1 Introduction

Constraints are a natural means of knowledge representation in many disparate fields. A constraint often takes the form of an equation or inequality, but in the most abstract sense is simply a logical relation among several variables expressing a set of admissible value combinations. The following are simple examples: the sum of two variables must equal 30; no two adjacent countries on the map may be coloured the same; the helicopter is designed to carry only one passenger, although a second can be carried in an emergency; the maths class must be scheduled between 9 and 11am, but it may be moved to the afternoon later in the term. It is this generality and simplicity of structure which underly the ubiquity of the constraint-based representation in Artificial Intelligence.

The process of identifying a solution to a problem which satisfies all specified constraints is termed constraint *satisfaction*. It has emerged as a fundamental technique for inference. Many years of research effort have been spent on developing constraint satisfaction and interest is still growing in these methods as they become increasingly widely applied. The unique structure and properties of constraint satisfaction problems are considered to be deserving of detailed examination in order that they can be exploited in the search for solutions.

Constraint-based reasoning is declarative, allowing the formulation of knowledge without specifying how the constraints should be satisfied. This decoupling of problem and solution technique has allowed the constraint satisfaction problem to be tackled from a number of different perspectives. Hence, a variety of techniques have been developed for finding partial/complete solutions of different kinds.

Despite its inherent simplicity, a constraint-based representation can be used very naturally to express real, difficult problems. For example, the problems of interpreting an image, scheduling a collection of tasks, planning an evacuation procedure, or diagnosing a fault in an electrical circuit can all be viewed as instances of the constraint satisfaction problem. The importance of constraint satisfaction to such an array of different fields has led to the development of new programming languages to support it.

There are, however, some problems that cannot easily be modelled via standard constraint-based techniques. For example, the classical formulation

offers no means to deal with over-constrained problems that admit no "perfect" solution. Consider, for instance, the opening example involving the capacity of the helicopter. In reality, if no other solution could be found a "compromise" solution would be constructed whereby the helicopter would carry a second passenger.

Furthermore, classical constraint satisfaction cannot efficiently support problems whose structure is subject to change. In the opening example concerning the scheduling of the maths class it is noted that the constraint may change. If this situation occurs, the natural approach is to attempt to repair the old solution, disturbing it as little as possible. This, however, is beyond the scope of the standard constraint satisfaction definition.

This book identifies and examines existing extended constraint-based methods which address these limitations, and shows how they can be further combined to solve still more complex problems.

1.1 Solving Classical CSPs

The classical *Constraint Satisfaction Problem* (CSP) [31, 100, 112, 162] involves a fixed, finite set of problem *variables*, each with an associated finite *domain* of potential values. A set of *constraints* range over the variables, specifying the allowed combinations of *assignments* of values to variables. In order to solve a classical CSP, it is necessary to find one or all assignments to all the variables such that all constraints are simultaneously satisfied. A constraint satisfaction solution procedure must find one/all such assignments or prove that no such solution exists.

It has been shown that the classical CSP is NP-complete [104]. Therefore, solution techniques for classical CSP, although many and varied, generally employ some form of search to explore the space of possible variable assignments (see Figure 1.1*a*). A number of such search procedures have been developed, the most basic being the well known *Backtrack* search [13]. Efficiency is crucial in solving the complex combinatorial problems to which CSP methods are typically applied. Therefore, constraint satisfaction algorithms have become increasingly sophisticated in their attempts to minimise search effort. The standard measures of search effort by which CSP algorithms are judged are the size of the space explored and the number of constraint *checks*, the latter being an examination of a constraint to see whether a particular combination of assignments is allowable.

Given the difficulty of solving classical CSPs, much of the research into constraint satisfaction has gone into finding ways to reduce the effort required. For example, a class of *pre-processing* algorithms [12, 98, 116] has been developed which simplify the problem prior to search. A process known as *enforcing consistency* [98] is used which examines the problem structure, making explicit constraints implied by, but not actually present in, the original problem formulation (see Figure 1.1*b*). The new constraints may manifest

in terms of the removal of elements (which have been shown not to form part of any solution to the problem) from certain domains, the removal of some tuples in existing constraints, or the creation of entirely new constraints. Identifying such "hidden" constraints saves time during subsequent search by restricting the size of the search space which must be explored.

However, a "once only" pre-processing phase can have only a limited effect on search effort. The most successful CSP algorithms combine sophisticated tree search with a certain amount of of consistency enforcing at each search node to form *hybrids* [121, 127] . Hybrid tree search/consistency enforcing algorithms take advantage of the fact that as search progresses, parts of the problem become "fixed" as variable assignments are made. Hence, simpler sub-problems are formed which may be examined by a consistency enforcing technique in exactly the same way as in a pre-processing phase (see Figure 1.1c). This method quickly uncovers "dead ends" in the search where the remaining sub-problem is shown to be insoluble. Much search effort is therefore saved as the algorithm can immediately try a different path towards a solution.

Heuristics also play an important role in reducing search effort, as presented in Figure 1.1. Ordering the three basic operations of a constraint satisfaction algorithm: variable selection, value selection and the order in which constraint checks are made, can have a substantial impact on efficiency. The guiding principle on which most CSP heuristics are based is to focus on the most constrained area of the problem, minimizing the branching factor of the search [136]. Heuristics have also evolved from early attempts which simply examine the size of each variable's domain to more sophisticated techniques which consider a range of factors such as the structure of the problem and the *tightness* of the constraints. The tightness of a constraint is the proportion of possible assignment combinations to the constrained variables disallowed by the constraint.

1.2 Applications of Classical CSP

The inherent simplicity of a constraint-based representation is the foundation for its potentially extremely wide applicability. A variety of different problems (see Figure 1.2) can, when formulated appropriately, be seen as instances of the classical CSP which can then be solved using the techniques described briefly in the previous section and in greater detail in chapter 2.

An early application of constraint satisfaction techniques appears, for example, in the field of machine vision. Waltz presents a method for producing semantic descriptions given a line drawing of a scene, leading to the Waltz algorithm [173]. In this case, variables are picture junctions which have potential values consisting of possible interpretations as corners, and constraints specify that each edge in the drawing must have the same interpretation at

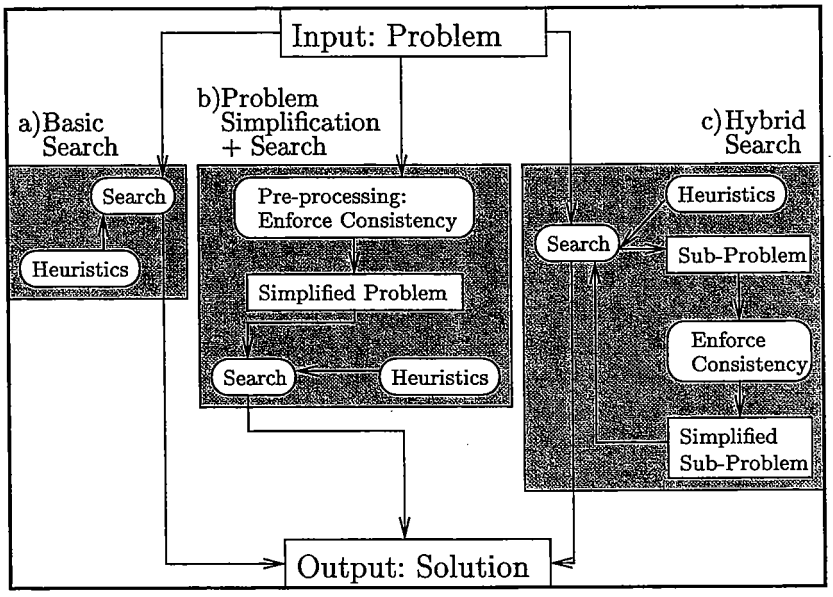

Fig. 1.1. Increasing Sophistication of Classical Constraint Satisfaction Algorithms.

both of its ends. Other examples of constraint satisfaction in machine vision can be found in [18, 87, 99, 117, 135, 147].

Another area that involves extensive use of constraint satisfaction is that of AI planning. Graphplan [14] reduces classical domain-independent planning [175] to the solution of a CSP. By this method, large efficiency gains were made as compared to previous state-of-the-art planning algorithms. A related approach [83] converts a planning problem into the solution of a propositional satisfiability (SAT) problem [70, 178]. SAT is the problem of deciding if there is an assignment to the variables in a propositional formula that make the formula true, and is a restriction of classical CSP. Several specialised search techniques for this type of problem have been developed [27, 146] which can then be used indirectly to solve the planning problem. Further applications to planning are to be found in [81, 154]. Chapter 5 contains more detail on this topic.

The field of scheduling also directly employs constraint satisfaction techniques. Fox presented the ISIS system for the scheduling of factory job-shops [52]. It is capable of representing a variety of constraint types, including organisational goals and physical constraints. Techniques have been developed to support both the cumulative [7] and disjunctive cases [6], where tasks can and cannot share resources respectively. Other CSP applications to scheduling are presented in [24, 126, 134].

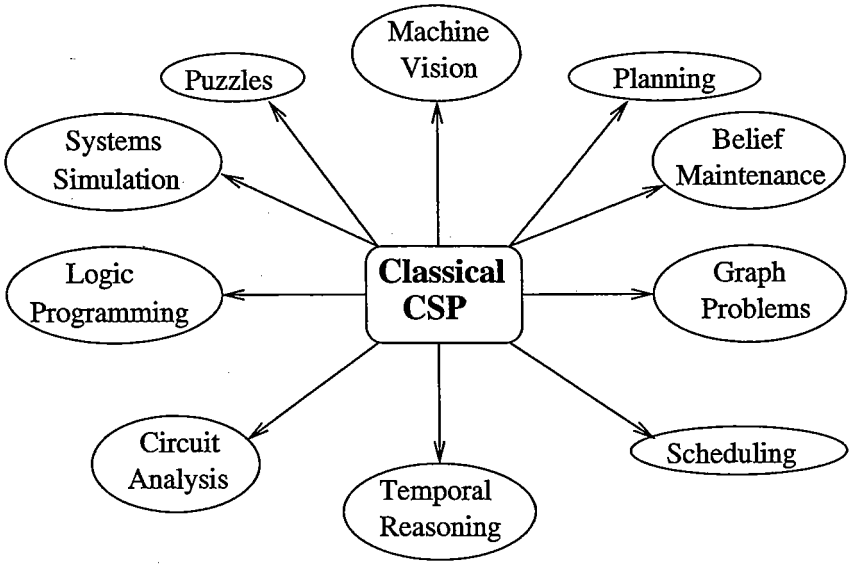

Fig. 1.2. Some Applications of Classical Constraint Satisfaction.

Classical CSP also plays a central role in the field of systems simulation [89, 107, 148]. This application requires the construction of a tree of possible system behaviours given an initial system state and some constraints. Starting from a particular system state and given rules specifying how individual variables' states may change, CSP techniques are used to construct efficiently all potential next system states, thus growing the required behaviour tree.

Embedding constraint satisfaction techniques into programming languages has proven to be very successful [169]. Constraint logic programming (CLP) enhances traditional declarative programming languages such as Prolog [78] to harness the power of CSP techniques. While searching a database of facts, a CLP program makes use of constraints to prune a significant proportion of the search tree. This results in large efficiency gains to the extent that CLP programs can rival custom solutions developed using imperative programming languages such as C. Other examples are to be found in [17, 165, 166].

The list of applications of CSP is as broad as it is deep. Further examples include belief maintenance [28, 32, 39, 102], puzzles [121], timetabling [48], temporal reasoning [1, 133, 155, 161], graph problems [21, 51, 103, 164], structural design [74], assigning system components to tasks [163], and circuit analysis [153]. Further discussion of the applications of classical CSP may be found in [169].

1.3 Limitations of Classical CSP

Classical CSP has a long and successful history of practical application. As the techniques of classical constraint satisfaction have been applied to more complex real problems, however, it has become increasingly clear that the classical formulation is insufficient. There are two main areas of weakness: firstly, the inability to deal with *flexibility* in the problem description in case no "perfect" solution exists, and secondly the ability to deal gracefully with changes to the problem structure.

In the case of both limitations identified above, a requirement is exhibited for an extension to classical CSP in order to cope gracefully with the demands made by more realistic problems. Such extensions do exist, albeit separately, and are discussed briefly in the following sections, and in more detail in Chapter 2.

1.3.1 Flexible CSP

Classical constraint satisfaction techniques support only *hard* constraints specifying exactly the allowed variable assignment combinations. Hard constraints are *imperative* (a valid solution must satisfy all of the constraints) and *inflexible* (constraints are either wholly satisfied or wholly violated). In reality, of course, problems rarely exhibit this rigidity of structure. It is common for there to be *flexibility* inherent which can be used to overcome over-constrainedness (i.e. a problem that admits no solution) by indicating where a sensible compromise can be made.

In order to address this weakness, classical constraint satisfaction techniques have been extended to incorporate different types of flexible constraints. The flexible constraints used attempt to model the "soft" constraints often found in real problems. The different methods of extending classical CSP to model flexibility are known under the umbrella term of flexible CSP (FCSP) [42, 58].

One common example is the use of *priorities* or weights attached to each constraint indicating its relative importance. In the over-constrained case, where a compromise-free solution does not exist, if only the constraints with the lowest importance are *relaxed* (i.e. removed from the problem) the likelihood of finding a soluble problem with a useful solution is maximised. For example, the GARI system [37] provides weighted pieces of advice in the process of producing plans for machining parts given their drawings. If there is no "perfect" solution, then the least important pieces of advice can be ignored.

Further applications of flexible CSP include: flexible job-shop scheduling [41], management of over-constrained resource scheduling problems [8], and control in the wine industry [137].

1.3.2 Dynamic CSP

A further limitation of classical CSP is in its assumption of a *static* problem. That is, once the sets of variables, domains and constraints have been defined, they are fixed for the duration of the solution process. In reality, of course, problems can be subject to change either as a solution is being constructed, or while a constructed solution is in use. Consider a large scale scheduling problem, such as scheduling astronomical observations on the Hubble Space Telescope [114]. Not only is the initial problem extremely difficult to solve (e.g. tens of thousands of observations must be scheduled each year, subject to a great variety of constraints), but it also changes continually: new observations may be submitted at any time for scheduling. Classical constraint satisfaction can deal at best only clumsily with this situation, considering the changed problem as an entirely new problem which must be solved from scratch.

A special case also exists in which the problem structure changes not due to external factors, but as a result of choices made in partially solving the problem. Consider a simple example involving configuring a new car for a customer. If the customer wants a sun-roof, further choices must be made concerning its type (glass or metal?) and operation (manual or electric?). The sub-choices are meaningless without the selection of a sun-roof in the first place. Classical CSP can only be coerced into solving this type of problem by explicating all ways in which the problem structure can change based on tentative sub-solutions.

To address these types of problem, the techniques of *dynamic* constraint satisfaction (DCSP) have also been developed. In order to model problems which change over time, constraints are added to and removed from the current problem description as necessary [32]. Specialised techniques are then employed to re-use as much of the solution or partial solution obtained for a problem before it changed with respect to the new problem state [166, 114, 167]. In order to model problems whose state changes based on the assignments of one or more variables, special constraints are introduced which serve to activate sub-parts of the problem structure given certain assignments [115].

1.4 Dynamic Flexible CSP

The two types of extension to classical CSP briefly described above: flexible CSP and dynamic CSP respectively, separately offer significant advances in the range of problems that CSP techniques can solve. Unfortunately, current dynamic constraint satisfaction research is founded almost exclusively on classical CSP, unable to take advantage of flexible constraints in a dynamic environment, and flexible CSP research is limited to static problems. Little has been done to combine dynamic and flexible constraint satisfaction in

order to maintain the benefits of both individual approaches to solve more complex problems.

The principal contribution of this book is in presenting the combination of these two previously disparate extensions, exploring techniques for coping with dynamic flexible CSPs (DFCSPs [110]). Based on a systematic review of classical, dynamic, and flexible CSP techniques, a matrix of possible instances of DFCSP is identified. Each instance combines particular individual methods for dynamic and flexible CSP and is suitable for modelling a different type of problem.

One such instance, combining restriction/relaxation-based dynamic CSP [32] and fuzzy flexible CSP [42] is investigated in detail, both in terms of solution techniques and the structure of the problems produced. The individual dynamic and flexible components are powerful and have already been successful in their own right. Hence, this particular instance of DFCSP is chosen as being sufficiently general to find a range of uses and as an indicator of the utility of DFCSP.

In this book, two techniques are developed to solve this type of DFCSP. The first is a novel integrated algorithm, Flexible Local Changes. The soundness and completeness of this algorithm is established, and an analysis of its time and space complexity is performed. The second is a heuristic enhancement of a standard branch and bound [93] approach to solving flexible CSP. In addition, an extensive empirical analysis is made of the structure and properties of this type of DFCSP and the performance of the two solution techniques.

1.5 Flexible Planning: a DFCSP Application

The second principal contribution of this book is in the application of DFCSP techniques to the field of AI Planning [175]. This field is an active and long-established research area with a wide applicability to such tasks as automating data-processing procedures [23], game-playing [152], and large-scale logistics problems [177]. Solving these types of problems demands a highly efficient approach due to their size and complexity. However, as per classical CSP, classical AI planning is unable to support flexibility in the problem description. Providing such an ability is an important step forward for the real-world utility of planning research.

Preparatory to the development of the flexible planner, a novel approach to classical plan synthesis in the Graphplan framework [14] is defined using dynamic CSP techniques [113]. It is based on the hierarchical decomposition of the initial problem into sequences of sub-problems. A dynamic CSP algorithm is then employed to efficiently solve these sequences. Empirical tests on benchmark problems show that this algorithm performs competitively against the current state-of-the-art solvers, and in some cases shows a performance gain.

Using the above non-flexible planner as a basis, a flexible planning system is developed. Firstly, the classical AI Planning problem description is extended to incorporate flexible constraints. For flexible plan synthesis, a hierarchical decomposition of the problem is made, similar to that used in the classical case. Hence, plan synthesis is reduced to the solution of a hierarchy of DFCSPs. Flexible Graphplan (FGP), a new flexible planner, is developed to solve such flexible planning problems. A test suite of flexible problems engineered to exhibit different structural properties is created, as well as a more complex benchmark flexible problem. FGP's performance on these flexible problems is analysed in the book.

1.6 Structure

The remainder of this book is structured as itemised below:

- Chapter 2: *The Constraint Satisfaction Problem.* This chapter contains a detailed review of the current state of the art of constraint satisfaction techniques. It begins with a formal description of classical constraint satisfaction problems and the methods for representing and reasoning with such problems. The wide variety of solution techniques for solving classical CSPs are then discussed in detail. The limitations of classical CSP are addressed, concentrating in particular on the need to support flexible constraints and an evolving problem structure. A review of existing extensions to the classical framework to support the former or the latter is then presented.
- Chapter 3: *Dynamic Flexible Constraint Satisfaction.* This chapter discusses the further evolution of constraint satisfaction techniques to support simultaneously flexible constraints and dynamic changes to the problem structure: dynamic flexible CSP (DFCSP). A matrix of possible DFCSP instances is defined via the combination of the different available dynamic and flexible extensions to classical CSP. A specific representative instance, fuzzy *rr*DFCSP, combining fuzzy constraint satisfaction and restriction/relaxation based dynamic CSP is considered in detail and two basic solution algorithms, one based on branch and bound [93] techniques and the other, Flexible Local Changes, on an existing dynamic CSP algorithm [167] are developed for this type of problem. A small worked example illustrates the utility of both algorithms.
- Chapter 4: *An Empirical Study of fuzzy rrDFCSPs.* This chapter contains an extensive empirical analysis of the structure and properties of fuzzy *rr*DFCSPs. A large set of random problem sequences are generated, varying five different parameters to alter their composition. Hybrid variants of the algorithms developed in Chapter 3 are then tested on the random problems, as well as the utility of various heuristics and efficiency-improving techniques. In particular, the results show the utility of using

dynamic flexible CSP algorithms as opposed to coercing algorithms capable of supporting flexibility only into solving the dynamic sequences.

- Chapter 5: *Dynamic CSP in Domain-independent AI Planning.* Domain-independent AI Planning is introduced in this chapter as an application domain for both dynamic and dynamic flexible CSP. The recent trend of reducing the planning problem to a CSP in order to make efficiency gains is continued. A novel solution technique, GP-*rr*DCSP, based on applying restriction/relaxation-based dynamic CSP techniques to the standard Graphplan [14] framework is developed and described in detail.
- Chapter 6: *GP-rrDCSP: Experimental Results.* An investigation is presented in this chapter into the utility of the GP-*rr*DCSP AI planner, as compared with state of the art planners on benchmark problems. Both basic and enhanced versions of GP-*rr*DCSP are tested over five different planning domains. The enhanced version performs particularly well, with both versions being competitive across the problem set. The results therefore establish GP-*rr*DCSP as a solid foundation upon which to build a flexible planning system in the following chapters.
- Chapter 7: *The Flexible Planning Problem and Flexible Graphplan.* This chapter defines the novel flexible planning problem, which extends classical domain-independent planning to support soft constraints. Fuzzy constraint satisfaction underlies this definition, allowing the association of subjective truth degrees with propositions and satisfaction degrees with plan operators and goals. Flexible plan synthesis is then supported via fuzzy *rr*DFCSP, generalising the dynamic CSP methods used for plan synthesis (as developed in Chapter 5) to create Flexible Graphplan (FGP).
- Chapter 8: *FGP: Experimental Results.* An evaluation of Flexible Graphplan is presented in this chapter. Initially, a specially constructed test suite of problems is used in order to examine the relationship between the structure of flexible planning problems and their expected solution difficulty. This test suite is also used to evaluate the utility of various enhancements to the basic FGP algorithm. Secondly, the novel flexible benchmark Rescue problem is introduced and used to evaluate FGP's performance on a more complex example.
- Chapter 9: *Conclusions.* This chapter contains conclusional remarks and details of important future work. Examples of future work include the exploration of other instances of DFCSP, leading to the eventual development of a generalised framework capable of supporting many such instances.
- Appendices: The four appendices respectively contain companion pseudocode, proofs of the soundness and completeness of the new algorithms developed in this book, full results for the empirical study performed in Chapter 4, and the details of the test suite and Rescue problem used in Chapter 8.

1.7 Contributions and their Significance

The main contributions of this book and their significance are as follows:

- Literature review: A systematic review of the constraint satisfaction literature is made, encompassing classical, flexible and dynamic techniques. To the best of the author's knowledge, this is the first systematic review of the CSP literature for several years.
- Identification of a matrix of dynamic flexible constraint satisfaction problems (DFCSPs): these problem types enable the modelling of complex problems, combining flexible constraints with a dynamic problem specification, previously beyond the reach of constraint satisfaction techniques.
- Development of DFCSP Solution methods: Principally, this involves the creation of a novel algorithm, Flexible Local Changes (FLC), to solve fuzzy *rr*DFCSPs, a particular representative instance of DFCSP. This is the first algorithm to address such problems. Furthermore, a heuristic enhancement of branch and bound is developed also to solve fuzzy *rr*DFCSPs. It is claimed that these algorithms are easily adaptable to solving other instances of DFCSP.
- Empirical analysis of fuzzy rrDFCSPs: Almost 40,000 random problem instances were generated, varying five parameters to alter their composition. This is the first such study of the structure and properties of this type of problem and also the first evaluation of the performance of variants of FLC and the branch and bound algorithm.
- Development of GP-*rr*DCSP, a new classical domain-independent planner: within the popular Graphplan framework, it hierarchically decomposes the original problem and uses *rr*DCSP techniques to solve the resultant sequences of sub-problems. This algorithm offers a new perspective on the solution of classical planning problems and is found to be competitive with the state of the art via experimental analysis.
- Creation of the Flexible Planning Problem: This is an extension of the classical planning problem using fuzzy sets. Through the association of satisfaction degrees with operators and truth degrees with propositions, this type of problem allows a tradeoff between making compromises to satisfy the plan goals to a certain extent in a short number of steps, versus reducing the number of compromises made but extending the length of the plan. This is the first specification of a planning problem which allows such a tradeoff to take place.
- Development of Flexible Graphplan (FGP): This is a novel algorithm to solve flexible planning problems. Based on the GP-*rr*DCSP algorithm, it too relies on the hierarchical decomposition of the initial problem into dynamic sequences of sub-problems. These flexible sub-problems are then solved using the FLC algorithm. FGP is the first algorithm to be able to solve flexible planning problems, and contains the first practical use of the FLC algorithm.