Exploiting Solution Clusters for Coarse-Grained Distributed Search

Andrew J. Parkes

CIRL
1269 University of Oregon
Eugene OR 97403–1269
USA
http://www.cirl.uoregon.edu/parkes

ABSTRACT

We consider distributed environments with high communication costs and complex local problems and argue that search can benefit from the exchange of messages containing relatively deep semantic information about the structure of the problems at each node. Specifically, we consider the case that messages contain compact representations of large clusters of solutions from each node. The intention being to weaken the constraints passed between nodes and so reduce the chance of failure in the receiving nodes.

We make some first steps towards such a distributed search method. We define clusters as appropriate for both sequential and distributed problems. Finding optimal (largest possible) clusters turns out to be NP-hard. However, optimality is not necessary for the search, and so we give some heuristic constructive algorithms for producing clusters of varying sizes. Initial experimental studies on the utility of clusters are made in a distributed domain based on random SAT problems. We find evidence that the use of clusters has the potential to greatly reduce the chance of the distributed search failing unnecessarily.

1. INTRODUCTION

Initial work on solving distributed CSPs (see [19] and others) had a very fine-grained distribution, for example, just one variable per node. This simplified algorithm development, however, in practice, systems often have a much coarser grained distribution. Indeed, Yokoo has extended some of the earlier work to the case of "complex local problems" [18].

In this paper we investigate the possibility that conversion from simple to complex local problems should also be accompanied by a change from simple to complex messages between nodes. In fine-grained theories the usual communication is just that of (speculative) assignments to variables (and "no-good" combinations there-of). When we have complex non-trivial local problems we consider the possibility of messages corresponding to entire sets of solutions to the local problems.

We are motivated by work on solutions clusters in phase transitions of complex systems [10, 11, and many others]. Consider systems that are close to optimality in the sense of being close to a phase transition between satisfiability and unsatisfiability. Naive intuition might suggest that the number of solutions will be very small, perhaps O(1), in such cases, however it appears that in many systems there are often an exponential number of solutions occurring within a relatively small region of the search space: a "solution cluster". That is, even optimal solutions of problems can still have a large amount of freedom.

The question we raise is whether distributed search can benefit by having the complex local nodes find clusters of their local problems and then communicate these clusters to other nodes. If we think of a cluster as an exponential number of solutions to its local problem then this clearly has the potential to reduce the amount of global backtracking: maybe one of the solutions will work for other nodes, even if a randomly selected single solution would probably not work.

A natural objection to such a scheme is that transmitting an exponential number of local solutions will be too expensive. However, an important aspect of the phase transition cluster work is that the cluster has a very compact description. At a high-level we will regard a cluster as a region of the search space that is rich in solutions (models) and that admits a simple compact description of the region and its set of models. It is important to note that we do not require that all of the assignments within the region are solutions, merely that it is model rich – this will allow for a much greater flexibility of choice of cluster, and communication of a larger number of models in a single message.

Overall, there is no reason for communication of clusters to be much more expensive than for usual messages. Instead the first issue is whether or not it can indeed help the overall search. We will make a preliminary attack upon this general question, though admit in advance that our results are suggestive only and not conclusive as to whether cluster-passing will be useful to real systems. However, our expectation is that in environments with high communication costs (such as the web?) we should generally favor deeper intra-node search over inter-node search, and so will want to pass messages based on a deeper understanding of the local node.

In Section 2 we formulate relevant notions, definitions, and associated computational complexities, of clusters for non-distributed (monolithic) theories, discuss how distributed problems with (complex) local nodes affect clusters, and in particular extend cluster notions to deal with "interfaces" between local nodes. Section 3 gives some simple greedy search methods to find appropriate clusters. In Section 4 we consider a simple form of distributed search and give some evidence that clusters can reduce the failure rates for the passage of tentative solutions between nodes. We also see how the effectiveness of various kinds of clusters varies with respect the the constrainedness of the local nodes, and their proximity to their own phase transitions. In Section 6 we return to a general discussion of issues raised, and the many possibilities for future work.

2. CLUSTER DEFINITIONS

In this section we review and extend the relevant terminology and definitions, and give some computational complexity results.

Firstly, we introduce some basic idea of clusters for the case of a single theory. We then look at how clusters will relate to a coarse-grained distributed search in which the variables at a node are not all "public". Finally, we extend appropriate cluster definitions to handle this form of distributed search.

2.1. CLUSTERS FOR A MONOLITHIC THEORY

Firstly, we address the non-distributed case; we consider a single constraint theory Γ. Here we take Γ to be a SATISFIABILITY instance in CNF, that is, a conjunct of clauses over a set of propositional variables. Although we focus on SAT, many of the definitions and results of the paper can be expected to apply directly to other systems such as constraint satisfaction problems or mathematical programming representations.

We identify a cluster with an associated "**cluster-defining partial assignment**," (CPA). The CPA associated with a cluster of models being the set of assignments they have in common.

We refer to variables not valued by the CPA as the residual variables; they are just the variables that distinguish between different members of the cluster. Each such CPA has an associated "residual theory", $\Gamma_R(CPA)$, defined by

$$\Gamma_R(CPA) = \Gamma|_{CPA} \tag{1}$$

that is, the theory after enforcing the literals of the CPA and simplifying. Solutions of the residual theory are then, by definition, the models forming the cluster itself. For non-triviality we also require the CPA to contain at least one model

$$CPA \wedge \Gamma \not\models \perp \tag{2}$$

It is natural to also impose a maximality condition on the CPA: any extension of the CPA should result in the loss of models from the cluster. That is, we require

$$\not\exists x \in \Gamma_R(CPA). \quad CPA \wedge \Gamma \models x \tag{3}$$

The CPA represents a "shrink-wrapping of the models of the cluster."

Finally, in practice, we will also want to demand that a cluster be "model-rich", and will do this be requiring that the residual theory be "under-constrained". The intention is that solving the residual theory is easy. If the residual is not under-constrained then we would expect to break up the cluster into many different clusters, and do so until each is under-constrained.

There are many possible ways one might try to define under-constrained, for example

1. A lower bound on the density of models in the residual theory. This has the practical problem that counting of models is #-P hard and makes the decision as to whether a CPA is acceptable unnecessarily difficult. Generally, we also want that the recognition problem for a CPA should be polytime.

2. An upper bound on the clause/variable ratio of the residual theory. This has the advantage of being trivial to measure and will be used in some of our algorithms.

3. Tractable residual theory. One might require that the residual theory be solvable in polytime, for example, 2-SAT or HORN-SAT. This is probably too strong a requirement, as we will typically just want that the residual theory be solvable quickly on average.

In general, we would expect that the decision as to what counts as under-constrained would be domain dependent and be made heuristically. Hopefully, if clusters are useful in practice then their utility would not be too sensitive to the exact definitions chosen.

Also, these definitions are meant to be guidelines rather than fixed. For example, it might be quite reasonable to replace the maximality requirement (3) with a more tractable condition such as

$$\not\exists\, x \in \Gamma_R(CPA). \quad CPA \wedge \Gamma \vdash_P x \tag{4}$$

where $\vdash_P$ is a polytime propagation procedure.

Note that if Γ has only one cluster then its CPA becomes the set of unary prime implicates[1] (UPIS), and is sometimes referred to as the backbone [10], or maybe as the set of frozen variables.

In the case that there are no residual constraints, $\Gamma_R(PA) = \texttt{true}$, we will say that the cluster is "pure", because then the cluster contains nothing but models. If the cluster contains non-models as well as models we will say that it is impure. We make the distinction because of the impact it has on the kinds of messages that will need to be transmitted during distributed search: with pure clusters we only need to transmit partial assignments and not also (residual) constraints.

Note that if a cluster is pure then the associated CPA is automatically maximal.

[1] We call x an implicate iff $\Gamma \models x$.

To investigate the complexity of finding clusters we formalize an associated decision problem (and temporarily drop the ill-specified requirement of "under-constrained residual):

Definition 2.1 CLUSTER

INSTANCE: Constraints Γ*, integer* K.

QUESTION: Is there a cluster of Γ*, and associated CPA, such that* $|CPA| \leq K$*? (That is, a cluster with at least* $n - K$ *residual variables.)*

Theorem 2.2 CLUSTER(MODEL) *is NP-complete*

Proof. Checking the length of the CPA is in polytime. Polytime checking of the maximality requirement on the CPA is equivalent to checking that the associated residual theory has no unary implicates. This can be achieved by taking the witness to a "yes" answer to be not just the CPA but also sufficient models from the residual theory to exclude any extensions fo the CPA. Specifically, the CPA is maximal (inextensible) iff for each literal l of the residual theory there is some model of the residual that contains l. Hence, we need to non-deterministically produce at most $2(n-K)$ additional models in order to exclude extension of the CPA. Hence the problem is in NP.

NP-hardness follows from the fact that the special case of $K = n$ just requires showing that some model exists, and hence is just SAT. ■

We just saw that CLUSTER is NP-hard, but maybe it is only hard because it contains the SAT problem of finding an initial model? Maybe the problem becomes easier if we have already found a model?

Suppose have an initial satisfying total assignment, or "reference model," m, and want to expand it to a cluster, that is with $CPA \subseteq m$.

Definition 2.3 PURE-CLUSTER(MODEL)

INSTANCE: Constraints Γ*, reference model* m*, integer* K

QUESTION: Is there a pure cluster of Γ*, and associated CPA, with* $|CPA| \leq K$*, and such that* m *is in the cluster?*

Theorem 2.4 PURE-CLUSTER(MODEL) *is NP-complete*

Proof.

Checking that the model m is in the cluster just means checking that the literals of the CPA are a subset of those of m, and hence is trivially polytime. Otherwise the problem is in NP for the same reasons as CLUSTER.

To see NP-hardness we reduce from the the NP-complete HITTING SET. From Garey+Johnson [5]:

Definition 2.5 HITTING SET

INSTANCE: Collection C of subsets of a finite set S, positive integer $K_H \leq |C|$

QUESTION: Is there a subset $S' \subseteq S$ *with* $|S'| \leq K_H$ *such that* S' *contains at least one element from each subset in* C*?*

Take S to be the set of positive literals, for each set $c \in C$ form a clause from the disjunct of its members. The subset S' then is the CPA. Furthermore, since S' hits each set it follows that the CPA satisfies every clause. That is the associated cluster is pure, and hence automatically maximal. The reference model is just the case that all variables are set to true. ■

Since the reduction produced only clauses with positive literals it follows that PURE-CLUSTER(MODEL) remains NP-complete in that case. Furthermore, HITTING SET remains NP-complete even if $|c| \leq 2$ for all $c \in C$, and so the problem remains NP-hard even for 2-SAT and with no negative literals.

One might also introduce variations in which we also impose a bound on the number of the clauses in the residual theory. (The rationale for this being that in a distributed system we do not want to have to communicate large numbers of clauses; both because of communications overhead, and because of the impact this might have on the receiving nodes). For example,

Definition 2.6 BOUNDED-CLUSTER(MODEL)

INSTANCE: Constraints Γ*, reference model* m*, integer* K*, integer* B.

QUESTION: Is there a cluster of Γ*, containing* m*, and with associated CPA having* $|CPA| \leq K$*, and also with the restriction the residual theory contains at most B clauses?*

This is NP-complete because the case $B = 0$ is just PURE-CLUSTER(MODEL)[2].

Hence, and perhaps surprisingly, finding a candidate CPA for a model cluster of a given required size can be NP-hard even if we are given a reference model.

However, it is important to note that the NP-hardness refers to finding optimal size clusters, and there is no requirement for the distributed search to find such largest clusters. Also, the number of the variables might be much less than the size of the initial theory. For example, the search for a cluster might only be exponential in the number of residual variables and hence manageable. Hence, as is common with NP-hardness results, the purpose of the proofs is merely to justify our use in later sections of search based algorithms such as greedy heuristic algorithms, and the usage of suboptimal size clusters.

Generally, we are looking for compact representations of a large number of models. We can, in principle, obtain an even more compact representation by using a "generator of a CPA", or G(CPA), meaning any subset of the literals of the CPA such that

$$G(CPA) \ \wedge \ \Gamma \models CPA \tag{5}$$

It would also be reasonable to encourage short generators by imposing conditions such as

- minimal G(CPA): cannot remove any single literal and retain the property (5).
- minimum G(CPA): the shortest possible G(CPA).

Conversely, in practice, the full entailment will be too difficult to check, and so we might weaken (5) to

$$G_p(CPA) \ \wedge \ \Gamma \vdash_P \ CPA \tag{6}$$

where $\vdash_P$ is some polytime propagation procedure.

[2]We do not know as yet the complexity if the size constraint is removed.

Although we do not use the G(CPA) in this paper we expect it could have a role in creation of no-goods (see Section 6). Also, if the G(CPA) is much shorter than the CPA then this is presumably an indication that the cluster is relatively well separated from other clusters. One potential usage of such information would be for the nodes to be able to supply a wide range of very different solution clusters: if a previous cluster caused a failure in some other node, then maybe it is better for the overall search that the node tries to provide a quite different solution.

In the case that the CPA contains all the models of Γ and hence is a backbone, then maybe it is reasonable to call the associated generator G(CPA) a "neck"; with full entailment the neck will be empty (because the backbone is entailed from Γ itself) but with just propagation we might well need to supply a few variables for the propagation to get started. Experiments, on Random 3SAT and not reported here, suggest that in such cases the neck can be much shorter than the backbone.

2.2. DISTRIBUTED SOLVERS AND INTERFACE VARIABLES

Suppose that we have a coarse-grained distribution in which the local problems at each node are non-trivial. In this case it becomes quite reasonable that not all the variables at a node are directly involved in constraints with variables at other nodes or groups of nodes. Typically, a node will respond to inputs from other nodes, perform an internal calculation and then broadcast derived relevant assignments to other nodes. We will model such an input/output behavior in terms of interfaces. In particular we will be focusing on the case in which the broadcast of the results of the internal calculation do not involve all the variables, but only a subset that we call the public variables of an outgoing interface.

For simplicity, we have focussed on simple distributed systems with a chain like structure as illustrated in Figure 1. Such a restriction allowed us to perform some initial experiments, see Section 4, to show cases in which passing of clusters does have the potential to help search. An example of such a chain might be a scheduling problem in which computation is dividing between processors based on the (expected) start times of tasks. Another example might be that of a logistics problem with a geographic partition of the problem: for example, a partition of the USA into western, mid, and eastern regions. Of course, we hope to be able to extend to more complex graphs in future work.

In an environment with high communication costs, and assuming that the distribution is not entirely predetermined by other requirements, then we can expect that a good distribution will also tend to reduce the communication links. Hence, good distributions are likely to encourage the outgoing interfaces to contain a small fraction of the variables of a theory.

2.3. CLUSTERS AND PROJECTIONS ONTO INTERFACES

We now have a separate local theory for each node. Furthermore, outgoing communications are in terms of public variables, with many of the local variables being private in the sense that no outside node directly needs to know their value. If we are to communicate a cluster to other nodes it hence follows that we first need to project it onto the relevant (outgoing) interfaces. In the section we discuss the most immediate options. Also, given all the issues of projection, it is not clear what definitions of clusters (if any) are going to be best for distributed systems, so for this paper we will just take rather direct extensions of those for the monolithic case.

We also restrict ourselves here to the case of a single outgoing interface. Hence, we split the variables into public variables x and private variables y, and write the overall constraints as $\Gamma(x, y)$. See Figure 2 for an informal illustration of the relevant variables.

By an output cluster we will mean a theory

$$\gamma(x) \quad = \quad CPA(x) \ \wedge \ \gamma_R(x) \tag{7}$$

consisting of a CPA together with a residual theory but only involving the public variables.

We will impose the natural requirement that any model of the output cluster is extensible to a model of the entire local theory:

$$\forall \ x_1. \ \ (x_1 \models \gamma) \ \longrightarrow \ \exists y_1 \ \ \Gamma(x_1, y_1) \tag{8}$$

However, just as for clusters of monolithic theories there are still many choices remaining. The main issue is the order in which we do the cluster generation and projection.

Firstly, we consider "cluster+project", that is we generate a cluster of the full theory and then project. For the CPA we can just restrict it to the public variables

$$CPA^P = CPA|_x \tag{9}$$

However, we cannot naively project the residual theory because it can contain private variables that will create implicit constraints between the public variables, Instead the projected residual arises by using logical equivalence after enforcing all the literals of the CPA

$$\gamma_R^P(x) \ \longleftrightarrow \ \exists \ y_1^R. \ \ \Gamma(x, y)|_{CPA} \tag{10}$$

Alternatively, we can "project+cluster". We project the entire theory onto the public interface

$$\Gamma^P(x) \ \longleftrightarrow \ \exists \ y. \ \ \Gamma(x, y) \tag{11}$$

and then derive output clusters as clusters of the projected theory $\Gamma^P(x)$. In this case, neighboring models of $\Gamma^P(x)$ can arise from quite different values for the private y variables. Hence, the result of "project+cluster" approach can contain the combination of many different clusters as derived from a "cluster+project" approach. Generally, we can expect "cluster+project" to be easier, but to produce smaller clusters than "project+cluster". Possibly there are interesting or practical hybrids, but we do not explore such possibilities here.

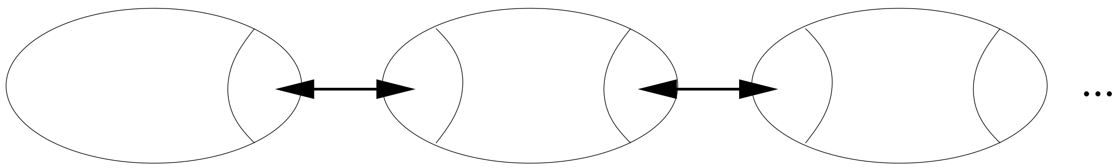

Figure 1: Interfaces for a simple chain of local nodes.

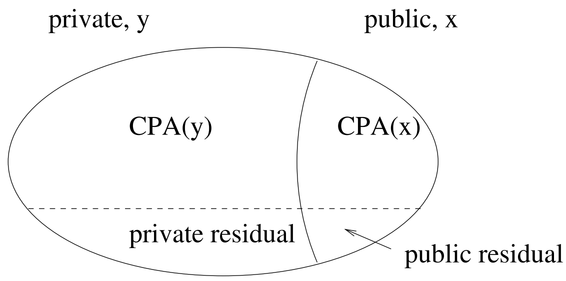

Figure 2: Schematic of the relations between public/private and CPA/residual for a single node, with single outgoing interface.

3. FINDING CLUSTERS

Given the results in the previous section about the NP-hardness of finding clusters it is clear that there are many opportunities for developing search methods to find them. However, our focus in this paper is the potential utility of clusters for distributed search, and so to make initial attempts to answer this we will just use some straightforward constructive methods, and have two basic choices:

- top-down : Procedure 3.1. We start from an empty assignment and then heuristically set variables until we determine we have obtained a cluster. Typically termination will be based on measuring the number of residual constraints in comparison to residual clauses. This has the danger that the literal selection could make a fatal error that eliminates all models, however this can be avoided by also using a pre-determined model in order to guarantee that at least that model will be in the cluster.

- bottom-up: Procedure 3.2. Starting from a model we de-value heuristically selected literals until terminating. After termination we propagate (or do full entailment) in order to guarantee the cluster is maximal.

Procedure 3.1 Top-Down
Input: Constraints Γ, (optional) Model m
  Partial Assignment P := Ø
    loop:
      l = select-literal(Γ,P) or select-literal(Γ,P,m)
      P := P + l
      P := propagate(Γ,P)   (return $\perp$ if detected)
      break if total-assignment(P) or terminate(P)
    end loop
  return P

Procedure 3.2 Bottom-Up
Input: Constraints Γ, Model m
  Partial Assignment P := m
    loop:
      l = select-literal(Γ,P,m)
      P := P - l
      break if P = Ø or terminate(P)
    end loop
  return propagate(Γ,P)

In these methods propagate(P) is a polytime propagation routine. In practice, for SAT, we use standard unit propagation $\vdash_P$ augmented with a standard failed literal test[3] [3]. For the size and complexity of clusters that we encounter this turns out to often coincide with full entailment. Note that if we are using values derived from a model m then if propagate(P) values, or re-values, any variables then it will set it to the same value as in m: If $l \in m$ is in the original model then m excludes $\neg l$ being entailed.

3.1. CLUSTER AND PROJECT

Firstly, we look at finding clusters of the entire local theory before projection onto the interface. We use the generic bottom up method of Procedure 3.2 and to instantiate it we need to specify select-literal(l) and the termination requirement terminate(P).

- termination: we require that the residual clause/variable ratio stays below 2.0; this is somewhat ad hoc but does give an "under-constrained residual."

[3] If there exists a literal l such that $\Gamma \wedge l \vdash_P \perp$ then enforce $\neg l$

- literal selection: we give two methods for unsetting variables. The first does just unsetting of what we call inexact variables, the second method follows this up with further heuristic unsetting of variables.

Firstly, we define what we mean by exact and inexact variables:

Definition 3.3 *"Exactness": A variable x is said to be exact with respect to a given satisfying assignment iff it is the sole satisfying literal in at least one clause.*

We take the name from the NP-complete problem "Exact 3-SAT" in which a clause is considered satisfied iff precisely one literal is true [5].

By construction, if we pick on any single inexact variable then we can unset it and will still have a satisfying (partial) assignment. If we try unsetting two or more inexact variables then we can no longer guarantee retaining a satisfying assignment, as, for example, the only satisfying literals in a clause might both be inexact. However, inexact variables are still good candidates for unsetting to generate a cluster.

Hence, in the first algorithm, Procedure 3.5, we use Procedure 3.4 to detect and unset all inexact variables This is repeated until no further exacts are found, or until the termination condition is reached. We then do propagation in order to satisfy the cluster maximality.

Procedure 3.4 Find-Inexacts
Input: Set of Clauses $\{c_i\}$, partial assignment P
  foreach variable x Inexact(x):=true
  foreach clause c_i
    if c_i has a sole satisfying literal y or $\neg y$
      then Inexact(y) := false
return { y | Inexact(y)=true }

Procedure 3.5 Bottom-Up-Inexacts
Input: Constraints Γ, Model m
  Partial Assignment P := m
    loop:
      I = Find-Inexacts(Γ,P)
      break if I = Ø or terminate(P)
      P := P - I
    end loop
  return propagate(Γ,P)

In the second method, Procedure 3.6, we also unset inexacts but also follow up with a second phase of unsetting of selected literals. Literals are selected based upon the size of the cluster that would remain if they were to be unset. Ties are broken by selecting literals that are most likely to lead to larger clusters when unset along with other variables.

Procedure 3.6 Bottom-Up-Full
Input: Constraints Γ, Model m
  P = Bottom-Up-Inexacts(Γ,m)
    loop:
      l = select-literal(Γ,P,m)
      P := P - l
      break if P = Ø or terminate(P)
    end loop
  return propagate(Γ,P)

After producing a cluster we need to project it onto the interface variables. In practice, for the experiments reported in the next section, the resulting clusters are sufficiently well constrained, and the residual theory is small enough, that the projection can be done by simply resolving away the internal variables.

3.2. PROJECT AND CLUSTER

The naive way is to resolve away variable in order to produce the equivalent SAT theory of the interface only, $\Gamma^P(x)$ of (11), and then to use the methods given above to find a cluster.

Unfortunately, simple removal of the private variables by resolution is impractical because the size of the theories blows up. Probably, in practice, some heuristic approximation methods will be suitable. However, for our experiments, the final size of the projected theory is actually quite small, and the blow-up problem only arises at intermediate stages of removal of private variables. Hence, we managed to avoid the blowup by producing the projected theory by explicit search for all the models on the interface variables. We then performed top-down search (without a reference model) with a termination criterion of achieving a pure cluster. This was done by iteratively selecting and enforcing literals that occur in the most remaining models. We emphasize that we do not expect such a naive method will be practical in general, however, it gave us a simple method to perform the experiments, and believe that there is great scope for developing heuristic methods to quickly produce good clusters.

4. EXPERIMENTAL RESULTS

As explained earlier, and in order to provide some early tests of the basic ideas, we will consider the simple case in which the distributed search is a very simple single probe, or "1-samp".

Specifically, we take a chain of problems, as in Figure 1. The first node generates a cluster and passes it to the second. In turn the second incorporates the cluster as additional constraints and passes this to the next node in the chain, etc. It is possible that some node fails to find any model or cluster, and then we shall say that the 1-samp probe failed.

We will measure the probability that we successfully reach the end of the chain and so have a solution to the entire problem. Our goal is to determine how this varies depending on the method used to generate clusters, and also to see how it depends upon the constrainedness of the problems at the local nodes, and in particular to how it relates to the phase transition likely to occur at the local nodes.

In light of the interest in phase transitions we consider a sequence of overlapping Random 3SAT problems. These are well-studied problems [14, and many others] and there is known to be a phase transition at a clause/variable ratio of about 4.25: above this value instances are almost always unsatisfiable, below it they are almost always satisfiable and also relatively easy to satisfy.

The particular configuration we use is that each local node

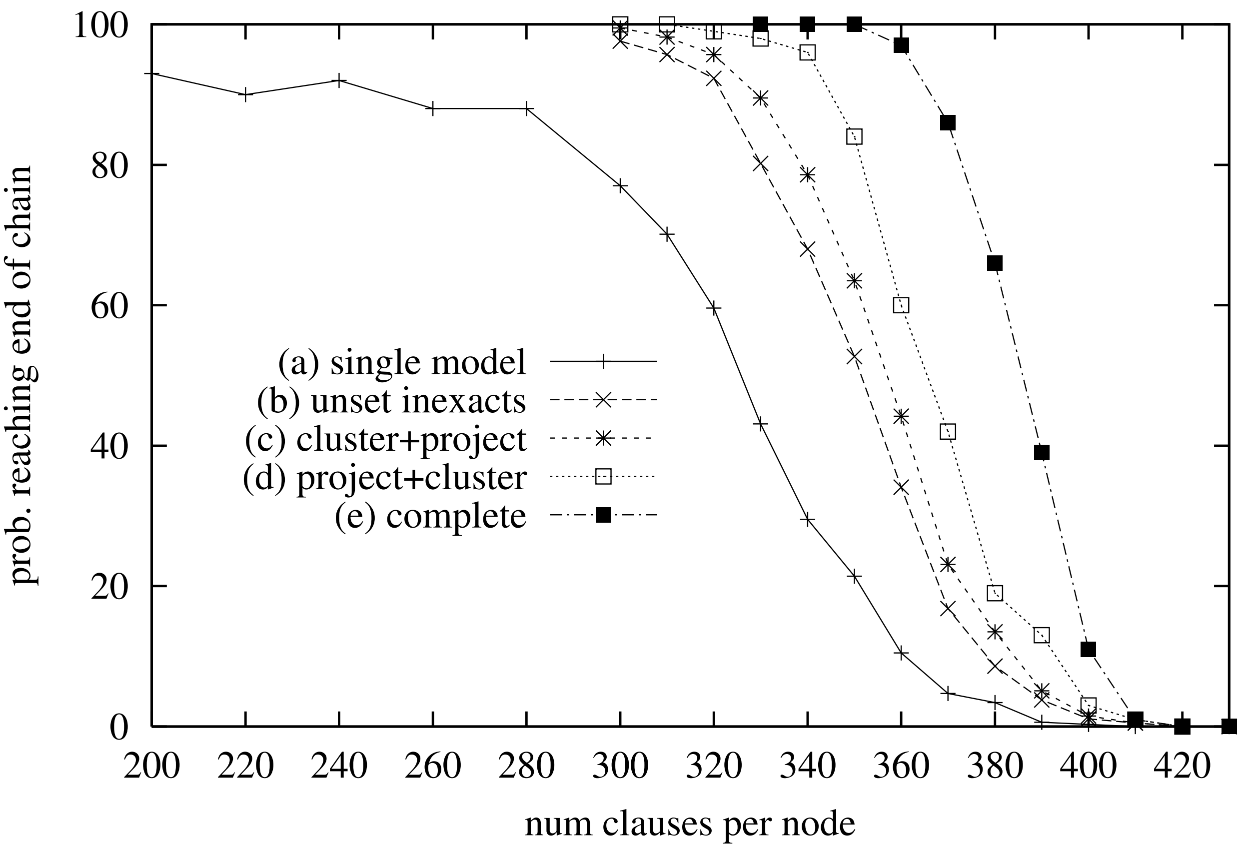

Figure 3: Probability of a single probe succeeding; Interface size 10, chain length, 3. With the given amounts of transfer of clusters a-e as described in Section 4.

consists of 100 variables, of these 10 are designated as output interface, and a disjoint 10 variables as the input interface. We take a chain length of 3, that is, 3 distinct local nodes. At 100 variables we need about 430 clauses before the instances have a 50% chance of being unsatisfiable, hence we look at cases ranging from the very easy-to-solve cases with 200 clauses, up to the much harder ones near to 430 clauses.

We measure the probability of 1-samp solving the entire theory and do this for 5 separate methods of generating clusters.

(a) Single model: a single satisfying total assignment is found using the local search algorithm WSAT [15] and projected onto the interface variables. If WSAT fails to find a model then we report failure, though we give it enough time such that on these problems any failure is almost surely a result of the theory being unsatisfiable.

(b) Unset inexacts: a single model is found as in (a) but we then apply the Procedure 3.5 of Section 3.1 to generate a cluster that is then projected onto the interface.

(c) Cluster+Project: as for (b), but we use Procedure 3.6 of Section 3.1 in order to generate slightly larger clusters before projection.

(d) Project+Cluster: As discussed in Section 3.2 we first project the entire local theory onto the interface and then generate a pure cluster of the projected theory. As discussed in Section 2.3 we can expect this cluster to be larger than that of a cluster+project method.

(e) Complete search. The local theories are combined and solved using a sequential systematic solver. This gives a reference line as it represents what would happen if the output "cluster" were just to be the entire projection onto the public variables: $\Gamma^P(x)$ of (11).

Generally, we expect that the size of the clusters produced will increase from the smallest with method (a) to the largest with (e).

The results for the success probability of search are given in Figure 3 where each of the lines, (a)-(e), corresponds to the methods above. Figure 3(e) is also the usual phase transition curve for satisfiability of the entire system and as such it is the best that any method can achieve.

As expected, the general trend is that the larger the cluster the larger the chance of success. Maybe the most significant result is that the method (a) performs poorly even in the relatively under-constrained region. For example around c=300 method (a) has a 20-30% chance of failure, whereas even a simple method such as (b) is achieving almost 100% success. In this region, simple unsetting of inexact variables reduces the failure rate by a factor of at least 5.

Possibly, passing impure clusters will ultimately cause more trouble than benefit for distributed search, however, Figure 3(d) shows even passing pure clusters can be beneficial.

Note we do not touch the question of how the distributed solver would compare against a centralized solver. We assume that the distribution is forced by other issues such as insufficient memory on a single processor, or perhaps data security requirements meaning that sending all the constraints to a central processor is simply not allowed.

Unfortunately, the implementations used in order to find such clusters were too preliminary for it to be meaningful to report times needed to find such clusters. However, we can expect that the order of the list (a)-(e) is from fastest to slowest. The method (b) of unsetting inexacts can also be expected to be relatively fast and is likely to be a very cheap post-processing step after finding an initial single model. Method (c) is likely to slightly slower than (b). Method (d) will probably be significantly slower than (c) as it requires some sort of projection onto the the interface, however, depending on the distributed search method, maybe spare CPU cycles can be used to build up such larger clusters. Finally, (e) is of course not meant as a practical method, but as a reference to measure the relative success of the others.

Future work in this Random 3SAT-based domain could look at how the problem parameters such as clause/variable ratios, interface size and chain length affect derived properties such as cluster sizes, purity or impurity of clusters, and how all this interacts with overall search costs.

5. RELATED WORK

Lesaint [8] has considered "maximal sets of solutions" for CSPs, and these correspond to what we have called pure clusters. He also presents algorithms to find such maximal sets, whereas we have taken a more opportunistic approach of first finding a single solution and then post-processing to produce a cluster. Presumably, Lesaint's methods have the potential to find better (larger) clusters, whereas post-processing methods are easier to integrate with existing solution methods.

Our work is also related to distributed asynchronous search methods based on the exchange of aggregates of (partial) solutions [16]. The work here differs in that it considers complex local problems, and so finding the aggregates/clusters becomes harder, and local phase transitions also became an issue. Conversely, we selected a much simpler (even simplistic) overall search routine. It would certainly be interesting to combine our work with that on complex local problems, [18] and that on the exchange of aggregates [16].

Another body of work with potential relations to clusters is that on interchangeability of values in CSPs [4, 6, 2]: groups of solutions (or locally consistent solutions) that are related by interchange of values of values for variables. It would be interesting to see whether interchanges (or symmetries) could be used to generate some forms of clusters, and whether the cluster methods might help to detect interchange symmetries in large problems.

Finally,[4] it has often been observed that using groups of solutions rather than individuals has appeared before in other contexts. Firstly, a group of solutions is naturally more robust than a single solution and so has is useful for problems with uncertainty [7]. Secondly, in problems with continuous variables then solutions are described by interval constraints on variables rather than equalities [13, 12, 1] and so solutions are naturally groups of solutions – rather akin to a pure cluster, but containing an infinite number of distinct "models." Possibly our work on impure clusters and distributed search is also applicable to numerical CSPs, or could benefit from the existing work in that area.

6. CLOSING DISCUSSION

This paper has been motivated by distributed search problems in which communication channels are relatively slow and expensive, and the local nodes can be significantly sized problems in their own right. A reasonable example would be a web-based set of agents working together to solve some some resource limited scheduling/timetabling problem.

Clusters, as known from phase transition and optimization work on non-distributed systems, are compact ways to represent an exponential number of solutions. We are proposing that distributed search methods in which local nodes compute and then exchange clusters might reduce communications costs and improve overall search. That is, instead of search passing simple message about discovered models, or speculative assignments, we consider more complex messages in particular ones giving information about many local solutions in a single message.

Hence a prime question is whether communicating various kinds of cluster can usefully reduce the number of messages. Or, similarly, whether search with a bound on the number of messages has a higher chance of success with clusters, than without. We hope to have set up some useful infrastructure for attacking such questions, and to have provided some evidence for the utility of the idea. In particular, we

- defined various kinds of clusters relevant to single theories and also to distributed problems. We focused on distributed problems with complex local nodes [18] and in which not all variables of a node are equally connected to other nodes.
- proved that finding optimal (largest-size) clusters is typically NP-hard. However, we expect that sub-optimal clusters will be quite adequate for the distributed search and we can expect to find these quickly using heuristics.
- gave some simple algorithms to find such clusters

[4] I am grateful to the anonymous reviewers for suggesting potential connections to work on numerical CSPs, and the associated references.

- experimentally investigated a simple incomplete distributed search method and found that communication with clusters can indeed dramatically increase the probability of success, with larger clusters leading to larger increases.

We have set up the basic formalization and have seen a case in which clusters have a potential to improve search. However, clearly further work is needed. Most pressing is the need to implement cluster-passing within a system where computational and communication costs are modeled or measured directly, and then to compare against a standard distributed algorithm. The goal will be to study the tradeoff between the computational cost of increasing the size of a given cluster, and the gains for the overall search from uses of a larger clusters. For such a test to be fair we will first need to improve our algorithms for finding clusters: currently they are rather naive, however we expect that the heuristics could be significantly improved. Most hopeful, at least in the short term, is to do simple methods such as unsetting of inexact literals (Section 3), so as to produce clusters with very little more cost than that of finding the initial solution. We will also need to consider more complex interactions between agents than the simple chains of Section 4.

We remark that some similar communication issues arise in the work on distraction and phase transitions [9]. Distraction raises the question of how soon an agent should pass on information about its own speculative search to other agents. If it passes on partial assignments before they have a good chance of being extendable to models then maybe it will cause other agents to waste lots of effort. Conversely, if it waits too long then it is harder for nodes to co-ordinate their efforts in compatible regions of the search space. Our work is related in that we are making the agents work harder to ensure that other agents have as much freedom as possible: rather than allowing early communication we are delaying it a bit even after a model is found. We suspect that in environments with high communication latency that it is better to do this extra local work and send a more informative message.

Our aim in studying the simple sampling-based nonsystematic search was to be useful to large systems that are at the edge of what is solvable, and for which satisficing rather than optimization might be the best we can hope for. However, in smaller systems it can be appropriate to use a (backtracking) systematic search method such as AWC [17]. If a local node detects an inconsistency between its internal constraints and messages received from other agents then it produces a no-good and broadcasts it to relevant agents. In our case we will need to produce no-goods that record that the combination of clusters received from other nodes is inconsistent. Pure clusters are totally described by the CPA which is a partial assignment and so can be used the same way as for more standard methods. However, this is also true even for impure clusters in which the residual theory is non-trivial. The no-good need not directly involve the residual theory because it is logically entailed by the CPA together with the constraints of the local node supplying the cluster (see (1)). Hence, it seems likely that cluster-passing methods can be implemented in a systematic solver, especially given that aggregates have already been used in an asynchronous complete search [16]. Of course, the node rejecting the CPA also has the option to observe that some subset, of the CPA, is itself sufficient to prove failure. Hence, it is likely that the CPA generators discussed in Section 2 could play a role in providing shorter, and hence more effective, no-goods. Also, note that our formalization of clusters was given in terms of decision problems. For practical problems we will also need to extend the definitions and algorithms to handle partial satisfaction or optimization problems.

Finally, we observe that distributed search methods can have troubles with making sure that every node is fully using its available CPU cycles. Maybe it would be useful for local nodes to absorb some spare cycles to do knowledge compilation by building clusters from models that have already been discovered. For example, even if it turns out to be better to communicate solutions as soon as they are discovered it might still be useful for a local node to expand it to a cluster in case a neighboring node requests some small changes to the initial solution. Possibly some of the delays from projection of theories and clusters onto the interfaces could also be reduced this way.

ACKNOWLEDGMENTS

This work was sponsored in part by grants from Defense Advanced Research Projects Agency (DARPA), number F30602-98-2-0181, and Air Force Research Laboratory, Rome, NY, under agreement numbered F30602-00-2-0532.

REFERENCES

[1] Frédéric Benhamou and Frédéric Goualard. Universally quantified interval constraints. In Rina Dechter, editor, *proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming (CP'2000). Lecture Notes in Computer Science vol. 1894*, pages 67–82. Springer-Verlag, 2000.

[2] Berthe Y. Choueiry and Gouvera Noubir. On the computation of local interchangeability in discrete constraint satisfaction problems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI–98)*, pages 326–333, 1998.

[3] Jon W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[4] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI–91)*, pages 227–233, 1991.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, 1979.

[6] Alois Haselböck. Exploint interchangeabilities in constraint satisfaction problems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 282–287, 1993.

[7] Luc Jaulin. *Solution globale er garantie de prolèmes ensemblistes*. PhD thesis, Université de Paris-Sud, Orsay, 1994.

[8] David Lesaint. Maximal sets of solutions for constraint satisfaction problems. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI–94)*, pages 110–114, 1994.

[9] D. Mammen and V. Lesser. Problem structure and subproblem sharing in multi-agent systems. In *International Conference on Multi Agent Systems (ICMAS 98)*, 1998.

[10] Remi Monasson and Riccardo Zecchina. Entropy of the k-satisfiability problem. *Physical Review Letters*, 76(21):3881–3885, 1996.

[11] Andrew J. Parkes. Clustering at the Phase Transition. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 340–345, Providence, RI, 1997.

[12] D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints: An International Journal*, 1:85–118, 1996.

[13] Djamila Sam-Haroud. *Constraint consistency techniques for continuous domains*. PhD thesis, Federal Institute of Technology, Lausanne, Switzerland, 1995.

[14] Robert Schrag and James M. Crawford. Implicates and prime implicates in Random 3SAT. *Artificial Intelligence*, 88:199–222, 1996.

[15] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring and Satisfiability*, pages 521–531. American Mathematical Society, 1996. Proceedings of the second DIMACS Implementation Challenge, October 1993.

[16] M. Silaghi, D. Sam-Haroud, and B. Faltings. Asynchronous search with aggregations. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI–2000)*, pages 917–922, 2000.

[17] M. Yokoo. Asynchronous weak commitment search for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on Constraint Programming*, pages 88–102, 1995.

[18] M. Yokoo and K. Hirayama. Distributed constraint satisfaction algorithm for complex local problems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 372–379, 1998.

[19] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.