An Approach for Achieving Local Consistency in Binary CSP Involving Disjunctive Constraints

Carlos Castro
INRIA Lorraine & CRIN
615, rue du Jardin Botanique, BP 101,
54602 Villers-lès-Nancy Cedex, France
e-mail: Carlos.Castro@loria.fr

Abstract

In this paper we present an approach for systematic manipulation of disjunctive constraints when verifying local consistency in CSP. We propose to decompose a set of constraints in several subproblems, making explicit difference between elementary and disjunctive constraints. Then local consistency is verified for each subproblem and the results, elimination of impossible values for the variables, are communicated through membership constraints. The algorithm stops when there are no more changes in these constraints. We use constructive disjunction to deal with disjunctive constraints in order to prune the search space. We prove that if we can decompose the set of disjunctive constraints in several subproblems whose set of variables are disjoints we can do better than the existing approaches, like choice point. This general approach for solving constraints fits very well in the particular case of scheduling problems, one of the most successful applications of constraint programming.

Keywords: Artificial Intelligence, Constraint Satisfaction Problems, Disjunctive Constraints, Constructive Disjunction.

1 Introduction

One of the most difficult problems in constraint solving comes from disjunctive constraints. Several techniques have been proposed by the Artificial Intelligence community to deal with this kind of constraints. Because of its theoretical and practical interest, manipulation of disjunctive constraints in Constraint Satisfaction Problems (CSP) is a hot research topic [13, 4, 16]. In this work we present an approach for systematic manipulation of disjunctive constraints when verifying local consistency. We first propose to decompose a set of constraints in two subproblems, the first one with all the elementary constraints and the second one with all the disjunctive constraints. Then we apply a graph decomposition algorithm on the second subproblem in order to obtain several sets of disjunctive constraints whose set of variables are disjoints. We verify local consistency for each subproblem and the results, elimination of impossible values for the variables, are communicated through membership constraints. The algorithm stops when there are no more changes in the membership constraints. We prove that if we can decompose the set of disjunctive constraints in at least two subproblems we can do better than the existing approaches, like choice point. We have implemented these ideas in a prototype for solving CSP and we have carried out some simple benchmarks to validate this theoretical result. We have realised that this general approach to manipulate CSP fits very well in the case of scheduling problems, one of the most successful applications of constraint programming [3]. This paper is organised as follows. Section 2 presents CSP, its definition and a brief description of techniques used to solve them. Section 3 introduces disjunctive constraints and presents different approaches used by the CSP community to deal with them. In section 4 we present our approach in detail. Finally, in section 5 we conclude the paper.

2 CSP

In this section we present a formal definition of CSP and briefly describe techniques used to solve them. More details can be found in [2].

2.1 Definitions

An *elementary constraint* $c^?$ is an atomic formula built on a signature $\Sigma = (\mathcal{F}, \mathcal{P})$, where $\mathcal{F}$ is a set of ranked function symbols and $\mathcal{P}$ a set of ranked predicate symbols, and a denumerable set $\mathcal{X}$ of variable symbols [1]. Elementary constraints are combined with usual first-order connectives. We denote the set of constraints built from Σ and $\mathcal{X}$ by $\mathcal{C}(\Sigma, \mathcal{X})$. Given a structure $\mathcal{D} = (D, I)$, where I is an interpretation function and D the domain of this interpretation, a $\langle \Sigma, \mathcal{X}, \mathcal{D}\rangle$-CSP is any set $C = (c_1^? \wedge \ldots \wedge c_n^?)$ such that $c_i^? \in \mathcal{C}(\Sigma, \mathcal{X})\ \forall i = 1, \ldots, n$. A solution of $c^?$ is a mapping from $\mathcal{X}$ to D that associates to each variable $x \in \mathcal{X}$ an element in D such that $c^?$ is satisfiable in $\mathcal{D}$. A solution of C is a mapping such that all constraints $c_i^? \in C$ are satisfiable in $\mathcal{D}$. Given a variable $x \in \mathcal{X}$ and a non-empty set $D_x \subseteq D$, the *membership constraint* of x is a relation given by $x \in^? D_x$. We use these membership constraints to make explicit the domain reduction process during the constraint solving. In practice, the sets D_x have to be set up to D at the beginning of the constraint solving process, and constraint propagation will eventually reduce them. As all first-order connectives can be expressed in terms of conjunctions and disjunctions we consider the set of constraints C as follows

$$C = \bigwedge_{x \in \mathcal{X}} (x \in^? D_x) \wedge \bigwedge_{i \in I} (c_i^?) \wedge \bigwedge_{j \in J} (c1_j^? \vee c2_j^?)$$

[1] For clarity, constraints are syntactically distinguished from formulae by a question mark exponent on their predicate symbols.

where I is the set of elementary constraints and J the set of disjunctive constraints. For simplicity reason we will only consider disjunctive constraints as disjunctions of only two elementary constraints. We use e, n, and a to denote the number of constraints, the number of variables and the size of the variable's domain, respectively, in a CSP, and we also denote by $Var(c^?)$ the set of variables in a constraint $c^?$. In this work we only consider Binary CSP, i.e., problems where at most two variables are involved in each constraint [2].

2.2 Solving CSP

Typical tasks defined in connection with CSP are to determine whether a solution exists, and to find one or all the solutions. In this section we present three categories of techniques used in processing CSP: Searching Techniques, Problem Reduction, and Hybrid Techniques. Kumar's work [7] is an excellent survey on this topic.

Searching Techniques in CSP Searching consists of techniques for systematic exploration of the space of all solutions. The simplest force brute algorithm *generate-and-test*, also called *trial-and-error search*, is based on the idea of testing every possible combination of values to obtain a solution of a CSP. This generate-and-test algorithm is correct but it faces an obvious combinatorial explosion. Intending to avoid that poor performance the basic algorithm commonly used for solving CSPs is the *simple backtracking* search algorithm, also called *standard backtracking* or *depth-first search with chronological backtracking*, which is a general search strategy that has been widely used in problem solving. Although backtracking is much better than generate and test, one almost always can observe pathological behaviour. Bobrow and Raphael have called this class of behaviour *thrashing* [1]. Thrashing can be defined as the repeated exploration of subtrees of the backtrack search tree that differ only in inessential features, such as the assignments to variables irrelevant to the failure of the subtrees. The time complexity of backtracking is $O(a^n e)$, i.e., the time taken to find a solution tends to be exponential in the number of variables [9]. In order to avoid the resolution of this kind of complex problem, the notion of problem reduction has been developed.

Problem Reduction in CSP Problem reduction techniques transform a CSP to an equivalent problem by reducing the values that the variables can take. Problem reduction is often refered to as *consistency maintenance* [12]. Consistency concepts have been defined in order to identify in the search space classes of combinations of values which could not appear together in any set of values satisfying the set of constraints. Mackworth [8] proposes three levels of consistency: node, arc and path-consistency. These names come from the fact that general graphs have been used to represent binary CSP [12]. The most widely used level of consistency is arc consistency whose definition is the following:

Given the variables $x_i, x_j \in \mathcal{X}$ and the constraints $c_i^?(x_i), c_j^?(x_j), c_k^?(x_i, x_j) \in C$, the arc associated to $c_k^?(x_i, x_j)$ is *consistent* if

$$\forall \alpha \in \alpha_{\mathcal{D}}^{\mathcal{X}} \;\; \exists \alpha' \in \alpha_{\mathcal{D}}^{\mathcal{X}} : \alpha \in Sol_{\mathcal{D}}(x_i \overset{?}{\in} D_{x_i} \wedge c_i^?(x_i))$$

$$\Rightarrow \alpha' \in Sol_{\mathcal{D}}(x_j \overset{?}{\in} D_{x_j} \wedge c_j^?(x_j) \wedge c_k^?(\alpha(x_i), x_j)).$$

A network of constraints is *arc-consistent* if all its arcs are consistent. In [10] Mohr and Henderson propose the algorithm AC-4 whose worst-case time complexity is $O(ea^2)$ and they prove its optimality in terms of time.

[2] As a disjunction is considered itself as a constraint we do not allow more than two variables involved in a disjunctive constraint.

It is important to realize that the varying forms of consistency algorithms can be seen as *approximation algorithms*, in that they impose *necessary* but not always *sufficient* conditions for the existence of a solution on a CSP, that is why they are often refered to as local consistency algorithms.

Hybrid Techniques As backtracking suffers from thrashing and consistency algorithms can only eliminate local inconsistencies, hybrid techniques have been developed. In this way we obtain a complete algorithm that can solve all problems and where thrashing has been reduced. Hybrid techniques integrate constraint propagation algorithms into backtracking in the following way: whenever a variable is instantiated [3], a new CSP is created; a constraint propagation algorithm can be applied to remove local inconsistencies of these new CSPs [17]. Embedding consistency techniques inside backtracking algorithms is called Hybrid Techniques. A lot of research has been done on algorithms that essentially fit the previous format. In particular, Nadel [11] empirically compares the performance of the following algorithms: Generate and Test, Simple Backtracking, Forward Checking, Partial Lookahead, Full Lookahead, and Really Full Lookahead. These algorithms primarily differ in the degrees of arc consistency performed at the nodes of the search tree.

3 Disjunctive Constraints

The combination of two elementary constraints with a disjunction operator is called a disjunctive constraint. A lot of combinatorial problems involve this kind of constraints. For example, in scheduling problems these constraints come from the fact that several tasks must use the same resource and the limited capacity of that resource does not allow to perform all tasks at a same time [14]. Let $Task_{ij}$ the start time of task i of job j and d_{ij} the duration of task i of job j. On a machine performing a simple task at a time, the capacity constraints enforce the mutual exclusion for each pair of tasks assigned to the same machine. If we consider task k of jobs i and j, the fact that on machine k job i runs before job j or vice versa can be expressed by the following disjunctive constraint

$$Task_{kj} \geq^? Task_{ki} + d_{ki} \vee Task_{ki} \geq^? Task_{kj} + d_{kj}$$

In order to perform all tasks using the same resource a sequential order must be established, these precedence relations that are not known a priori are determinated by the solution to a scheduling problem. This feature changes the nature of the problem, and no efficient polynomial algorithm can be exhibited for solving all problems involving disjunctive constraints. In this section we present some techniques to deal with disjunctive constraints.

3.1 Choice Point

The first approach used by the Constraint Logic Programming (CLP) community to deal with disjunctive constraint was to choose one disjunct during the search process, i.e., an a priori choice is made and one disjunct is posted, if the resulting set of constraint is inconsistent then the other disjunct is chosen and posted. This approach is based on the general idea of backtracking, the search space is not reduce actively but only when a clause is non deterministically chosen possibly leading to combinatorial explosion. In the worst case 2^{ND} combinations of constraints have to be analysed, where ND stand for the number of disjunctions. So, for many problems such an approach introduces too many choice points and yields an unsatisfactory performance.

[3]Variable instantiation is also called labelling process.

3.2 Binary Variables

Another technique to deal with disjunctive constraints, widely used by the Operational Research community, is to introduce binary $(0, 1)$ variables [15]. Each of both values activates one disjunct and deactivates the other. A constraint is said activated if it is trivially satisfied for all value combinations of all its variables. This gives the effect of setting the constraint at a choice point, but in this case the labelling routine can select the variable for labelling at the best point in the search. Considering the disjunctive constraint

$$Task_{kj} \geq^? Task_{ki} + d_{ki} \vee Task_{ki} \geq^? Task_{kj} + d_{kj}$$

which establishes that on machine k job i runs before job j (first disjunct) or job j runs before job i (second disjunct). We introduce a binary variable $X_{ij} \in \{0, 1\}$ in the following way

$$\begin{aligned} (1 - X_{ij}) * M + Task_{kj} &\geq^? Task_{ki} + d_{ki} \\ X_{ij} * M + Task_{ki} &\geq^? Task_{kj} + d_{kj} \end{aligned}$$

where M is a large enough number. In this way we have transformed a disjunctive constraint in a conjunction of two elementary constraints. If $X_{ij} = 1$ then the second constraint is active, trivially satisfied, and the first disjunct will constraint the value of the variables, in other words, job i runs before job j on machine k.

As soon as a value is assigned to the binary variable during the solving process one disjunct will be entailed by the set of constraints and the other one will be used to reduce the variables domain. In the worst case the labelling process will try values for the binary variables, using it as a choice point as in the first approach, i.e., in the worst case we also have to analyse 2^{ND} cases.

3.3 Constructive Disjunction

A rather new approach is called constructive disjunction, which lifts common information from the alternatives [13]. We explain this idea using an example taken from [16]. Consider the following set of constraints that enforces the mutual exclusion of jobs A and B on machine i

$$\begin{aligned} Task_{iA} + 7 \leq^? Task_{iB} \quad &\vee \quad Task_{iB} + 7 \leq^? Task_{iA} \\ Task_{iA} &\in^? \{1 \ldots 10\} \\ Task_{iB} &\in^? \{1 \ldots 10\} \end{aligned}$$

The first disjunct constraints $Task_{iA} \in^? \{1, 2, 3\}$ and $Task_{iB} \in^? \{8, 9, 10\}$, the second disjunct constraints $Task_{iA} \in^? \{8, 9, 10\}$ and $Task_{iB} \in^? \{1, 2, 3\}$. Thus independent which alternative will succeed we know that neither $Task_{iA}$ nor $Task_{iB}$ can take the values $\{4, 5, 6, 7\}$. The common information that we can deduce from both alternatives is $Task_{iA} \in^? \{1, 2, 3, 8, 9, 10\}$ and $Task_{iB} \in^? \{1, 2, 3, 8, 9, 10\}$, the union of the set of remaining values for the variables in each disjunct. This is the essence of constructive disjunction, extract common information from the alternatives and thus allows other parts of the computation to benefit from this extra information, disjunctive constraints are used actively without a priori choices. In the worst case, if not extra information can be extracted from the disjunctions, the labelling process will analyse 2^{ND} cases.

4 Systematic Manipulation of Disjunctive Constraints

In this section we present our approach for achieving local consistency in CSP involving disjunctive constraints. The level of local consistency most widely used by the CSP community, arc consistency, can be achieved in a polynomial time for a set of elementary constraints, but when we consider disjunctive constraints the problem becomes harder. The general idea we propose is to decompose a problem in two subproblems, the first one with all the elementary constraints and the second one with all the disjunctive constraints. Local consistency can be achieved efficiently for the first subproblem, and the second subproblem is used to extract extra information using a constructive disjunction approach. As both subproblems share variables, information about values of variables must be communicated, that is done through the use of membership constraints. Figure 1 presents the general schema considering the set of constraints in the form that we explain in section 2.

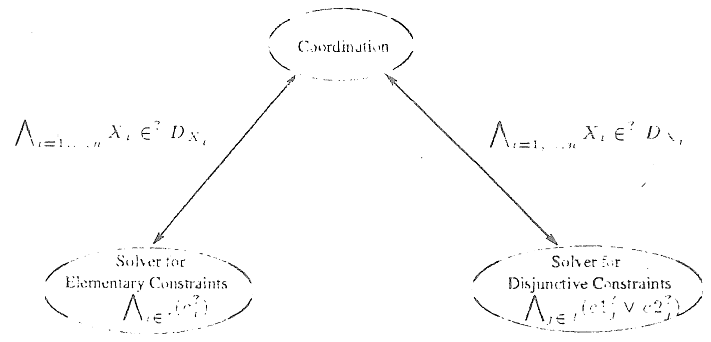
Coordination
$\bigwedge_{i=1,\dots,n} X_i \in^? D_{X_i}$
$\bigwedge_{i=1,\dots,n} X_i \in^? D_{X_i}$
Solver for
Elementary Constraints
$\bigwedge_{i \in I} \{c_i^?\}$
Solver for
Disjunctive Constraints
$\bigwedge_{j \in J} \{c1_j^? \vee c2_j^?\}$

Figure 1: General schema for manipulating the set of constraints

But, as verifying local consistency for a set of disjunctive constraints is a hard problem, we propose to decompose the set of disjunctive constraints as much as possible. In order to do that we use a graph decomposition algorithm which will detect the maximum number of subproblems whose set of variables are disjoints. In this way we deal with a set of easier problems, and the added cost is not significative since the decomposition algorithm has a linear time complexity. Figure 2 presents the refined general schema.

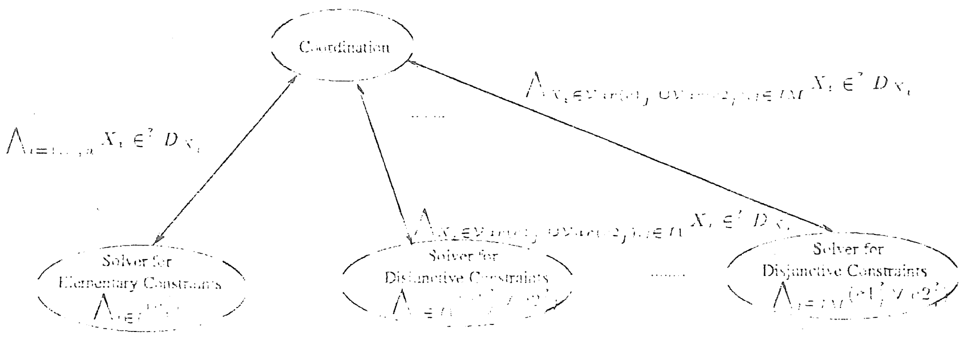

Figure 2: Refined general schema

Once we apply the graph decomposition algorithm on the set of disjunctive constraints we obtain M

subsets

$$J = \bigcup_{i=1..M} J_i$$

so the initial set C of disjunctive constraints can be expressed by

$$\bigwedge_{i=1..M} \bigwedge_{j \in J_i} (c1_j^? \vee c2_j^?)$$

and such that

$$\forall i \in J_k, j \in J_l : k \neq l \Rightarrow (Var(c1_i) \cup Var(c2_i)) \cap (Var(c1_j) \cup Var(c2_j)) = \emptyset$$

In the diagram of figure 2 the coordination level is in charge of decompose the set of disjunctive constraints and add the adequate membership constraints coming from the subproblem with elementary constraints. In the same way, the coordination level send to this subproblem the results obtained from the set of disjunctive constraints. Local consistency is verified for each subproblem and the results are communicated through the membership constraints. the algorithm stops when there is no more changes in the set of membership constraints. For clarity reasons we express the sets of elementary and disjunctive constraints in the following way

$$C_1 = \bigwedge_{i \in I} c_i^?$$

$$C_2 = \bigwedge_{i=1..M} C_{2J_i}$$

where

$$C_{2J_i} = \bigwedge_{j \in J_i} (c1_j^? \vee c2_j^?)$$

These ideas are expressed in the following algorithm

```
begin
Get $C_1$ and $C_2$ from the set of constraints $C$
repeat
  Verify local consistency for $C_1$
  Decompose $C_2$ as $\bigwedge_{i=1..M} C_{2J_i}$
  for each $C_{2J_i}$ do
    Verify local consistency using a constructive disjunction approach
    if one of the constraints, $c1_j$ and $c2_j$ for $j \in J_i$, is inconsistent with the associated membership constraints then
      Eliminate the disjunctive constraint from $C_2$
      Add the other elementary constraint to $C_1$
    end_if
  end_do
until The set of membership constraints does not change
end
```

Theorem 1 *The algorithm terminates and it is correct.*

Proof: Termination of local consistency algorithms is well known, so we only have to prove the termination of the loop **Repeat**. In the worst case, after verification of local consistency for all C_{2J_i}, the resulting membership constraints will be different because only one element has been eliminated, as we have at most n variables in the set of disjunctive constraints and each variable can take a values, the maximal number of iterations is (an), so at most after (an) iterations the algorithm will terminate. Correctness is evident because we only eliminate values when they are locally inconsistent, so we do not eliminate any solution.

Theorem 2 *If we can decompose a set of constraints in M subgraphs the worst case time complexity of our algorithm is bounded by $M2^{ND-M+1}$, where ND is the total number of disjunctive constraints.*

Proof: If $ND_i : i = 1, \ldots, M$ denotes the number of disjunctive constraints in the subgraph i, in each iteration we have to verify local consistency for $\sum_{i=1}^{M} 2^{ND_i}$ sets of disjunctive constraints. As $ND = \sum_{i=1}^{M} ND_i$, we have $\sum_{i=1}^{M} 2^{ND_i} \leq M2^{ND-M+1}$. In the worst case $M = 1$, so the worst case time complexity is $O(2^{ND})$, the same result as the choice point approach.

Evidently the use of our approach depends on the specific characteristics of the problem. First, the more we can decompose the set of disjunctive constraints the more efficient will be the use of constructive disjunction because we will deal with easier problems (this is the meaning of the expression $M2^{ND-M+1}$). Second, the benefits of using constructive disjunction depend on the extra information we can extract from the disjunctions, so the more restricted are the constraints the more information we can extract from them, i.e., more impossible values will be eliminated. That is why our approach must be seen as a preprocessing step, after that another technique should to be used, such as choice point, for example.

In [3] we have applied our approach to solve job-shop problems where the general idea of decompose a set of disjunctive constraints correspond to manipulate as a whole the constraints related to a particular machine but independently for different machines, so for problems involving M machines we are sure that we can decompose the set of disjunctive constraints in M subproblems.

Finally, it is important to note that our approach can be naturally implemented using several solvers in parallel for verifying local consistency for the sets of disjunctives constraints.

4.1 Examples

Two advantages of the approach presented in this work with respect to the choice point can be better understood in the following examples. We consider the following problem:

$$
\begin{array}{rcccl}
 & z & \overset{?}{\leq} & 5 & \qquad (1)\\
 & y & \overset{?}{\leq} & z & \qquad (2)\\
 & x & \overset{?}{\leq} & y & \qquad (3)\\
x \overset{?}{\geq} 1 & & \vee & & x \overset{?}{\leq} 6 \qquad (4)\\
y \overset{?}{\geq} 2 & & \vee & & y \overset{?}{\leq} 7 \qquad (5)\\
z \overset{?}{\geq} 4 & & \vee & & z \overset{?}{\leq} 1 \qquad (6)\\
 & x, y, z & \overset{?}{\in} & [0, \ldots, 10] & \qquad (7)
\end{array}
$$

Constraints 1, 2 and 3 correspond to elementary constraints, constraints 4, 5 and 6 correspond to disjunctive constraints. If we verify local consistency for the set of elementary constraints we obtain the modified membership constraints $x \overset{?}{\in} [0, \ldots, 3]$, $y \overset{?}{\in} [1, \ldots, 4]$ and $z \overset{?}{\in} [2, \ldots, 5]$. Applying a decomposition algorithm on the set of disjunctive constraints 4 - 6 we obtain three subproblem, so we verify local consistency for each of them using a constructive disjunction approach.

- Verifying local consistency for the disjunctive constraint 4 and the associated membership constraint $x \in^? [0, \ldots, 3]$, we obtain $x \in^? [1, \ldots, 3]$ and $x \in^? [0, \ldots, 3]$ [4]. So, the resulting membership constraint using a constructive disjunction approach is $x \in^? [0, \ldots 3]$, it is unchanged.
- Verifying local consistency for the disjunctive constraint 5 and the associated membership constraint $y \in^? [1, \ldots, 4]$, we obtain $y \in^? [2, \ldots, 4]$ and $y \in^? [1, \ldots, 4]$. So, the resulting membership constraint is $y \in^? [1, \ldots, 4]$, it is unchanged.
- Verifying local consistency for the disjunctive constraint 6 and the associated membership constraint $z \in^? [2, \ldots 5]$, the first branch has no solution and the second one $z \in^? [4, \ldots 5]$. So, the resulting membership constraint is $z \in^? [4, \ldots 5]$.

But, as additional information we know that only one disjunct is possible in the third disjunction. We can post the second disjunct of the third disjunctive constraint (constraint 6), the only possible alternative in that disjunction, as an elementary constraint, eliminate that disjunctive constraint, and verify again local consistency for the set of elementary constraints.

Another interesting situation occurs if we replace the third disjunctive constraint (constraint 6) by the following:

$$z \geq^? 6 \vee z \leq^? 1$$

When we verify local consistency for this constraint and the membership constraint $z \in^? [4, \ldots 5]$ both disjunct are inconsistent with the membership constraint, so no value remains possible for the variable z and then the problem has no solution. It is interesting to note that this situation could have been also detected using a choice point approach, but it is well known that the number of leaves visited by that approach depend on the order of the input of the constraints, however the performance of our approach does not depend on that order. If we create the choice points posting the disjunctive constraints in the order 4, 5 and 6 we will visit 8 leaves before detect that the problem has no solution, but if we create the choice points posting firstly the third disjunctive constraint we have to visit only 2 leaves to verify that the problem has no solution.

We can see how our approach allows to reduce the search space and also eliminate some disjunctive constraints.

4.2 Implementation

In general, we are interested in solving CSP using computational systems, a logical framework integrating rewrite rules and strategies [5]. We have implemented a prototype of a solver for CSP which is currently executable in the system ELAN [6], an interpreter of computational systems[5]. We have integrated in this prototype the ideas presented in this work as a preprocessing phase that carries out local consistency verification. Once we have decomposed the set of disjunctive constraints in several subproblems we run a solver for each subproblem in order to verify local consistency in parallel, in this way we can fully profit of the advantages of this approach. All details about the prototype can be obtained at `http://www.loria.fr/~castro/CSP/csp.html`.

[4] To verify local consistency for these subproblems we use a choice point approach that generates two branchs, each one for each disjunct.

[5] ELAN is available via anonymous ftp at `ftp.loria.fr` in the directory `/pub/loria/protheo/softwares/Elan`. Further information can be obtained at `http://www.loria.fr/equipe/protheo.html/PROJECTS/ELAN/elan.html`

5 Conclusion

We have presented an approach to deal systematically with disjunctive constraints in CSP when verifying local consistency. We have shown how this approach can eventually reduce the search space and eliminate some disjunctions. We have proved that when we can decompose the set of disjunctive constraints we can do better than the existing approaches, like choice point. In real life problems, like scheduling, it is often possible to decompose the set of disjunctive constraints, so we think that our approach can be an interesting contribution for practical applications of CSP techniques. As future work we are interested in to estimate the benefits of apply this approach in terms of parameters of the set of constraints and we are also interested in to integrate these ideas in the search process.

References

[1] Daniel G. Bobrow and Bertram Raphael. New Programming Languages for Artificial Intelligence Research. *Computing Surveys*, 6(3):153–174, September 1974.

[2] Carlos Castro. Solving Binary CSP Using Computational Systems. In José Meseguer, editor, *Proceedings of the First International Workshop on Rewriting Logic and its Applications. RWLW'96*, volume 5, pages 245–264, Asilomar, Pacific Grove, CA, USA, September 1996. Electronic Notes in Theoretical Computer Science.

[3] Carlos Castro. An Approach for Systematic Manipulation of Disjunctive Constraints in CSP: An Application to Scheduling Problems. Submitted, 1997.

[4] Jean Jourdan and Thierry Sola. The Versatility of Handling Disjunctions as Constraints. In Maurice Bruynooghe and Jaan Penjam, editors, *Proceedings of the 5th International Symposium on Programming Language Implementation and Logic Programming, PLILP'93, Tallinn, Estonia*, number 714 in Lecture Notes in Computer Science, pages 60–74. Springer-Verlag, August 1993.

[5] Claude Kirchner, Hélène Kirchner, and Marian Vittek. Designing Constraint Logic Programming Languages using Computational Systems. In Pascal Van Hentenryck and Vijay Saraswat, editors, *Principles and Practice of Constraint Programming. The Newport Papers*, pages 131–158. The MIT press, 1995.

[6] Claude Kirchner, Hélène Kirchner, and Marian Vittek. *ELAN. User Manual*. INRIA Lorraine & CRIN, Campus scientifique, 615, rue du Jardin Botanique, BP-101, 54602 Villers-lès-Nancy Cedex, France, November 1995.

[7] Vipin Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *Artificial Intelligence Magazine*, 13(1):32–44, Spring 1992.

[8] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8:99–118, 1977.

[9] Alan K. Mackworth and Eugene C. Freuder. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 25:65–74, 1985.

[10] Roger Mohr and Thomas C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986.

[11] B. Nadel. Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 287–342. Springer-Verlag, 1988.

[12] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[13] Pascal van Hentenryck, Vijay Saraswat, and Yves Deville. Design, Implementation, and Evaluation of the Constraint Language cc(FD). In *Constraint Programming: Basic and Trends. Selected Papers of the 22nd Spring School in Theoretical Computer Sciences*. Springer-Verlag, Chtillon-sur-Seine, France, May 1994.

[14] Mark G. Wallace. Applying Constraints For Scheduling. In Mayoh B and Penjaam J, editors, *Constraint Programming*. NATO ASI Series. Springer-Verlag, 1994.

[15] H.P. Williams. *Model Building in Mathematical Programming*. J. Wiley and Sons, New York, 1978.

[16] Jörg Würtz and Tobias Müller. Constructive Disjunction Revisited. In *Proceedings of the twentieth German Annual Conference on Artificial Intelligence, KI-96*, September 1996. To appear.

[17] Martin Zahn and Walter Hower. Backtracking along with constraint processing and their time complexities. *Journal of Experimental and Theoretical Artificial Intelligence*, 8:63–74, 1996.