Algo dec_21

Introduction

Suppose we are given a problem

$$\exists_X \phi(X,\overline{X}) \wedge \alpha(\overline{X}) = [\exists_X \phi(X,\overline{X})] \wedge \alpha(\overline{X})$$

Where ϕ and α are in CNF, and X and $\overline{X}$ are disjoint sets of variables.

Since α does not depend on X, it can actually be removed from the problem. We might still decide to use it while running the factor graph algorithm, since adding more factors can only give tighter results, but it's not necessary, so let's ignore it for now.

Suppose running the factor graph algorithm gives us $\phi'(\overline{X})$

The solution is strictly an over-approximation if there are solutions to:

$$\phi'(\overline{X}) \wedge \neg \exists_X \phi(X,\overline{X}) = \phi'(\overline{X}) \wedge \forall_X \neg \phi(X,\overline{X})$$

i.e., there exists an $\overline{X}$, say $\overline{a}$, such that $\overline{a}$ satisfies the factor graph solution $\phi'(\overline{X})$, and makes the original problem ϕ unsat. This algorithm is based around the hunt for such $\overline{a}$ assignments, and using them to make our solution tighter.

Using $\overline{a}$ to get better solutions

One way to ensure that the factor graph solution $\phi'(\overline{X})$ is not satisfied by $\overline{a}$ is simply to introduce $\neg\overline{a}$ as a factor before running the algorithm. Also note that $\neg\overline{a}$ is a disjunction of literals, and therefore, can be added as a CNF clause to the original formula. And because it does not contain any variables from $\overline{X}$, it should be included into $\alpha(\overline{X})$ instead of $\phi(X,\overline{X})$.

Searching for $\overline{a}$

Let's look at $\phi(X,\overline{a})$. This is the same as $\phi(X,\overline{X})$, but with some clauses removed, and some clauses shorter (without the $\overline{X}$ variables). Also look at another formula $\psi(X)$, which was created by removing all the non quantified variables $\overline{X}$ from $\phi(X,\overline{X})$. (Note that each clause in ϕ has at least one variable from X). $\psi(X)$ is the same as $\phi(X,\overline{a})$, but it might also have a few extra clauses which were removed by $\overline{a}$. Therefore,

$$\psi(X) \Rightarrow \phi(X,\overline{a})$$

And therefore, in order for $\phi(X,\overline{a})$ to be unsat, $\psi(X)$ must also be unsat, and therefore, $\psi(X)$ must have some minimal unsat cores (MUC) $\mu_1, \mu_2 \ldots \mu_m$.

Furthermore, because the clauses in of $\phi(X,\overline{a})$ are a subset of the clauses of $\psi(X)$, the set of MUCs of $\phi(X,\overline{a})$ must be a subset of the MUCs of $\psi(X)$.

Therefore, the following problem:

Find $\bar{a}$ such that $\phi(X, \bar{a})$ is unsat, and $\phi'(\bar{a})$ is not $\perp$.

is now reduced to a simpler problem:

Assignment driven search: Find $\bar{a}$ such that applying it to $\overline{X}$ in $\phi(X, \overline{X})$ generates/preserves one or more MUCs of $\psi(X)$ and $\phi'(\bar{a}) = \top$.

or equivalently:

MUC driven search: For each MUC μ of $\psi(X)$, check whether it is possible to find an assignment $\bar{a}$ of $\overline{X}$ in $\phi(X, \overline{X})$ such that the clauses in μ are preserved, and $\phi'(\bar{a}) = \top$.

The following two sections explore these two techniques.

Assignment driven search

Find $\bar{a}$ such that applying it to $\overline{X}$ in $\phi(X, \overline{X})$ generates/preserves one or more MUCs of $\psi(X)$ and $\phi'(\bar{a}) = \top$.

Let us look at how an assignment $\bar{a} = l_1 \wedge l_2 \wedge \ldots \wedge l_N$ affects $\phi(X, \overline{X}) = D_1 \wedge D_2 \wedge \ldots \wedge D_M$. Each disjunctive clause D_i either:

- Disappears, if it contains *any* of the l_j literals.

or:

- Otherwise (if it does not have any l_i literals), it loses all the $\neg l_j$ literals it might have.

Furthermore, each clause C_k in $\psi(X) = C_1 \wedge C_2 \wedge \ldots \wedge C_M$ comes from a corresponding D_k in $\phi(X, \overline{X})$ that has all the literals from C_k, plus some literals on the variables in $\overline{X}$, and C_k would be preserved as a clause in $\phi(X, \bar{a})$ if and only if C_k does not have the negation of any literals from $\bar{a}$.

Therefore, if two clauses C_j and C_k in $\psi(X)$ have opposite signs of some literal l, then at most one of the clauses can be preserved in any given assignment $\bar{a}$. (Suppose C_j contains l and C_k contains $\neg l$, then if $\bar{a}$ contains l, it preserves C_j but omits C_k, and if $\bar{a}$ contains $\neg l$ instead, then it preserves C_k and suppresses C_j.)

Therefore, if we create an undirected graph G:

- whose nodes are the clauses D_i in $\phi(X, \overline{X})$,
- and two nodes D_j and D_k are connected by an edge if and only if they have opposite literals on some variable in $\overline{X}$, then each

then each assignment $\bar{a}$ will preserve a set of clauses C_i whose corresponding set of clauses D_i will create an independent subset of this graph.

Furthermore, since our objective is to look for assignments that preserve MUCs, we can make sure we do not miss any MUCs by looking at all maximal independent subsets. Therefore, the algorithm now becomes:

Create the graph G as described above.
For every maximal independent set I :
    Find the corresponding subset S_I of clauses from $\psi(X)$
    If S_I is un-satisfiable:
        Find the minimal partial assignment $\bar{a}$ of $\overline{X}$ that reduces $\phi(X,\overline{X})$ to a superset of S_I
        If $\phi'(\bar{a}) \neq \perp$:
            use $\bar{a}$ to improve $\phi'(\overline{X})$

MUC driven search

The disadvantage with an assignment driven search from the previous section is that there might be intersecting independent subsets generated by the algorithm, causing the algorithm to redundantly look for MUCs in those intersecting sets multiple times. Another approach would be to instead generate MUCs and then check if they form an independent subset.

For every MUC μ of $\psi(X)$:
    Find the corresponding set σ_μ of nodes in G
    If σ_μ forms an independent set in G :
        Find the minimal partial assignment $\bar{a}$ of $\overline{X}$ that reduces $\phi(X,\overline{X})$ to a super set of μ
            If $\phi'(\bar{a}) \neq \perp$:
                Use $\bar{a}$ to improve $\phi'(\overline{X})$

This algorithm would not work well if $\psi(X)$ has many MUCs, and none of whose corresponding subset of nodes in G forms an independent subset.
An improved approach would be to avoid searching for *ingenerable* MUCs (whose corresponding set of nodes in G don't form an independent set, and therefore, make it impossible to find an assignment of $\overline{X}$ variables that can generate the MUC) by modifying the MUC search algorithm. As it turns out, all the algorithms in *MUST* (the leading MUC generation tool at the time of writing this document) work by searching for an *unexplored set* of clauses. If the set of clauses is satisfiable, then it marks all subsets of the set as *explored*, and the algorithm repeats. If the set of clauses is unsat, then (after marking all supersets as explored) the algorithm proceeds to *shrink* the set to find the minimal unsatisfiable core, the MUC. While there are many flavours of this algorithm utilizing different insights on how to generate a "good" unexplored set, and how to shrink in a "better" way, all the algorithms fundamentally depend on a data structure for marking sets of clauses (and their super/sub sets) as explored, and searching for unexplored sets. This works well for us, as it allows us to mark *ingenerable* sets as already explored and avoid their exploration for the generation of MUCs. Thus, our algorithm becomes:

For every variable $\bar{x} \in \overline{X}$:
    For every clause C_i in $\phi(X,\overline{X})$ that has the literal $\bar{x}$:
        For every clause C_j in $\phi(X,\overline{X})$ that has the literal $\neg\bar{x}$:
            Find the corresponding clauses D_i and D_j in $\psi(X)$
            Mark the set $\{D_i, D_j\}$ and all its super-sets as *explored*
For every MUC μ of $\psi(X)$: // discovered while avoiding explored sets
                                   // therefore guaranteed to be *generatable*
    Find the minimal partial assignment $\bar{a}$ of $\overline{X}$ that reduces $\phi(X,\overline{X})$ to a super set of μ
    If $\phi'(\bar{a}) \neq \perp$:
        Use $\bar{a}$ to improve $\phi'(\overline{X})$

Thus, we have effectively modified the MUC discovery algorithm to avoid exploring MUCs that are not of interest to us (since they cannot be generated by any assignment $\bar{a}$ of $\overline{X}$). Extending the idea further, we would like also avoid exploring MUCs whose corresponding assignment does not satisfy the factor graph solution $\phi'(\overline{X})$ (i.e., avoid $\bar{a}$ s.t. $\phi'(\bar{a}) eq \perp$). Ideally we would do this at the seed generation stage, perhaps by marking all such sets as already explored. But it's not obvious how to achieve this.

However, once an unexplored set of clauses has been selected for shrinking, then along with it's satisfiability, we can also check for it's compatibility with ϕ' . If it is compatible, then we can continue shrinking as per normal, and in fact, never have to check for compatibility for any subsets. On the other hand, if it isn't compatible, we cannot rule it out completely, because a subset being explored during the process of shrinking might become compatible. Instead, we must now check for compatibility at each step of the shrinking process.

And therefore, since we have to check for ϕ satisfiability at all steps of the shrinking process, it seems it might just be better to perform the check at the last step, once an MUC has been found. However, the question is, whether the ϕ satisfiability checks can be used to guide the shrinking process, to avoid exploring sets of clauses that do not satisfy ϕ , and therefore minimise the number of calls to sat solving.