A Hybrid Schema for Systematic Local Search

William S. Havens and Bistra N. Dilkina

Intelligent Systems Lab
Simon Fraser University
Burnaby, British Columbia
Canada V5A 1S6
{havens, bnd}@cs.sfu.ca

Abstract. We present a new hybrid constraint solving schema which retains some systematicity of constructive search while incorporating the heuristic guidance and lack of commitment to variable assignment of local search. Our method backtracks through a space of complete but possibly inconsistent solutions while supporting the freedom to move arbitrarily under heuristic guidance. The version of the schema described here combines minconflicts local search with conflict-directed backjumping. It is parametrized by a variable ordering relation which controls the order in which the search space is explored. Preliminary experimental results are given comparing two instances of the schema to forward checking with conflict-directed backjumping [17] (*FC-CBJ*).

1 Introduction

Constraint Satisfaction Problems (CSPs) are ubiquitous in scheduling, planning, configuration and other combinatorial tasks. In general, solving CSPs is NP-hard. Modern search algorithms for CSPs are of two basic types: 1) constructive backtrack algorithms and; 2) stochastic local search algorithms. Constructive algorithms systematically expore a search tree of possible solutions [12]. They construct a consistent partial solution by extending it one variable assignment at a time and backtracking on failure, until every variable is assigned a consistent value under the problem constraints. Unfortunately, bad variable assignments made early in the search tree persist for an exponential length of time until a proof of inconsistency for a whole subtree can be found [5, 11].

In contrast, local search algorithms (*e.g.* [8, 14, 18]) start with a complete but inconsistent assignment of variables and iteratively change individual assignments until all the constraints are satisfied. Local search is incomplete, but it is not plagued by the tyranny of early bad decisions in the search tree. These algorithms are free to move arbitrarily in the search space by following the heuristic gradient.

However, the lack of systematicity of local search algorithms makes remembering the history of past states problematic. In the worst case, a state cache exponential in the size of the CSP is required [1].

There has been recent research into hybrid search schemes which combine desirable aspects of constructive and local search methods [4, 7, 11, 13–16]. These algorithms attempt to retain some of the systematicity of constructive search while allowing for a maximum amout of flexibility of movement in the search space. In the work reported here, we define a new hybrid scheme, called *Maximal Constraint Solving (MCS)*, which extends these methods as follows:

- The scheme operates from a nearly complete instantiation of values to variables [7, 16].
- The notion of a maximally consistent solution relaxes the requirement that the constructed partial solution remain consistent [4].
- Forward checking of both assigned and unassigned variables is performed [16].
- Systematicity is enforced using a nogood cache of known inconsistent variable assignments [7, 10, 11, 19].
- Various randomized (arbitrary) backtracking methods are supported [9, 11, 13, 21].
- The scheme is parameterized by a variable ordering relation which realizes multiple possible search control stuctures.

In Section-2 we give some preliminary definitions. In Section-3 we outline a novel characterization of backtrack search in order to explain the MCS schema and give an informal synthesis of the schema. Section-4 then presents a more formal characterization of the scheme which is in turn followed in section-5 by some preliminary experimental results. Finally in section-6, we briefly relate our work to other relevant research in the field.

2 Definitions

We begin with a few necessary definitions then describe the schema.

Definition 1. *A* Constraint Satisfaction Problem (CSP)*, is a triple* (V, D, C)*, where* $V = \{v_1, ..., v_n\}$ *is a set of variables with corresponding domains* $D = \{D_{v_1}, ..., D_{v_n}\}$ *and* C *is a set of k-ary constraints. A* k-ary constraint *defines the allowed combinations of values for a subset of k variables* $(v_1, ..., v_k)$ *from* V*.*

We are interested in variable assignments, written $\langle x = a \rangle$, for variable $x \in V$ and $a \in D_x$, which during search act as unary constraints that are repeatedly added and retracted. In addition, it will be useful to refer to sets of variable assignments, called *labels*, to represent environments, solutions and nogoods.

Definition 2. *A* label*,* $\lambda(X) = \{\langle x = a \rangle\}_{x \in X}$*, is a set of variable assignments, where* $X \subseteq V$ *and* $a \in D_x$*.*

A label corresponds to a snapshot of variable assignments at a particular point in the search. The current assignments of these variables may or may not correspond to the assignments specified in the label.

Definition 3. *A label, λ, is* valid *iff every variable assignment $\langle x = a \rangle \in \lambda$ is the current assignment of the variable x.*

During search we will induce nogoods, i.e. partial assignments that are not part of any consistent solution.

Definition 4. *A* nogood *is a label, $\lambda_\perp = \{\langle x = a \rangle\}_{x \in X}, X \subseteq V$, whose partial assignment of variables, X, is precluded from any global solution.*

Nogood are stored in a nogood cache, Γ, which records the infeasibility of particular variable assignments in any global solution. Now we can be more precise about which variable assignments are allowed in the current environment of other variables.

Definition 5. *An assignment, $\langle x = a \rangle$, is* disallowed *iff $\exists \lambda_\perp \in \Gamma$ s.t. $\langle x = a \rangle \in \lambda_\perp$ and $\lambda_\perp \setminus \{\langle x = a \rangle\}$ is valid. Otherwise, the assignment remains* allowed.

A variable assignment is disallowed if it is the *culprit* of a nogood whose remaining assignments constitute a valid label in the current environment. Contrary to usual practice[19, 6], the culprit in a nogood can be any variable in the label. To reiterate, we are concerned only with allowed/disallowed variable assignments whether or not these assignments are consistent/inconsistent under the constraints on these variables.

Definition 6. *The* live domain Δ_x *of a variable x is its allowed elements*

$$\Delta_x = \{a \in D_x \mid \langle x = a \rangle \text{ is allowed } \}$$

We use a heuristic valuation function to choose among the elements in the live domain of a variable.

Definition 7. *The function, $f(a)$, of a particular variable assignment, $\langle x = a \rangle$, defines its* heuristic valuation *based on the constraints on x and the current environment of other variable assignments.*

These valuations will be used to determine which domain elements constitute maximal assignments for x.

Definition 8. *$\langle x = a \rangle$ is a* maximal assignment *iff $\forall b \in D_x, f(a) \leqslant_s f(b)$.*

If every variable obtains a maximal assignment, then the label, $\{\langle x = a \rangle\}_{x \in V}$, is a *maximal solution.* No variable prefers a different assignment in the environment of the other variables. However, this maximal solution may not satisfy all the constraints in C thereby inducing a nogood on these constraints.

3 Informal Synthesis

Instead of viewing backtrack search as tree search, we consider that the current state of the search divides the variables, $\{x_1, \ldots, x_n\}$, into three (possibly empty) classes:

1. Those variables, $\{x_1, \ldots, x_{i-1}\}$, which comprise the current consistent partial solution, called the `CONSISTENT` variables. The assignment of each of these variables satisfies all the CSP constraints on this class.
2. The variable, x_i, at the fringe of the search identified as the *culprit* for backtracking is in state `INF`. Its current assignment is known to be infeasible given the environment of the `CONSISTENT` variables.[1]
3. The remaining variables, $\{x_{i+1}, \ldots, x_n\}$, are the future variables which remain unassigned and hence have state `NULL`.

Figure 1 shows these classes with the fringe of the search proceeding from left to right according to the assigned (possibly dynamic) ordering of the CSP variables. A solution has been found when every CSP variable is `CONSISTENT` and both the `INF` and `NULL` classes are empty. This simple model applies to various traditional backtracking schemes. In chronological backtracking, any inconsistency in the partial solution causes the culprit to be the `CONSISTENT` variable with the lowest precedence. In backjumping [5] and related intelligent backtracking schemes [19], when the culprit is identified within the `CONSISTENT` variables, then every other variable in this class but higher in the ordering is returned to the unassigned `NULL` category.

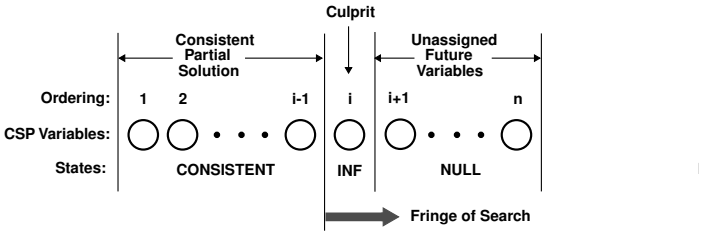

Fig. 1. The three possible states for backtrack search algorithms

But constructive algorithms will tour a significant fraction of the exponential search space if allowed, even when constraint propagation techniques [12] are employed. Instead practitioners rely upon variable and value ordering heuristics to improve the search efficiency. Heuristics are most informed when every variable

[1] The `INF` class is empty as search proceeds forward but instantiated during backtracking

in the CSP is assigned and least effective when no variables are assigned. In constructive search, the latter case tragically corresponds to the first variable in the search tree. Early mistakes in constructive search are exponentially expensive but must be made with the least available information for heuristic choice. The very mechanism of backtracking is self defeating by being hostile to the use of effective heuristics to guide the search. What we desire is a new synthesis incorporating the systematicity of constructive search with the heuristic effectiveness of local search.

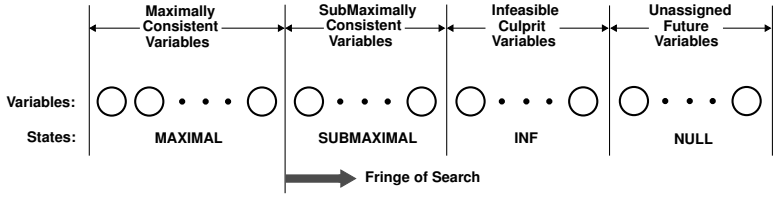

Fig. 2. The four possible states for maximal constraint solving

In our MCS schema, instead of maintaining partial consistent assignment, we allow variable assignments that are inconsistent. The variable states for backtrack search identified previously in Figure 1 are insufficient. We require a distinction be made in Figure 1 between variables assigned maximal values and those remaining assigned variables which can choose (potentially) better allowed assignments from their live domains. Thus the class `CONSISTENT` is divided into two subclasses `MAXIMAL` and `SUBMAXIMAL` as shown in Figure 2. Variables can belong to one of four possible classes given the constraints on their variables and the nogoods which have been induced on their assignments.

1. A variable is `MAXIMAL` when its assignment is both allowed and maximal. The variable prefers the current assignment from its live domain.
2. If a variable assignment is feasible but another better (more maximal) assignment is possible given then the variable is `SUBMAXIMAL`.
3. The variable is in an infeasible state, `INF`, if it is disallowed, i.e. if the current assignment is known nogood in the current environment of other variable assignments. It must change its assignment if possible but induce a nogood otherwise.
4. Otherwise, the variable is in the `NULL` state. Either it can choose a new `MAXIMAL` assignment from its live domain or it must induce a new nogood on the environment of other variable assignments which have emptied its domain.

```
1   function solve(V, C) {
2       α = {⟨x = null⟩}_{x∈V}
3       repeat {
4           while (α ≠ MAXIMAL) {
5               let x = select(V);
6               assign(x);
7               if (λ_nullset ∈ Γ) return noSolution;
8           }
9           ∀c ∈ C s.t. c is inconsistent {
10              λ⊥ = label(c);
11              add λ⊥ to Γ;
12          }
13      } until (α = MAXIMAL);
14      return α;
15  }
```

Fig. 3. The *solve* algorithm finds a consistent solution of the variables V for the constraints C.

4 Systematic Local Search

In this section, we develop our approach more precisely.

4.1 Maximal Constraint Solving

Our MCS schema searches heuristically through a space of maximally consistent variable assignments while backtracking on assignments which are not acceptable solutions. It discards the maintenance of a totally consistent partial solution in backtrack search [4]. Instead, we require that every variable in the partial solution be assigned a value which is both *allowed* and *maximal*.

Thus, it combines desirable aspects of systematic backtrack search and heuristic local search in the same asynchronous schema [14, 21]. Variables use value ordering heuristics to choose a *maximal assignment* from their live domain of possible values. If no allowed values remain for some variable then that variable induces a nogood and backtracks. When all variables have chosen a maximal assignment then these assignments constitute a *maximal solution*. Such a solution is a mutual local maxima for every variable. If the maximal solution does not exhibit full consistency then the solution induces nogoods and again the system backtracks.

Our systematic local search algorithm is listed in Figure 3. Given a set of variables, V, and constraints, C, $solve(V, C)$ returns the first solution, α, whose assignments satisfy each constraint in C. The algorithm operates as follows. In the initial global assignment, α, (line 2) every variable is `NULL`. The loop beginning in line 3 is repeated until every variable assignment in α is in class `MAXIMAL` (in line 13). Then the solution, α, is returned in line 14. While every

variable is not MAXIMAL (line 4), a variable, x, is chosen via the variable ordering heuristic, $select(V)$.[2] Then the variable, x, is assigned according to the method described in the next subsection.However, if an *empty nogood*, $\lambda_{nullset}$, is ever derived (line 7) then the algorithm returns failure indicating that no solution exists.

When every variable is MAXIMAL, the current global assignment, α, is a maximal solution but may not be a consistent solution. Beginning in line 9, for every constraint, $c \in C$, which is not satisfied, a new nogood, $\lambda_{\perp}$, is derived from c and added to the nogood cache, Γ, in line 11.[3]

4.2 Variable Assignment

We define a method, $assign(x)$, which specifies the action taken by variable, x, given the current state of that variable (according to the classification of Figure 2). The semantics of this method are given in the table of Figure 4 as state transition rules.

State	Condition	Action
MAXIMAL	<x=a>, a $\in \Delta_x$, not $\exists b \in D_x$ s.t. f(b) < f(a)	NOP
SUBMAXIMAL	<x=a>, a $\in \Delta_x$, $\exists b \in D_x$ s.t. f(b) < f(a)	choose a new maximal state
INF	<x=a>, a $\in \Delta_x$,	if $\Delta_x \neq \emptyset$ then choose a new maximal state else environment λ_x is nogood assign <x=null>
NULL	<x=null>	if $\Delta_x \neq \emptyset$ then choose a new maximal state else NOP

Fig. 4. Possible states of a variable, x, and consequent action given the current global assignment and live domain of variable, x, used by the method, $assign(x)$.

Variable x is in a MAXIMAL state if it is assigned $\langle x = a \rangle$ and $f(a)$ is maximal. No action is required (NOP) since the variable cannot possibly find a better (more maximal) state. Similarly, if x is in a SUBMAXIMAL state when assigned but another element, $b \in \Delta_x$, exists such that $f(b) \leqslant f(a)$. Then the action is to choose $\langle x = b \rangle$ as a new MAXIMAL state for x according to the minconflicts heuristic. However, x is in the INF state if assigned $\langle x = a \rangle$ but the value $a \notin \Delta_x$.

[2] Variable ordering is defined immediately below.

[3] In practice, without an infinite nogood store, the algorithm stills remains incomplete since it does not systematically cover the entire search space.

Again the appropriate action depends on whether other elements remain in Δ_x. If so then choose some new assignment and enter the `MAXIMAL` state. Otherwise, report that λ_x is nogood and undo the current assignment by assigning $\langle x = null \rangle$ thus entering the `NULL` state.

Finally, variable x is in the `NULL` state whenever unassigned. Initially, all (future) variables are `NULL` and variables return to this state whenever they are assigned $\langle x = null \rangle$. The appropriate action depends on whether other elements remain in Δ_x. If so then choose some new assignment and enter the `MAXIMAL` state. Else, do nothing (NOP) until the environment, λ_x, of other variables constraining x changes forcing x into some other state.

Nogoods force the variables away from maximal but unacceptable solutions to the CSP. It is analogous to other digression mechanisms in local search such as *tabu search*[8] where the tabu list records local maxima which cannot be revisited for some period. The difference here is that our digression mechanism is systematic. It inherits the sophisticated logical machinery of intelligent backtracking but operates within the local search paradigm.

The instance of the MCS schema presented here, *MinCon-CBJ*, uses *min-conflicts* which chooses assignments which violate the minimum number of constraints on x given the current state of the search. It is known that good value ordering heuristics can greatly improve the efficiency of both constructive and local search algorithms [20]. In general, we assume that effective heuristics are problem specific and hence a parameter of the schema. The valuation function can be redefined to compute something different than constraint violations.

Definition 9. *The* heuristic valuation*, $f(a)$, for each $a \in D_x$, is the number of constraints that disallow the variable assignment $\langle x = a \rangle$ in the current environment.*

The constraint violations are computed based on forward checking propagation. We maintain conflict counts for all variables, so that we can always ask the question how many constraints will be violated if a particular assignment were made in the current environment, similarly to Prestwich [16]

4.3 Variable Ordering

Variable ordering, $select(V)$, determines the overall control structure in our approach. From Figure 2, we are reminded that variables assume four different classes corresponding to their current assignments and live domains. Thus we select variable execution order based on these classes. We define a total order among the variables by first choosing precedence among the categories `SUBMAXIMAL`, `INF` and `NULL`. We do not include the `MAXIMAL` category because executing a maximal variable does not have any effect. In addition, we choose a separate strategy within each class that imposes a total order. Both of the ordering among the classes and the class strategies would influence the behaviour of the algorithm. We always choose to execute the variable lowest in the resutling total ordering.

For the *MinCon-CBJ* we experimented with two different versions of the variable ordering heuristic method, $select(V)$:

1. *minCon-Heuristic* implements the precedence ordering, `NULL` $\prec$ `SUBMAXIMAL` $\prec$ `INF`, which says that we prefer a full assignment of variables and then move heuristically to make all submaximal assignments maximal before backtracking any infeasible variables. Within the `NULL` class, variables are ordered lexicographically. `SUBMAXIMAL` variables are ordered by maximum possible improvement, such as [4]. Variables in the `INF` class are not ordered but chosen randomly. Basically, this ordering is aiming at maximizing the heuristic guidance. It resorts to dealing with `INF` only after all heuristically desirable decisions have been made.
2. *minCon-Systematic* implements the ordering, `INF` $\prec$ `NULL` $\prec$ `SUBMAXIMAL`, which attempts to resolve infeasible variables before pursuing a full assignment before repairing any submaximal assignments.
   This method resolves nogoods first trying to always maintain an allowed partial solution. Variables within class `INF` are chosen randomly as the nogood culprit, thus achieveming diversification. Like the above method, `NULL` variables are chosen for execution lexicographically and `SUBMAXIMAL` variables by maximum improvement.

There are other interesting possibilities such as giving the `SUBMAXIMAL` class highest precedence. Also we could decide to choose among the variables in the `SUBMAXIMAL` class the variable assignment with the worst valuation rather than with the best possible improvement. Or we could choose randomly among the `NULL` variables. The numerous different instances of the variable ordering based on the four classes we have defined are very interesting. Further investigation is needed to compare their performance and evaluate their behaviour.

5 Experimental Results

The experiments described in this section are very preliminary. We generated an ensemble of solvable random binary CSPs each having 20 variables and domain size 10 around the phase transition region for probabilities p1=0.5 and p2 varying from 0.32 to 0.42 in increments of 0.01. Following the advice of [3], we used complete search (FC-CBJ) [17] to identify 200 soluble instances for each p2 value. Our results are shown in Figure 5. As expected, the hardest instances for FC-CBJ where at approximately $\kappa = 0.95$ which required the most number of backtracks (assignment changes) to find a first solution. As κ increases, we found it more and more difficult to find soluble instances.

Our *MinCon-CBJ* method performed very well on this random ensemble upto and beyond the phase transition point using either the *minCon-Systematic* or the *minCon-Heuristic* variable ordering heuristics (control strategies). Indeed, both methods outperformed *FC-CBJ* significantly on smaller values of κ where the number of solutions to the CSP is expected to be large. Our method was approximately twice as efficient as *FC-CBJ* at the transition point. Beyond $\kappa =$

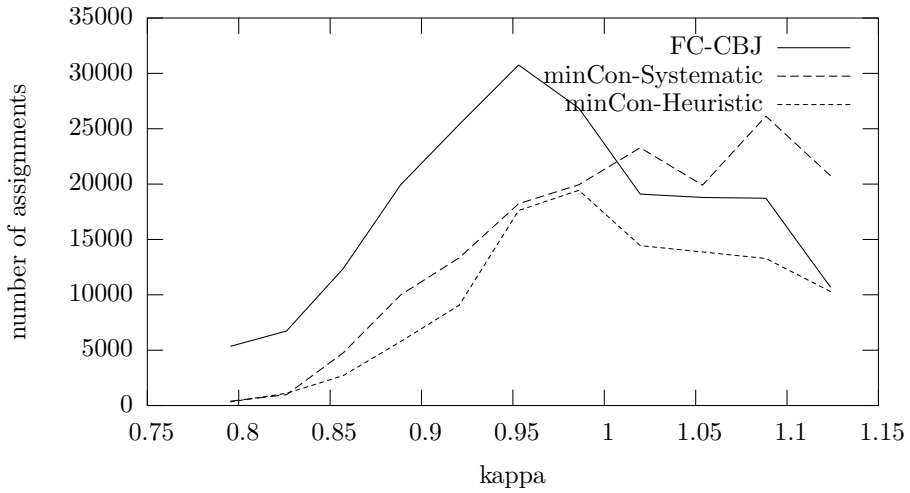

Fig. 5. Results comparing constraint-directed backjumping (FC-CBJ) against MinCon-CBJ using both minCon-Systematic and minCon-Heuristic variable ordering heuristics on random binary CSPs instances $\langle 20, 10, 0.5, p2 \rangle$

.

1.0, *MinCon-CBJ* using the *minCon-Heuristic* control strategy still significantly dominates the complete search method. We believe the superiour performance is due to the strategy of finding a full and maximal (but perhaps inconsistent) assignment before backtracking inconsistent constraints. However, *MinCon-CBJ* using the alternate *minCon-Systematic* control strategy loses its effectiveness for high values of κ. These overconstrained problems are easy for complete search which fails whole subtrees quickly. We conjecture that they are more difficult for our second control strategy because we are backtracking incompletely through a space of full but inconsistent assignments. For high values of κ, systematicity becomes more important, allowing the extremely tight constraints to prune the search quickly.

Obviously, many other search control structures can be realized by specifying different variable ordering strategies as above. We are currently investigating some promising variants.

6 Discussion

Efforts to synthesize constructive and local search have made good progress. Recent work has focused on breaking the tyranny of systematic backtracking while maintaining a consistent partial solution. Gomes *et al* [9] randomize the identification of the culprit variable for backtracking. Prestwich [16] forgoes completeness in an incomplete dynamic backtracking (IDB) algorithm which identifies

the culprit variable for backtracking heuristically (or randomly). IDB is viewed as a local search method which maintains a consistent partial assignment while attempting to maximize the number of assigned variables. Our scheme, similarly, utilizes the ability to move arbitrarily by choosing randomly or according to any other strategy among the `INF` variables. In addition, we retain the freedom of heuristic order of reassignment among the variables in the `SUBMAXIMAL` and `NULL` categories. Jussein and Lhomme [11] describe a hybrid framework which supports two types of moves: 1) extending a consistent partial solution which does not violate any cached nogood, otherwise; 2) repairing an inconsistent state by moving in a neighbourhood which does not violate the identified conflict nor the nogood cache. Various heuristic strategies are possible for either type of move. These two moves directly correspond to choosing among the `NULL` variables and choosing among the `INF` variables respectively, while the heuristic stratigies they discuss are equivalent to the variable orderings within these two classes used our framework.

In the instance of the MCS presented here we use *minconflicts*. Minton *et al.* [14] not only identified this natural value ordering heuristic for CSPs but developed a constructive search algorithm, called *informed backtracking*, which worked simultaneously from a consistent partial solution and total inconsistent assignment. The latter is used by minconflicts to choose value assignments during backtracking. But perhaps the most influential synthesis has been the work of Freuder and Wallace [4] which abandoned the maintenance of a consistent partial solution for soft CSPs. Their method substitutes *branch&bound* techniques for backtracking in a variety of well known constructive algorithms. They also identify the notion of a maximal solution for soft CSPs. By subsitituting the notion of a maximal assignment for consistency in partial solutions, we obtain the power of working from a full assignment similarly to Freuder and Wallace [4]. Thus value ordering heuristics have access to an artifact with as much information as possible to guide the search. While here we are presenting an algorithm for solving relational CSPs, it is easy to see possible extensions of the MCS schema to soft CSPs. Every time we reach a maximal solution, we could remember the one with best quality and always fail the whole solution thus allowing each individual constraint to be inconsistent in future solutions but not the exact same combination.

Another important and relevant research issue is the tenure policy for nogoods in the cache. In an insightful short paper, Ginsberg and McAllester [7] suggested that constructive and local search differ mainly in the memory (nogood cache) used to remember past states (failures). For systematic search algorithms the cache size can be capped. For examples, chronological backtracking requires only $O[n]$ space to remember past variable assignments but has an extremely rigid control structure. Less rigid is dynamic backtracking [6] requiring only $O[nd]$ nogood space but only slightly relaxing the systematicity of the search. Clearly, for purely local search which always follows the heuristic gradient, the number of nogoods induced can approach $O[d^n]$. However, Lynce *et al* [13], who describe a hybrid algorithm employing randomized backtracking for solv-

ing hard boolean satisfiability problems, claim this is not a problem in practice. Havens [10] shows empirical evidence that bounded nogood caches work well in distributed backtracking. Otherwise, completeness must be forsaken. In [11], a finite cache in the form of a Tabu [8] is used. Whereas [16] remembers no previous states at all which sometimes leads to deadlock. We seek to explore the middle ground of currently unknown search algorithms which maximize the use of good heuristics to guide the search while retaining only enough systematicity to limit the required size of the nogood cache. Similarly to the hybrid framework presented by Jussein and Lhomme [11], the MCS schema is parametrized and can support different nogood tenure policies. We believe that polynomial space caching schemes such as *k-relevance learning* [1] will provide the right balance between the maximum size of the cache and the ability to move freely in the search space.

7 Conclusion

We presented a new hybrid constraint solving schema which retains some systematicity of constructive search by remembering nogoods and backtracking while incorporating the heuristic guidance of local search by working on a complete and maximally consistent solution. We divide the CSP variables into four classes and define an asynchronous variable execution model that operates on these states. Various search algorithms can then be defined parametrically by specifying different variable ordering heuristics for each class allowing for arbitrary changes in variable assignment and hence freedom to move. Finally, we gave two instances of our scheme, both of which are new algorithms combining desirable aspects of both constructive and iterative search. Some interesting experimental results were presented which suggest that these methods work well. We are currently investigating additional instances and caching policies. Future work involves the extension of this framework to soft and valued CSPs using the cSemiring formalism [2] and replacing the notion of a consistent solution with that of an acceptable solution.

References

1. R.Bayardo and D.Miranker (1996) A Complexity Analysis of Space-Bounded Learning Algorithms for the Constraint Satisfaction Problem, in *proc. AAAI-96*, pp.298-304.
2. S.Bistarelli, U.Montanari and F.Rossi (1995) Constraint Solving over Semirings, in *proc. IJCAI-95*, Morgan-Kaufman.
3. D.A.Clark, J.Frank, I.P.Gent, E.MacIntyre, N.Tomov and T.Walsh (1996) Local Search and the Number of Solutions, in *proc. Principles and Practice of Constraint Programming*, pp.119-133.
4. E.C. Freuder and R.J. Wallace (1992) Partial Constraint Satisfaction, *Artificial Intelligence* 58:21-70
5. J. Gaschnig (1979) Performance measurement and analysis of certain search algorithms. Tech. report CMU-CS-79-124, Carnegie-Mellon University

6. M.L. Ginsberg (1993) Dynamic backtracking. *Journal of Aritificial Intelligence Research* 1:25-46
7. M.L. Ginsberg and D.A. McAllester (1994) GSAT and dynamic backtracking. in *proc. 4th Int. Conf. on Principles of Knowledge Representation and Reasoning*
8. F.Glover (1990) Tabu search: a tutorial, *Interfaces 20*(74-94).
9. C. Gomes, B. Selman and H. Kautz (1998) Boosting Combinatorial Search through Randomization, *proc. Fifteenth National Conference on Artificial Intelligence*, pp.431-437, Madison, WI, AAAI Press.
10. W.S.Havens (1997) NoGood Caching for MultiAgent Backtrack Search, in *proc. AAAI-97 Constraints and Agents Workshop.*
11. N. Jussein and O. Lhomme (2002) Local search with constraint propagation and conflict-based heuristics, *Artificial Intelligence* 139:21-45.
12. V. Kumar (1992) Algorithms for constraint satisfaction problems - a survey, *AI Magazine 13*(32-44)
13. I. Lynce, L. Baptista and J. P. Marques-Silva (2001) Stochastic Systematic Search Algorithms for Satisfiability, *LICS Workshop on Theory and Applications of Satisfiability Testing (LICS-SAT)*, June 2001.
14. S. Minton, M. Johnston, A. Phillips and P. Laird (1990) Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58:161-205
15. P. Morris (1993) The breakout method for escaping from local minima. in *proc. AAAI-93* pp.40-45
16. S. Prestwich (2001) Local Search and Backtracking vs Non-Systematic Backtracking, *proc. AAAI 2001 Fall Symposium on Using Uncertainty within Computation.*
17. P.Prosser (199x) Hybrid Algorithms for the Constraint Satisfaction Problem, *Computational Intelligence 9(3)*:269-xxx
18. B.Selman, H.Levesque and D.Mitchell (1992) A new method for solving hard satisfiability problems, in *proc. 10th National Conf. on Artificial Intelligence,*pp.440-446.
19. R.M. Stallman and G.J. Sussman (1977) Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9:135-196
20. E.Tsang (1994) *Foundations of Constraint Satisfaction*, Academic Press.
21. M.Yokoo (1994) Weak commitment search for solving constraint satisfaction problems, in *proc. AAAI-94*, pp.313-318.