqueness can best be described as an oracle; for this reason, in all the solutions we have given, uniqueness is never assumed, but it is proven constructively from the givens;

– the fact is, there is no known way of exploiting the assumption of uniqueness for writing any general resolution rule for uniqueness; and we can take no inspiration in the Sudoku case, because all the known techniques based on the assumption of uniqueness are Sudoku specific;

– in the Sudoku case, if any of the known rules of uniqueness is added in its usual form to a resolution theory with the confluence property, it destroys confluence (see *HLS* for an example); however, we have not explored the possibility of other (more complex) formulations that could preserve it;

– still in the Sudoku case, it does not seem that the known rules for uniqueness have much resolution power; there is no known example that could be solved if they were added to "standard" resolution rules but that could not otherwise.

Of course, we are not trying to deter anyone from using uniqueness in practice, if they like it, in CSPs for which it allows to formulate specific resolution rules, such as Sudoku (where it has always been a very controversial topic, but it has also led to the definition of smart techniques); in some rare cases, it can simplify the resolution paths. We are only explaining why we chose not to use it in our theoretical approach. One should always keep in mind that theory often requires more stringent constraints than practice.

17.2.2. Minimal instances vs density and tightness of constraints

Two global parameters of a CSP, its "density of constraints" and its "tightness", have been identified in the classical CSP literature. Their influence on the behaviour of general-purpose CSP solving algorithms has been studied extensively and they have also been used to compare such algorithms. (As far as we know, these studies have been about unrestricted CSP instances; we have been unable to find any reference to the notion of a minimal instance in the CSP literature.)

Definitions (classical in CSPs): the *density of constraints* of a CSP is the ratio between the number of label pairs linked by some constraint (supposing that all the constraints are binary) and the total number of label pairs; the *tightness* of a CSP is the ratio between the number of label pairs linked by some "strong" constraint (i.e. some constraint due to a CSP variable) and the number of label pairs linked by some constraint.

Density reflects the intuitive idea that the vertices of a undirected graph (here, the graph of labels) can be more or less tightly linked by the edges (here the direct binary contradictions); it also evokes a few general theorems relating the density and the diameter of a random graph (a topic that has recently become very attractive because of communication networks). Tightness evokes the difference we have mentioned between Sudoku or LatinSquare (tightness 100%, for any grid size) and N-Queens (tightness ~ 50%, depending on n).

In the context of this book, relevant questions related to these parameters should be about their influence on the scope of the various types of resolution rules with respect to the set of minimal instances of the CSP. However, how the definitions of these two parameters should be adapted to this context is less obvious than it may seem at first sight. The question is, should one compute these parameters using all the labels of the CSP or only the actual candidates? In the latter case, they would change with each step of the resolution process.

Taking the 9×9 Sudoku example, the computation is easy for labels: there are 729 labels (all the nrc triplets) and each label is linked by some constraint to 8 different labels on each of the n, r, c axes, plus 4 remaining labels on the b axis. Each label is thus linked by some constraint to the same number (28) of other labels and one gets a density equal to 28/728 = 3.846%. More generally, for n×n Sudoku with $n = m^2$, density is: $(4m^2-2m-2)/(m^6-1)$; it tends rapidly to zero (as fast as $4/n^2$) as the size n of the grid increases.

However, considering the first line of each Sudoku resolution path in this book, one can check that for a minimal puzzle, after the Elementary Constraint Propagation rules have been applied (i.e. after the straightforward initial domain restrictions), the number of candidates remaining in the initial resolution state RS_P of an instance P is much smaller. As all that happens in a resolution path depends only on RS_P, a definition of density based on the candidates in RS_P can be expected to be more relevant. But, the analysis of the first series of 21,375 puzzles produced by the controlled-bias generator, leads to the following conclusions, showing that neither the number of candidates in RS_P nor the density of constraints in RS_P have any significant correlation with the difficulty of a puzzle P (measured by its W rating):

– the number of candidates in RS_P has mean 206.1 (far less than the 729 labels) and standard deviation 10.9; it has correlation coefficient -0.20 with the W rating;

– the density of constraints in RS_P has mean 1.58% (much less than when computed on all the labels) and standard deviation 0.05%; its has correlation coefficients -0.16 with the number of candidates in RS_P and -0.06 with the W rating.

One (seemingly more interesting) open question is: is there a correlation between the rating of the current "simplest" possible elimination and the current density (based on the current set of candidates before the elimination). In the instances with a hard first step that we checked, there was no significant deviation from the mean; but the question may be worth more systematic investigation.

Can tightness give better or different insights? This parameter plays a major role in the left to right extension steps of the partial chains of all the types defined in this book. In n×n Sudoku or n×n LatinSquare, tightness is 100%, whatever the value of n; these examples can therefore not be used to investigate this parameter. If there are few CSP variables, there may be few chains. In this context, it should however be noticed that, from the millions of Sudoku puzzles we have solved, problems that appear for the hardest ones solvable by whips or g-whips arise from two opposite causes: not only because there are too few partial whips or g-whips (and no complete ones), but also because there are too many useless partial whips or g-whips (eventually leading to computational problems due to memory overflow).

One idea that needs be explored in more detail is that the possible statistical effects of initial density or tightness of constraints on complexity are minimised (as is the case for the number of givens) by considering the thin layer of minimal instances (because they have a unique solution). But the 16×16 and 25×25 Sudoku examples in section 11.5 show that they cannot be minimised to the point of limiting the depth of T&E in a way independent of density (or grid size).

17.3. About ratings, simplicity, patterns of proof

Our initial motivations included three broad categories of (vague) requirements:

– a "pure logic", "pattern-based", "rule-based", "constructive" solution with "no guessing",

– an "understandable", "explainable" solution,

– and a "simplest" solution.

If the first type has been given a precise meaning and has been satisfied in Part I, and if the second can be considered as a more or less subjective mix of the other two, one may wonder what the third has become or rather how it had to be refined.

17.3.1. About general ratings and the requirement for the "simplest" solution

For any instance P of any CSP, several ratings of P have been introduced: W, B, gW, gB, S+W, S+B, SW, SB,… All of them have been defined in pure logic terms, they are invariant under the symmetries of the CSP (if its constraints are properly modelled) and they are intrinsic properties of P. They have also been shown to be largely mutually consistent, i.e. they assign the same finite ratings "most of the time" to instances in T&E(1)[6] – which probably already includes much more than what can be solved "manually" by normal human beings.

Moreover, if one nevertheless wants to go further, we have defined the WW, BB, W*W, B*B ratings and we have shown that the BB rating is finite for any instance in T&E(2), i.e. that can be solved with at most two levels of Trial-and-Error.

What the multiplicity of these logically grounded ratings also shows is that there is one thing all our formal analyses cannot do in our stead: choosing what should be considered as "simplest". And we strongly believe that there can be no universal *a priori* definition of simplicity of a resolution path, even when one adopts a hardest-step view of simplicity and even for a problem as "simple" as Sudoku, let alone for the general finite CSP. Simplicity can only depend on one's specific goals. For definiteness, let us illustrate this with the Sudoku CSP.

If one is interested in providing examples of some particular set of techniques or promoting them, then a solution considered as the simplest must (tautologically) use only these techniques; the job will then be to provide nice handcrafted examples of such puzzles (and, sometimes, to carefully hide the fact that they are exceptional in the set of all the minimal puzzles); this is the approach implicitly taken by most Sudoku puzzle providers and most databases of "typical examples" associated with computerised solvers. Unfortunately, apart from those here and in *HLS*, we lack both formal studies of such sets of techniques and statistical analyses of their scopes.

If one is interested in the simplest pattern-based solution for all the minimal puzzles, then, considering the statistical results of chapter 6, a whip solution could certainly be considered as the simplest one, *statistically*; a g-whip solution would be a good alternative, as the structural complexity of g-whips is not much greater than that of whips. "Statistically" means that, in rare cases, a better solution including Subsets or g-Subsets or Reversible-Subset-Chains or S-whips or W-whips could be found – "better" in the sense that it would provide a smaller rating (at the cost of using more complex patterns). Although it is hard to imagine a motivation for this when whips or g-whips would be enough, one could also use W_p*-whips or B*-braids, i.e. rely on T&E(2) contradictions as if they were ordinary constraints; doing this may ultimately be only a matter of personal taste [provided that confusion is not

[6] Strictly speaking, this has been shown in precise terms only for 9×9 Sudoku, but there are serious indications that it remains true for the other logic puzzles we have examined.

created by comparing without caution ratings that involve these derived constraints with those that do not].

If one is interested in the "hardest" instances, then it should first be specified precisely what is meant by "hardest" (in particular with respect to which rating); this may seem obvious, but it remains frequent on Sudoku forums to see (implicit) references to two different ratings in the same sentence. In Sudoku, puzzles harder than the "hardest" known ones with respect to the prevailing SER rating keep being discovered. One can consider that Part III of this book (apart from chapter 8) is dedicated to resolution rules for the hardest puzzles (not in the sense of the SER, but in the broader sense that they are not solvable by braids or g-braids, or equivalently by at most one level or T&E or gT&E). Much depends on two parameters: the maximal depth d of Trial-and-Error necessary to solve these instances and the maximal look-ahead p necessary to solve them at depth d-1. [Even for 9×9 Sudoku, although we have shown that there are very strong reasons to conjecture that d = 2 and p = 7, i.e. that every puzzle can be solved by B_7-braids, we have no formal proof of this.]

The T&E(2) land is where many different possibilities appear. For instances there, instead of looking for the simplest solution with respect to the universal BB rating, one can consider two simpler approaches: 1) the $B_?B$ classification, possibly followed by a B_p-braids solution, and 2) the B_p*-braids view. As an illustration of the latter, the solution given for EasterMonster in section 12.3.3.1 proceeds in two steps: the first step provides the main lines of the proof as a sequence of B*-whips[1] eliminations; the second step should contain the "details" of the proof by exhibiting the bi-braids justifying each of these B*-whips[1]. This led us to introduce the general notion of a pattern of proof, but this is a vast topic and we have only skimmed it.

As shown by the sk-loop examples in chapter 13, it may occasionally happen that application-specific patterns (often tightly related to patterns of givens enjoying very particular symmetries or quasi-symmetries) reduce the complexity of an instance (measured in this case by the $B_?B$ classification). However, for the very hardest instances, it may also happen that the whole requirement of simplicity becomes merely meaningless: the existence of extremely rare but very hard instances that cannot be solved by any "simple" rules (in a vague intuitive sense of "simple") is a fact that cannot be ignored.

17.3.2. About adapting the general ratings to an application

The Futoshiki CSP allows two additional comments about how the general ratings introduced in this book can easily be adapted to a particular CSP in order to better take into account any "natural" notion of simplicity in specific applications:

– although “ascending chains” of any size are equivalent to series of whips of length one, they are so natural that presenting them as whips would make the resolution paths look unnecessarily complicated, with lots of elementary and boring steps; this means that, in some cases, our requirement of simplicity cannot be defined based only on formal criteria but it may have to take into account matters of presentation; however, from a technical point of view, this is more a cosmetic than a deep matter;

– “hills” and “valleys” raise a much more interesting question; they are almost as natural and obvious patterns as ascending chains, whatever their size; although they can always be considered as Subsets or as S-whips and their complexity in terms of the equivalent Subsets or S-whips would be much higher than that of ascending chains, it would be intuitively absurd to assign them a much greater complexity, because there is not much difference between finding or understanding hills and valleys and finding or understanding ascending chains, and this does not depend on their size; fortunately, stability for confluence allows to combine any B_n or gB_n theory with hills and valleys of unrestricted size without loosing confluence; this means that hills and valleys can consistently be assigned any rating one wants in the B_n or gB_n hierarchy; said otherwise, one can refine the notion of simplicity in such a way that it becomes adapted to the specificities of the Futoshiki CSP, without loosing the benefits of the general theory; if needed, this illustrates again the importance of the confluence property.

The above remarks can be transposed to Kakuro and to the coupling rules: any resolution theory should include them (and we have accordingly defined the $^+$ variants of all the theories introduced in this book: BRT^+, W_1^+, …).

17.3.3. Similarity between Subset and whip/braid patterns of same size

We have noticed a remarkable formal similarity between the Subset and the whip/braid patterns of same size (see Figure 11.3 and comments there). It has appeared in very explicit ways in the proofs of the confluence property and of the generalised “T&E(T) vs T-braids” theorems for the S_p-braids and B_p-braids. But the general subsumption theorems in section 8.7 and the Sudoku-specific statistical results in Table 8.1 suggest that whips/braids have a much greater resolution power than Subsets of same size. As mentioned in section 8.7.3, these results indicate that the definition of Subsets is much more restrictive than the definition of whips/braids. And Table 11.1 shows that the same kind of very large difference in resolution power remains true for the generalised braids including these patterns as right-linking elements, at least for the Sudoku CSP.

In Subsets, transversal sets are defined by a single constraint. In whips, the fact of being linked to the target or to a given previous right-linking candidate plays a role very similar to each of these transversal sets. But being linked to a candidate is much less restrictive than being linked to it via a pre-assigned constraint; in this respect, the three elementary examples for whips of length 2 in sections 8.7.1.1 and 8.8.1 are illuminating. As shown by the subsumption and almost-subsumption results in section 8.7, the few cases of Subsets not covered by whips because of the restrictions related to sequentiality are too rarely met in practice to be able to compensate for this.

For the above reasons, we conjecture that, in any CSP, whips/braids have a much greater resolution potential than Subsets of same length p, at least for small values of p; and B_p-braids have a much greater resolution potential than S_p-braids. For large values of p, it is likely true also, but it is less clear because there may be an increasing number of cases of non-subsumption but there may also be more ways of being linked to a candidate. Much depends on how many different constraints a given candidate can participate in. This is an area where more work is necessary.

17.4. About CSP-Rules

As mentioned in the Foreword and as can be checked by a quick browsing of this book, it is almost completely written at the logic level; it does not say much about the algorithmic or the implementation levels – beyond the fact that our detailed definitions provide unambiguous specifications for them, whichever computer language one finally chooses. However, a few general indications on CSP-Rules may be welcome.

In this section, it may be useful for the reader not yet familiar with the basic principles of expert systems and/or inference engines to read one of the quick introductions that are widely available on the Web (in particular the notions of a rule base and a fact base); the CLIPS documentation can be browsed, but this is not essential for reading what follows.

17.4.1. CSP-Rules

Almost all[7] the resolution paths appearing in this book were obtained with the current last version of CSP-Rules (version 1.2), the generic finite CSP solver we wrote in the rule-based language of the CLIPS[8] inference engine.

[7] The only exceptions are the few N-Queens examples, for which we did not implement the necessary interface (mainly because we could not find any generator of N-Queen instances and we did not want to spend time on writing one, so that we finally have only very easy instances). Two other exceptions are mentioned explicitly in the text.

[8] CLIPS for Mac OSX, version 6.30. CLIPS is the acronym for “C Language Integrated Production System”; it is a distant descendant of OPS (the Official Production System) but its syntax (inherited from ART, a commercial expert system shell) is much better. CLIPS is free,

In principle, CSP-Rules can also be run on JESS[9] (all the rules we have implemented use only the part of the syntax ensuring compatibility). But JESS is slower and we have given up trying to fill up the compatibility issues when coding the application-specific parts of the various CSPs or to deal with Java-specific memory management problems.

CSP-Rules was designed from the start as a *research tool*, with the main purpose of proving concretely that the general resolution rules and the simplest-first strategy defined in this book can be implemented in a generic way and can lead in practice to real solutions for different CSPs, even for their hard instances. Another purpose was to allow quick implementation of tentative rules and to test their resolution potential with respect to those we had already defined. Finally, we also wanted to make it easy to add application-specific rules (such as sk-loops in Sudoku, hills and valleys in Futoshiki or coupling rules in Kakuro) or to code alternative strategies without having to deal with a programming language like C.

Saying that we conceive CSP-Rules as a research tool means in particular that it was not designed with high speed or low memory purposes in mind, although it includes a few standard tricks to avoid too fast memory explosion and it has been used several times to solve millions of instances. It seems obvious to us that a direct implementation in C or any other procedural language could lead to large improvements in computation times and memory requirements, especially for hard instances – although the exponential increase of the number of partial patterns (with respect to their length) before a full one can be used to produce an elimination is inherent in some instances. The reference to g-labels and S-labels instead of g-candidates and Subsets in g-whips and S-whips is a key for many optimisations of memory.

CSP-Rules is a descendant of SudoRules, the Sudoku solver we originally developed in parallel with the writing of *HLS*. As the main parts of the later versions of SudoRules were already written in an almost application independent way, it was easy to maximally reduce and to isolate the unavoidably application-specific parts. The version of SudoRules (16.2) based on CSP-Rules that was used in the Sudoku examples presented in this book is 100% equivalent to (i.e. it produces exactly the same resolution paths as) the last version before the split (namely 15b.1.12, which has been our version of reference at the time of writing *CRT*), when the same rules are enabled.

---

which probably largely contributed to make it one of the most widely adopted shells. Another reason is that CLIPS implements the RETE algorithm that made OPS famous, with all the improvements that appeared since that time, making it one of the most efficient shells.

[9] Current version as of this writing, i.e. 6.1p2. JESS is the acronym for "Java Expert System Shell"; it was initially the Java version of CLIPS; but, due to the underlying language, it has grown up differently and there are now compatibility issues.

The current version of CSP-Rules implements the following sets of rules (we have also implemented other tentative rules but they are not mentioned in this book because they did not lead to interesting results):

– BRT (i.e. ECP + Single + Contradiction detection + Solution detection),
– bivalue-chains, whips, braids,
– g-bivalue-chains, g-whips, g-braids,
– forcing whips, forcing braids,
– bi-whips, bi-braids,
– forcing bi-whips, forcing bi-braids,
– W*-whips, B*-braids.

For each of these patterns and for each possible length, CSP-Rules has two or three rules (one or two for building the partial patterns, one for detecting the full ones and doing the eliminations), plus an activation rule (used mainly for memory optimisation) and a tracking rule (as they are mainly used for tracking the numbers of partial patterns and for statistics, their output does not appear in the resolution paths given here). All these rules are written only in the generic terms of candidates, g-candidates, CSP-variables, links and g-links. Their effective output (what we want to appear in a resolution path) is controlled by a set of global variables.

CSP-Rules also implements the generic parts of functions used in the left-hand side of rules (when it is both possible and more efficient to make a test [linked, glinked, …] than to write an additional explicit condition pattern) or for the interfacing with specific applications (e.g. for printing the different steps of the resolution path – although it already implements the generic parts of the output functions). Any application must provide the specific parts of these functions.

CSP-Rules also provides the possibility of computing T&E(T) and bi-T&E(T) for any resolution theory T whose rules are programmed in CSP-Rules.

Because it was too hard to do this in sufficiently efficient ways, CSP-Rules does not implement a generic version of Subsets (let alone of g-Subsets). Instead, it has a standard version of Subsets (upto size four) valid for CSPs based on a square (or rectangular) grid (like most of the examples in this book), with a sub-version with blocks as in Sudoku. In the Kakuro CSP, its adaptation to the case of Subsets restricted to sectors was straightforward.

The generation of instances is not part of CSP-Rules.

17.4.2. Configuration of an application for solving an instance

Any application (any particular CSP) has a configuration file allowing to choose the resolution theory one wants to use, i.e. which patterns should be enabled and up to which size. Technically, "enabled" means loaded into the rule base; it does not mean "activated". An enabled pattern gets activated only if necessary (i.e. if shorter ones are nor enough to solve the instance under consideration).

Consistency of the chosen parameters is ensured automatically, e.g.

– for any pattern P[n] depending on a size or length parameter n, if P[n] is explicitly enabled, then P[n-1], … P[1] are automatically enabled;

– if g-braids of length upto n are enabled, then braids and g-whips of length upto n are enabled if they have not been explicitly enabled with a larger length;

– if g-whips of length upto n are enabled, then whips of length upto n are enabled if they have not been explicitly enabled with a larger length;

– if braids of length upto n are enabled, then whips of length upto n are enabled if they have not been explicitly enabled with a larger length…

However, bivalue-chains are not automatically enabled when whips are enabled. This may be changed in the future. But we have found it useful to keep this degree of freedom, as enabling special types of whips sometimes allows to find different whip resolution paths (see an example in section 5.10.3).

17.4.3. Resolution strategies predefined in CSP-Rules

The current version of CSP-Rules has only one resolution strategy, the "simplest-first", with the priorities as described in section 7.5.2:
ECP > S >
biv-chain[1] > whip[1] > g-whip[1] > braid[1] > g-braid[1] >
… > …
biv-chain[k] > whip[k] > g-whip[k] > braid[k] > g-braid[k] >
biv-chain[k+1] > whip[k+1] > g-whip[k+1] > braid[k+1] > g-braid[k+1] > …

A few things are easy to change, such as assigning braids[k] a higher priority than g-whips[k] or introducing more special cases of whips. For radically different strategies, the main problem would not be to code them in CSP-Rules, but to first define them (see the remarks in section 17.1.4).

17.4.4. Applications already interfaced to CSP-Rules

As of this writing, the current version of CSP-Rules has application-specific interfaces (and in some cases a few application-specific resolution rules, possibly including alternative versions of the rules in BRT, e.g. different rules for Naked and Hidden Singles) for the following CSPs: *LatinSquare, Sudoku, Futoshiki, Kakuro, Map-colouring, Numbrix® and Hidato®*. For each of them, the volume of the source code of the application-specific part (including mainly input-output functions) is between 3% and 5% of the total generic CSP-Rules part. For Sudoku, more functions had been written in the previous versions of SudoRules, but they were mainly intended for statistical analyses and cannot be considered as necessary for the normal resolution of instances; moreover, with a little more adaptation work, they could also be made generic, if needed.