CS 70 Discrete Mathematics for CS
Spring 2005 Clancy/Wagner Notes 9

Lecture Notes 8 described methods for logical reasoning based on satisfiability testing and introduced Minesweeper. In this lecture, we show how to construct a complete Minesweeper program. We show that some aspects of Minesweeper are computationally intractable if handled naïvely. Methods for problem decomposition can help.

Minesweeper in CNF

Lecture Notes 8 gave a simple example of how to formulate a logical description of a Minesweeper display as a set of clauses. If d is a display, $CNF(d)$ denotes the corresponding CNF expression. We will now show how to construct $CNF(d)$ systematically for any display.

$CNF(d)$ consists of propositions arising from each of the known squares, plus the global constraint on the total number of mines left. We begin with the known squares. Consider a known square, such as (2,1) in the following example repeated from Lecture Notes 8:

2			
1	1	**1**	1
	1	2	3

(2,1) has 1 mine adjacent to it. There are 5 adjacent squares; 2 of them have been probed and are known to be safe; 0 of them are marked as a mine already. There are $n = 5 - 2 - 0 = 3$ unknown adjacent squares, of which $k = 1 - 0 = 1$ are mines. So we need to express in CNF the proposition that k of the n adjacent unknown squares are mines. Call this proposition $KN(k,n)$.

Let's see what we have to work with. CNF requires a conjunction of disjunctions of literals. A disjunction of literals means "at least one ... ," i.e., an inequality. The literals can be either "(i,j) contains a mine" or "(i,j) doesn't contain a mine." Because $KN(k,n)$ is entirely symmetric with respect to the unknown squares, we might expect to generate clauses that are symmetric—for example, all positive literals or all negative literals.

We begin by writing $KN(k,n)$ as two inequalities:

$$KN(k,n) \equiv (U(k,n) \wedge L(k,n))$$

where

$U(k,n)$ means that *at most* k of the n squares contain a mine.
$L(k,n)$ means that *at least* k of the n squares contain a mine.

So how do we express "at most k" using clauses that say "at least one"? Consider any subset of $k+1$ squares from the n unknown squares. If at most k are mines, then at least one is not a mine; the converse is also true. So we have the following:

$U(k,n) \equiv$ for any $k+1$ squares out of n, at least one is not a mine.

Similarly, consider any subset of $n-k+1$ squares: if at least k of all n squares are mines, then at least one of any $n-k+1$ squares must be a mine; the converse is also true. Hence

$L(k,n) \equiv$ for any $n-k+1$ squares out of n, at least one is a mine.

Applying these formulations to the square (2,1) in the above example, where $k=1$ and $n=3$, we obtain:

$U(1,3) \equiv$ for any 2 squares out of 3, at least one is not a mine.
$L(1,3) \equiv$ for any 3 squares out of 3, at least one is a mine.

Translating into a Boolean expression, we obtain

$$U(1,3) \equiv (\neg X_{1,2} \vee \neg X_{2,2}) \wedge (\neg X_{2,2} \vee \neg X_{3,2}) \wedge (\neg X_{3,2} \vee \neg X_{1,2})$$
$$L(1,3) \equiv (X_{1,2} \vee X_{2,2} \vee X_{3,2})$$

exactly as we had in Lecture Notes 8.

Note that these expressions are valid when $k > 0$ and $k+1 \leq n$. The case $k = 0$ simply means that $KN(0,n)$ is the conjunction of the clauses $\neg X_i$ for all i. The case $k+1 > n$ can only arise if $k=n$, i.e., all n variables are mines; then we simply have the clauses X_i for all i.

We can also generate a CNF expression recursively as follows:

$$KN(k,n) \equiv ((X_n \implies KN(k-1,n-1)) \wedge (\neg X_n \implies KN(k,n-1)))$$

with base cases at $k=n$ and $k=0$, as before. Assuming $KN(k-1,n-1)$ and $KN(k,n-1)$ can be expressed in CNF, it is a simple distributivity step to express $KN(k,n)$ in CNF. The expressions resulting from this recursion look slightly different from those obtained above, but are logically equivalent.

In addition to the "local" constraints arising from squares, we also have the global constraint from the total number of remaining mines, M:

G: Exactly M of the unknown squares on the board contain mines.

If there are B remaining unknown squares, this is a set of clauses of the form $KN(M,B)$.

Preview: Counting things

We will now take a slight detour to check into how many clauses we are generating for Minesweeper.

Let $|KN(n,k)|$ be the number of clauses in $KN(n,k)$, using our first construction. How many is this? We have the following equation:

$$|KN(n,k)| = |L(n,k)| + |U(n,k)| = C(n,n-k+1) + C(n,k+1)$$

where the notation $C(n,k)$ is defined as follows:

Definition 9.1 (Combinations): $C(n,k)$ is the number of distinct subsets of size k drawn from a set of size n.

For example, $C(4,2) = 6$ because there are 6 subsets of size 2 in any set of size 4. $C(n,k)$ is often pronounced "n **choose** k." We will also define a related quantity $P(n,k)$:

CHOOSE

Definition 9.2 (Permutations): $P(n,k)$ is the number of distinct ordered k-tuples drawn without replacement from a set of size n.

The basic distinction between $P(n,k)$ and $C(n,k)$ is that for $P(n,k)$ order matters, whereas for $C(n,k)$ it does not. It is easiest to derive a formula for $P(n,k)$ first:

Theorem 9.1: *For any natural numbers n, k, such that $k \leq n$,*

$$P(n,k) = \frac{n!}{(n-k)!}$$

Proof: The first element of the tuple can be drawn in n ways, the second in $(n-1)$ ways, and so on down to the last element, which can be drawn in $n-k+1$ ways. Hence, $P(n,k) = n \cdot (n-1) \cdots (n-k+1) = n!/(n-k)!$. $\square$

Any subset of k elements from n will occur repeatedly in the set of permutations, with $k!$ different orderings. Hence, we have the following formula for $C(n,k)$:

Theorem 9.2: *For any natural numbers n, k, such that $k \leq n$,*

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

We can see, by symmetry, that the following identity holds:

$$C(n,k) = C(n,n-k)$$

Hence, returning to our formula for the number of clauses we generate, we have

$$|KN(n,k)| = C(n,n-k+1) + C(n,k+1) = C(n,k-1) + C(n,k+1)$$

In the worst case, for an individual square, $n=8$ and $k=4$, giving us $|KN(8,4)| = C(8,3) + C(8,5) = 112$ clauses. This isn't too big. But for the global constraint, on an 8×8 board with 10 mines (an easy case), we have $|KN(64,10)| = C(64,9) + C(64,11) = 771,136,366,336$. So we'll think of another way to handle the global constraint!

Incidentally, it's not too hard to prove a nasty lower bound on the largest number $C(n,k)$, for any given n.

Theorem 9.3: *For some k, $C(n,k) \geq 2^n/(n+1)$.*

Proof: Consider the sum over all k of $C(n,k)$, which is the sum of the number of subsets of size k, for all k. This is just the total number of subsets of a set of size n, which is 2^n. That is,

$$\sum_{k=0}^{n} C(n,k) = 2^n$$

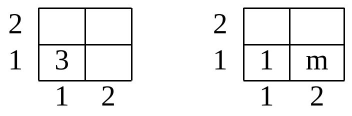
Now the sum contains $n+1$ numbers, so at least one of them must be greater than or equal to $2^n/(n+1)$. $\square$

GENERALIZED PIGEONHOLE PRINCIPLE

The last step is an application of the **generalized pigeonhole principle** (see Rosen, p.245): if N objects are placed in k boxes, then there is at least one box with at least $\lceil N/k \rceil$ objects.

End of detour! We'll return to counting later when we do probability.

A brain-dead algorithm for Minesweeper



Given a CNF representation $CNF(d)$ of a Minesweeper display d, there are some “obvious” inferences one can often make. For instance, in the example on the left, the CNF representation will be three **unit clauses** with positive literals:

UNIT CLAUSES

$$(X_{1,2}) \wedge (X_{2,2}) \wedge (X_{2,1})$$

Simply by and-elimination, we can see that the three squares contain mines (as one would hope). Similarly, in the example on the right, the CNF representation will be

$$(\neg X_{1,2}) \wedge (\neg X_{2,2})$$

Again, we have unit clauses, this time with negative literals, and can conclude immediately that $(1,2)$ and $(2,2)$ are safe. Thus, the conclusions that are “obvious” to a human player are also “obvious” in the CNF representation. So we can define our first simple algorithm:

Definition 9.3 (Brain-Dead Minesweeper):

```
Given a display $d$, generate $CNF(d)$.
If $CNF(d)$ contains a positive unit clause $(X_{i,j})$, mark $(i,j)$ as a mine
else if $CNF(d)$ contains a negative unit clause $(X_{i,j})$, probe $(i,j)$
else probe a random unknown square.
```

You can measure how well this does—not especially well! There are many cases where this does not do well; the two examples in Lecture Notes 8 (“three 1s” and “five 1s”) have no “obvious” move, but do have logically sound moves. Just doing the obvious inferences does not yield complete strategy:

COMPLETE

Definition 9.4 (Completeness (of an inference procedure)): A proof procedure is **complete** iff it can prove every proposition that is entailed by any given proposition.

Logical algorithms for minesweeper

To obtain a complete logical algorithm for Minesweeper, we use the notion of satisfiability testing from Lecture Notes 8. Remember that if $CNF(d) \wedge (\neg X_{i,j})$ is unsatisfiable, then $X_{i,j}$ is entailed by $CNF(d)$.

Definition 9.5 (Logical Minesweeper, Mark I):

```
Given a display $d$, generate $CNF(d)$.
If $CNF(d)$ contains a positive unit clause $(X_{i,j})$, mark $(i,j)$ as a mine
else if $CNF(d)$ contains a negative unit clause $(X_{i,j})$, probe $(i,j)$
else if $CNF(d) \wedge (\neg X_{i,j})$ is unsatisfiable for any $X_{i,j}$ in $CNF(d)$, mark $(i,j)$ as a mine
else if $CNF(d) \wedge (X_{i,j})$ is unsatisfiable for any for any $X_{i,j}$ in $CNF(d)$, probe $(i,j)$
else probe a random unknown square.
```

Notice that the algorithm doesn't specify which mine to mark first if several mines can be identified, nor which square to probe first if there are several safe squares. It is an interesting exercise to prove the following:

Theorem 9.4: *Between any two random moves, the Mark I algorithm marks exactly the same set of mines and uncovers the same set of safe squares, regardless of the order of selection.*

So, the Mark I basically makes every logically guaranteed move. In theory, this is fine; in practice, it won't work well at all. We have already seen that the global constraint can have exponentially many clauses. Moreover, the global constraint is defined on *all* the variables on the board, so for an $X \times Y$ board we'll have to enumerate 2^{XY} models. Hopeless!

We will have to adopt a more subtle strategy. Let's divide $CNF(d)$ into the local constraints $C(d)$ and the global constraint $G(d)$. First, we could simply pretend the global constraint doesn't exist:

Definition 9.6 (Logical Minesweeper, Mark II): Identical to the Mark I except that $CNF(d)$ is replaced by $C(d)$, the CNF form of the local constraints.

Theorem 9.5: *Every square that is "guaranteed safe" or "guaranteed mine" with respect to $C(d)$ is also "guaranteed safe" or "guaranteed mine" with respect to $C(d) \wedge G(d)$.*

That is, the Mark II's guaranteed moves are correct even though it ignores the global constraint! Is this some weird special property of minesweeper and the nature of the global constraint? Actually, it's just a special case of a much simpler and more powerful theorem regarding the **monotonicity** of logic:

MONOTONICITY

Theorem 9.6: *For any propositions A, B, and C, if $A \models C$ then $A \wedge B \models C$.*

The proof of this is left as an exercise. It's called monotonicity because as the set of known facts grows, the set of entailed conclusions grows monotonically; adding more known facts can *never* invalidate a previously derived conclusion.

FRINGE

BACKGROUND

The Mark II is much more efficient than the Mark I, because the variables of $C(d)$ are just those unknown variables that are *adjacent* to known squares. We'll call these the **fringe**. The runtime of the Mark II is $O(2^F)$, where F is the size of the fringe. The remaining background squares will be called the **background**; there are B background squares.

Mark II plays a pretty good game of minesweeper, but it sometimes has to guess in cases where the global constraint actually entails some guaranteed moves. There are two cases. First, the global constraint may rule out some models of the local constraints, so that guaranteed moves on the fringe can be made based on the remaining models. Second, the local constraints may determine some fixed number of mines on the fringe, such that the background must be all mines (or all empty); this allows guaranteed moves in the background squares. (You are asked to supply examples of these cases.) How can we incorporate the global constraint in our algorithm? Essentially by adding additional checks into the satisfiability test, where those checks are implemented "extralogically" (i.e., *outside* formal logic) by counting mines in models. Details of the Mark III are left to the homework.

Problem structure

Consider the problem shown in Figure 1(a). Any human looking at it would recognize instantly that there are really two separate problems here, if we ignore the global constraint. (It is also clear that there are no logical moves in either of them.)

The Mark II algorithm, however, generates a CNF representation that includes all the fringe variables—12 in all, giving $2^{12} = 4096$ models. We would like to be able to solve the subproblems separately. Each has

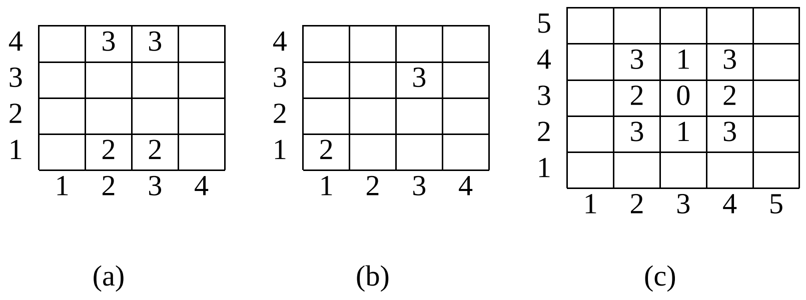


Figure 1: Problem decomposition. (a) Two completely disjoint subproblems. (b) Partial decomposition: if we knew $X_{2,2}$, we would have two disjoint subproblems. (c) Here, one possible cutset is $\{X_{2,1}, X_{3,1}, X_{4,1}, X_{2,5}, X_{3,5}, X_{4,5}\}$.

six variables, so we'd have a total cost of at most $2^6 + 2^6 = 128$ models. This huge reduction is available precisely *because* the exponential function grows so fast as the exponent increases. In fact, if we can divide a problem of size n into n/k constant-size pieces of size k, the total cost grows as $O(2^k \cdot n/k)$, which is *linear* in n!

PROBLEM DECOMPOSITION

Now we need to define the idea of **problem decomposition** more precisely. Decomposability is not something specific to minesweeper; it depends on some special structure in the CNF representation that some minesweeper problems generate. If we can recognize that special structure, we can do problem decomposition for *any* problem that can be represented in CNF.

Intuitively, the idea is that the variables divide into two sets

$$\{X_{1,4}, X_{1,3}, X_{2,3}, X_{3,3}, X_{4,3}, X_{4,4}\} \text{ and } \{X_{1,1}, X_{1,2}, X_{2,2}, X_{3,2}, X_{4,2}, X_{4,1}\}$$

such that no variable in one set has any "connection" to any variable in the other set. In CNF, two subsets of clauses are "disconnected" if they have no variables in common.

DISCONNECTED

Definition 9.7 (Disconnected expressions): Two Boolean expressions A and B are **disconnected** if they have no variables in common.

Figure 2(a) shows a graph (a network of nodes and links) illustrating the connectedness of the variables for the minesweeper problem in Figure 2(a). Two nodes are directly connected in the graph if they appear together in a clause. Disconnection of two sets of nodes is easy to see in this representation.

Now we need to understand the consequences of disconnection for the task of logical reasoning. Because satisfiability is our canonical tool for reasoning, let's see what happens:

Lemma 9.1: *If two Boolean expressions A and B are disconnected, then $A \wedge B$ is satisfiable iff A is satisfiable and B is satisfiable.*

Proof: Let $\mathbf{X}_A$ be the variables of A and $\mathbf{X}_B$ be the variables of B.
($\Rightarrow$): Show that if $A \wedge B$ is satisfiable then A is satisfiable and B is satisfiable. Let M_{AB} be any model for $A \wedge B$. Then A is true in M_{AB} and B is true in M_{AB}. Let M_A be the subset of M_{AB} specifying variables in A, and define M_B similarly. Since A and B share no variables, A must be true in M_A and B in M_B; hence A and B are satisfiable.
($\Leftarrow$): Show that if A is satisfiable and B is satisfiable then $A \wedge B$ is satisfiable. Let M_A be any model for A and M_B be any model for B. Define $M_{AB} = M_A \cup M_B$. If A is true in M_A then it is true in M_{AB}; similarly for B. Hence $A \wedge B$ is true in M_{AB}, so $A \wedge B$ is satisfiable. $\square$

From the lemma, the theorem we need follows directly:

Theorem 9.7: *Let C be a CNF expression; and let C_1 and C_2 be CNF expressions such that (1) C_1 and C_2*

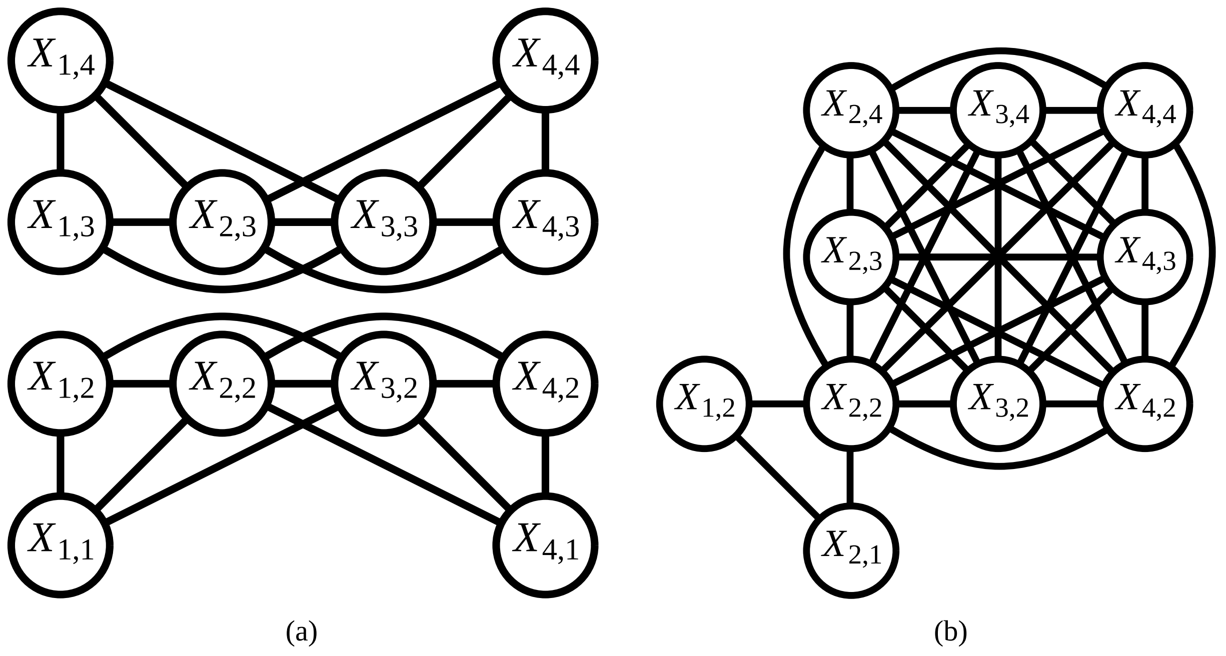


Figure 2: (a) Graph showing connectedness of the variables in Figure 1(a). (b) Graph for Figure 1(b).

share no variables, and (2) the union of the clauses in C_1 and C_2 gives exactly the clauses in C. Then for any proposition A, $C \models A$ if and only if $C_1 \models A$ or $C_2 \models A$.

The "if and only if" is important here. The $\Leftarrow$ direction is obvious—simply an application of monotonicity. It holds even if the subsets *do* share variables! Thus, any square that is guaranteed with respect to any subset of the constraints is also guaranteed with respect to all the constraints; so one way to achieve efficiency at the expense of completeness is just to divide the variables into small subsets and prove whatever guarantees one can from each subset.[1] The $\Rightarrow$ direction says that if the subsets are disconnected, we can be sure that doing the proofs using the subsets does *not* lose any guarantees.

Our next example, in Figure 1(b), shows a case where the variables cannot be divided into two disconnected sets. The graph is shown in Figure 2(b). So the simple decomposition approach will not help, and we appear to be faced with a 10-variable fringe or 1024 models.

But consider the following strategy: if we knew the truth value of $X_{2,2}$, then that variable would disappear from all clauses containing it (replaced by T of F). In that case, the remaining variables would be divided into two disconnected sets. This is easy to see in Figure 2(b): removal of the node for $X_{2,2}$ disconnects the two halves. The plan, then, is to solve the two halves with $X_{2,2} = T$ and with $X_{2,2} = F$. The total cost is at most $2(2^2 + 2^7) = 264$, substantially less than 1024.

CUTSET A set of nodes whose removal divides a graph into two or more disconnected components is called a **cutset**. For the problem just examined, the cutset is $\{X_{2,2}\}$. Sometimes it is not trivial to find a good cutset; for example, in Figure 1(c), the smallest cutset has six variables—e.g., $\{X_{2,1}, X_{3,1}, X_{4,1}, X_{2,5}, X_{3,5}, X_{4,5}\}$. This is best ascertained by looking at the graph for the problem. Without the cutset, the problem has $2^{16} = 65,536$ models to check. There are 64 possible assignments for the cutset variables, and the remaining subproblems have 5 variables each, so the total time is at most $64(2^5 + 2^5) = 4096$, i.e., 16 times faster than the original formulation.

This section has given some ideas about the importance of problem structure and some methods of taking

[1] We'll explore this idea more in the homework.

advantage of it. Clearly, graphs and graph algorithms have a lot to do with it, and these are studied in much greater depth in later courses.