Chapter 10
Optimization in CSPs

10.1 Introduction

In previous chapters, we have looked at techniques for solving CSPs in which all solutions are equally good. In applications such as industrial scheduling, some solutions are better than others. In other cases, the assignment of different values to the same variable incurs different costs. The task in such problems is to find optimal solutions, where optimality is defined in terms of some application-specific functions. We call these problems Constraint Satisfaction Optimization Problems (CSOP) to distinguish them from the standard CSP in Definition 1-12.

Moreover, in many applications, the constraints are so tight that one normally cannot satisfy all of them. When this is the case, one may want to find compound labels which are as close to solutions as possible, where closeness may be defined in a number of ways. We call these problems Partial Constraint Satisfaction Problems (PCSP).

Relatively little research has been done in both CSOP and PCSP by the CSP research community. In this chapter, these problems will be formally defined, and relevant techniques for solving them will be identified.

10.2 The Constraint Satisfaction Optimization Problem

10.2.1 Definitions and motivation

All optimization problems studied in operations research are constraint satisfaction problems in the general sense, where the constraints are normally numerical. Here, we use the term Constraint Satisfaction Optimization Problems (CSOP) to refer to the standard constraint satisfaction problem (CSP) as defined in Definition 1-12, plus the requirement of finding optimal solutions.

Definition 10.1:

> A **CSOP** is defined as a CSP (Definition 1-12) together with an optimization function f which maps every solution tuple to a numerical value:
>
> $$(Z, D, C, f)$$
>
> where (Z, D, C) is a CSP, and if S is the set of solution tuples of (Z, D, C), then
>
> $f: S \rightarrow$ numerical value.
>
> Given a solution tuple T, we call $f(T)$ the ***f*-value** of T. ■

The task in a CSOP is to find the solution tuple with the optimal (minimal or maximal) f-value with regard to the application-dependent optimization function f.

Resource allocation problems in scheduling are CSOPs. In many scheduling applications, finding just any solution is not good enough. One may like to find the most economical way to allocate the resources to the jobs, or allocate machines to jobs, maximizing some measurable quality of the output. These problems are CSOPs.

In order to find the optimal solution, one potentially needs to find all the solutions first, and then compare their f-values. A part of the search space can only be pruned if one can prove that the optimal solution does not lie in it — which means either no solution exists in it (which involves knowledge about solutions) or that the f-value in any solution in the pruned search space is sub-optimal (which involves knowledge about the f-values).

10.2.2 Techniques for tackling the CSOP

Finding optimal solutions basically involves comparing all the solutions in a CSOP. Therefore, techniques for finding all solutions are more relevant to CSOP solving than techniques for finding single solutions.

Among the techniques described in the previous chapters, solution synthesis techniques are designed for finding all solutions. Problem reduction methods discussed in Chapter 4 are in general useful for finding all solutions because they all aim at reducing the search space. The basic search strategies introduced in Chapter 5 are in general applicable to both finding all solutions and finding single solutions. Variable ordering techniques which aim at minimizing backtracking (the minimal width ordering) and minimizing the number of backtracks (the minimal bandwidth ordering) are more useful for finding single solutions than all solutions. On the other hand, the *fail first principle* (FFP) in variable ordering is useful for finding all solutions as well as single solutions, because it aims at detecting futility as soon as possible so as to prune off more of the search space.

Techniques for ordering the values are normally irrelevant when all solutions are required because in such a case, all values must be looked at. Values ordering will be useful in gather-information-while-searching strategies if more nogood sets can be discovered in searching one subtree rather than another, and one has the heuristics to order the branches (values) in order to maximize learning.

In the following sections, we shall introduce two important methods for tackling CSOPs which have not been introduced in this book so far. They are the branch and bound (B&B) algorithm and genetic algorithms (GAs). The former uses heuristics to prune off search space, and the latter is a stochastic approach that has been shown to be effective in combinatorial problems.

10.2.3 Solving CSOPs with branch and bound

In solving CSOPs, one may use heuristics about the *f* function to guide the search. Branch and bound (B&B), which is a general search algorithm for finding optimal solutions, makes use of knowledge on the *f* function. He we continue to use the term *solution tuple* to describe compound labels which assign values to all those variables satisfying all the constraints (Definition 1-13). Readers should note that a solution tuple here need not refer to the optimal solution in a CSOP. B&B is a well known technique in both operations research and AI. It relies on the availability of good heuristics for estimating the best values ('best' according to the optimization function) of all the leaves under the current branch of the search tree. If reliable heuristics are used, one could be able to prune off search space in which the optimal solution does not lie. Thus, although B&B does not reduce the complexity of a search algorithm, it could be more efficient than the chronological backtracking search. It must be pointed out, however, that reliable heuristics are not necessarily available. For simplicity, we shall limit our discussion to the depth first branch and bound strategy and its application to the CSOP in this section.

10.2.3.1 A generic B&B algorithm for CSOP

To apply the B&B to CSOP, one needs a heuristic function h which maps every compound label CL to a numerical value (h: $CL \rightarrow$ number). We call this value the ***h*-value** of the compound label. For the function h to be admissible, the h-value of any compound label CL must be an *over-estimation* (*under-estimation*) of the *f*-value of any solution tuple which projects to CL in a *maximization* (*minimization*) problem.

A global variable, which we shall refer to as the *bound*, will be initialized to minus infinity in a maximization problem. The algorithm searches for solutions in a depth first manner. It behaves like Chronological_Backtracking in Chapter 2, except that before a compound label is extended to include a new label, the h-value of the current compound label is calculated. If this h-value is less than the bound in a maximization problem, then the subtree under the current compound label is pruned. Whenever a solution tuple is found, its *f*-value is computed. This *f*-value will become the new bound if and only if it is greater than the existing bound in a maximization problem. When this *f*-value is equal to or greater than the bound, the newly found solution tuple will be recorded as one of the, or the best solution tuples so far. After all parts of the search space have been searched or pruned, the best solution tuples recorded so far are solutions to the CSOP.

The Branch_and_Bound procedure below outlines the steps in applying a depth-first branch and bound search strategy to solving the CSOP, where the maximum *f*-value is required. Minimization problems can be handled as maximization problems by substituting all *f*- and *h*-value by their negation. For simplicity, this procedure returns only one solution tuple which has the optimal *f*-value; other solution tuples which have the same *f*-value are discarded.

```
PROCEDURE Branch_and_Bound( Z, D, C, f, h );
/* (Z, D, C) is a CSP; f is the function on solution tuples, the f-value is
    to be maximized; h is a heuristic estimation of the upper-bound of
    the f-value of compound labels */
BEGIN
    /* BOUND is a global variable, which stores the best f-value found
        so far; BEST_S_SO_FAR is also a global variable, which
        stores the best solution found so far */
    BOUND ← minus infinity; BEST_S_SO_FAR ← NIL;
    BNB( Z, { }, D, C, f, h );
    return(BEST_S_SO_FAR);
END /* of Branch_and_Bound */


PROCEDURE BNB(UNLABELLED, COMPOUND_LABEL, D, C, f, h);
BEGIN
    IF (UNLABELLED = { }) THEN
        BEGIN
            IF (f(COMPOUND_LABEL) > BOUND) THEN
                BEGIN           /* only one optimal solution is returned */
                    BOUND ← f(COMPOUND_LABEL);
                    BEST_S_SO_FAR ← COMPOUND_LABEL;
                END;
        END;
    ELSE IF (h(COMPOUND_LABEL) > BOUND) THEN
        BEGIN
            Pick any variable x from UNLABELLED;
            REPEAT
                Pick any value v from $D_x$;
                Delete v from $D_x$;
                IF (COMPOUND_LABEL + {<x,v>} violates no con-
                    straints)
                THEN BNB(UNLABELLED – {x}, COMPOUND_LABEL
                    + {<x,v>}, D, C, f, h);
            UNTIL ($D_x$ = { });
        END /* of ELSE IF */
    END /* of BNB */
```

Note that the Branch_and_Bound procedure is only sound and complete if $h(CL)$ indeed returns an upper-bound of the *f*-value. If the heuristic *h* may underestimate the *f*-value, then the procedure may prune off search space where optimal solutions lie, which causes sub-optimal solution tuples to be returned.

The efficiency of B&B is determined by two factors: the quality of the heuristic function and whether a "good" bound is found at an early stage. In a maximization problem, if the *h*-values are always over-estimations of the *f*-values, then the closer the estimation is to the *f*-value (i.e. the smaller the *h*-value is without being smaller than the *f*-value), the more chance there will be that a larger part of the search space will be pruned.

A branch will be pruned by B&B if the *h*-value of the current node is lower than the bound (in a maximization problem). That means even with the heuristic function fixed, B&B will prune off different proportion of the search space if the branches are ordered differently, because different bounds could be found under different branches.

10.2.3.2 Example of solving CSOP with B&B

Figure 10.1 shows an example of a CSOP. The five variables x_1, x_2, x_3, x_4 and x_5 all have numerical domains. The *f*-value of a compound label is the summation of all the values taken by the variables. The task is to find the solution tuple with the maximum *f*-value.

Figure 10.2 shows the space explored by simple backtracking. Each node in Figure 10.2 represents a compound label, and each branch represents the assignment of a value to an unlabelled variable. The variables are assumed to be searched under the ordering: x_1, x_2, x_3, x_4 and x_5. As explained in the last section, B&B will perform better if a tighter bound is found earlier. In order to illustrate the effect of B&B, we assume that the branches which represent the assignment of higher values are searched first.

Figure 10.3 shows the space searched by B&B under the same search ordering as