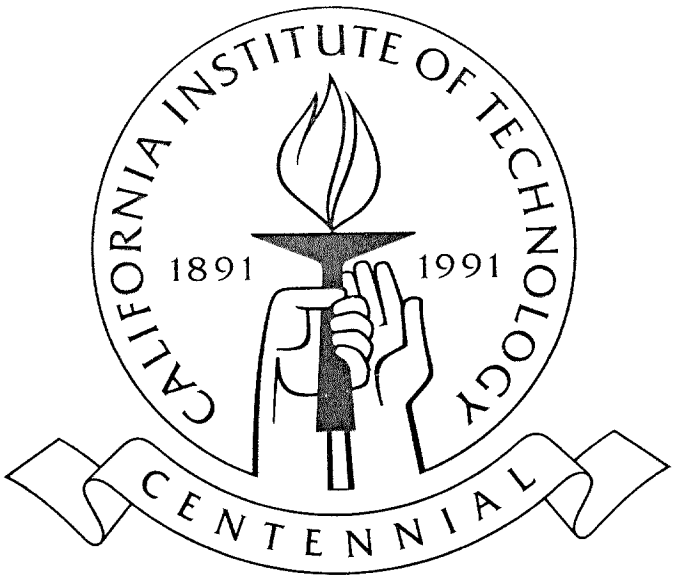

Characterizing NP and Measuring Instance Complexity

Stephen Judd

Computer Science Department
California Institute of Technology

Caltech-CS-TR-90-11

Characterizing *NP* and Measuring Instance Complexity

Stephen Judd
Computer Science Department,
Caltech 256-80
Pasadena, CA 91125

Version 1.4
July 31, 1990

Abstract

A generic *NP*-complete graph problem is described. The calculation of certain predicate on the graph is shown to be both necessary and sufficient to solve the problem and hence the calculation must be embedded in every algorithm solving *NP* problems.

This observation gives rise to a metric on the difficulty of solving *an instance* of the problem.

There appears to be an interesting phase transition in this metric when the graphs are generated at random in a "2-dimensional" extension. The metric is sensitive to 2 parameters governing the way graphs are generated: p, the density of edges in the graph, and K, related to the number of points in the graph. The metric seems to be finite in part of the (p, K)-space and infinite in the rest. If true, this phenomenon would demonstrate that *NP*-complete problems are truly monolithic and can easily exhibit strong intrinsic coupling of their variables throughout the entire instance.

Contents

1 The SIS Problem

1.1 Notation

A *relation*, R, on a set of *attributes*, β, into a set of values, B, is a set of total functions (called tuples) from α into B (which for simplicity here we will assume is $[K] = \{1, 2, 3, \ldots, K\}$).

$$R \subseteq \{f | f : \alpha \rightarrow [K]\}$$

α is referred to as $\text{DOM}(R)$. Below is an example relation called R with $\text{DOM}(R) = \{\text{a,b,c,d,e}\}$ which has 6 constituent tuples:

$R =$

a	b	c	d	e
1	2	3	3	3
1	2	3	3	1
2	4	4	3	1
2	4	5	3	1
2	4	2	3	2
2	4	1	3	2

In this tableau each row is a distinct tuple, the ordering of the rows does not matter, and the ordering of the columns does not matter as long as the attributes are permuted along with the rest of the column.

The *projection* operator, π, takes two arguments, one a relation and one a set of attributes. The set of attributes must be a subset of the attributes of the relation. It produces one relation which has only those attributes and whose tuples are each equal to one of the original tuples when restricted to the new set of attributes. Formally, if F is a relation, and $\beta \subseteq \text{DOM}(F)$ is a set of attributes, then

$$\pi_\beta(F) = \{g : \beta \rightarrow [K] \mid \exists f \in F \; g = f|_\beta\}.$$

For example, if R is the relation above then

$\pi_{\{a,b\}}(R) =$

a	b
1	2
2	4

$\pi_{\{d\}}(R) =$

d
3

$\pi_{\{b,d,e\}}(R) =$

b	d	e
2	3	3
2	3	1
4	3	1
4	3	2

The *join* operator, $\bowtie$, takes two relations as arguments and produces a relation. Every tuple in the result must be an extension of a tuple in both of the arguments. Formally, if $\alpha = \text{DOM}(F)$ and $\beta = \text{DOM}(G)$, then

$$F \bowtie G = \{h : \alpha \cup \beta \to [K] \mid \exists f \in F, g \in G \ni f = h|_\alpha,\ g = h|_\beta\}$$

As examples, let

$$R = \begin{array}{|cccc|}\hline \text{a} & \text{b} & \text{c} & \text{d} \\ \hline 1 & 4 & 2 & 3 \\ 1 & 4 & 3 & 3 \\ 2 & 4 & 4 & 1 \\ 5 & 4 & 3 & 1 \\ 2 & 5 & 5 & 5 \\ \hline \end{array} \quad S = \begin{array}{|ccc|}\hline \text{a} & \text{e} & \text{f} \\ \hline 1 & 2 & 3 \\ 2 & 4 & 4 \\ 2 & 4 & 5 \\ \hline \end{array} \quad T = \begin{array}{|cc|}\hline \text{c} & \text{d} \\ \hline 3 & 3 \\ 3 & 1 \\ \hline \end{array}$$

then

$$R \bowtie S = \begin{array}{|cccccc|}\hline \text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\ \hline 1 & 4 & 2 & 3 & 2 & 3 \\ 1 & 4 & 3 & 3 & 2 & 3 \\ 2 & 4 & 4 & 1 & 4 & 4 \\ 2 & 4 & 4 & 1 & 4 & 5 \\ 2 & 5 & 5 & 5 & 4 & 5 \\ 2 & 5 & 5 & 5 & 4 & 4 \\ \hline \end{array} \quad S \bowtie T = \begin{array}{|ccccc|}\hline \text{a} & \text{e} & \text{f} & \text{c} & \text{d} \\ \hline 1 & 2 & 3 & 3 & 3 \\ 1 & 2 & 3 & 3 & 1 \\ 2 & 4 & 4 & 3 & 3 \\ 2 & 4 & 4 & 3 & 1 \\ 2 & 4 & 5 & 3 & 3 \\ 2 & 4 & 5 & 3 & 1 \\ \hline \end{array}$$

$$R \bowtie T = \begin{array}{|cccc|}\hline \text{a} & \text{b} & \text{c} & \text{d} \\ \hline 1 & 4 & 3 & 3 \\ 5 & 4 & 3 & 1 \\ \hline \end{array} \qquad \begin{array}{|c|}\hline \text{a} \\ \hline 1 \\ \hline \end{array} \bowtie S = \begin{array}{|ccc|}\hline \text{a} & \text{e} & \text{f} \\ \hline 1 & 2 & 3 \\ \hline \end{array}$$

The last example has a relation written into the expression as a constant. The result of a join operator has as attribute set the union of the attributes of its arguments. The number of tuples in a join can be larger than either of its arguments (as many as the product of the numbers in the arguments), or it can be smaller (even zero). Join commutes and associates.

If $\mathcal{R} = \{R_1, R_2, \ldots\}$ is a set of relations, then

$$\bigbowtie_{R \in \mathcal{R}} R = R_1 \bowtie R_2 \bowtie \ldots$$

is the obvious analogue to the common big sigma notation.

The *select* operator, σ, takes a relation, an attribute, and a value as arguments and produces a relation. The attribute is required to be in the domain of the relation. Every tuple in the result has the given value for the given attribute. Formally, if S is a relation, v is an attribute, and i is a value,

$$\sigma_{\substack{v \\ i}}(S) = \{h : \text{DOM}(S) \to [K] \mid h \in S,\ h(v) = i\}$$

Select commutes and associates and we shall use the following abbreviation:

$$\sigma_{\substack{v_1 & v_2 & \dots & v_m \\ i_1 & i_2 & \dots & i_m}}(S) = \sigma_{\substack{v_1 \\ i_1}}(\sigma_{\substack{v_2 \\ i_2}}(\dots \sigma_{\substack{v_m \\ i_m}}(S)))$$

Lemma 1 $S \supseteq \sigma_{\substack{v_1 \\ i_1}}(S) \supseteq \sigma_{\substack{v_2 \\ i_2}}(\sigma_{\substack{v_1 \\ i_1}}(S)) \supseteq \dots.$ □

Lemma 2 *Whenever* $\alpha \subseteq \text{DOM}(R_2)$, *and* $\beta \subseteq \alpha \cup \text{DOM}(S)$, $R_1 = \pi_\alpha(R_2) \Longrightarrow \pi_\beta(R_1 \bowtie S) \supseteq \pi_\beta(R_2 \bowtie S)$. □

Lemma 3 $h \in \pi_\gamma(R) \Longrightarrow \exists h' \in R \ni h'|_\gamma = h.$ □

Lemma 4 *Let* γ *be a subset of the attributes of relation* R, $\gamma \subseteq \text{DOM}(R)$. *Then* $\pi_\gamma(R) = \emptyset \iff R = \emptyset$. □

Lemma 5 *If* R *and* S *are relations,* $\beta \subseteq \text{DOM}(R)$, $\gamma = \text{DOM}(S)$, *and* $h \in R \bowtie S$ *then* $h|_{\beta \cup \gamma} \in \pi_\beta(B) \bowtie S$. □

Lemma 6 *If* R *and* S *are relations,* $\alpha = \text{DOM}(R)$, $\gamma = \text{DOM}(S)$, $\alpha \supseteq \beta \supseteq \alpha \cap \gamma$, *and* $\exists h \in \pi_\beta(R) \bowtie S$ *then* $\exists h' \in R \bowtie S$ *such that* $h'|_{\beta \cup \gamma} = h$.

Proof: From $\exists h \in \pi_\beta(R) \bowtie S$ and the definition of join we have $\exists f \in \pi_\beta(R),\ g \in S$ such that $h|_\beta = f,\ h|_\gamma = g$. (Note $f|_{\beta\cap\gamma} = g|_{\beta\cap\gamma}$.) But $\exists f \in \pi_\beta(R) \Longrightarrow \exists f' \in R$ such that $f'|_\beta = f = h|_\beta$ by lemma 3. Consider the function $h' : \alpha\cup\gamma \rightarrow [K]$ that is the extension of both f' and g, i.e. $h'|_\alpha = f'$ and $h'|_\gamma = g$. This is well defined iff $f' = g$ on the intersection of their domains. Now from $\beta \supseteq \alpha\cap\gamma$ we know $\beta\cap\gamma \supseteq \alpha\cap\gamma$ and together with $g|_{\beta\cap\gamma} = f|_{\beta\cap\gamma} = (f'|_\beta)|_{\beta\cap\gamma} = f'|_{\beta\cap\gamma}$ this implies $f'|_{\alpha\cap\gamma} = g|_{\alpha\cap\gamma}$, so h' is well defined; and we have the four conditions to show $h' \in R \bowtie S$. Moreover, h' is an extension of h, and $\alpha \supseteq \beta \Longrightarrow h'|_{\beta\cup\gamma} = h$, as required. □

Lemma 7 *Let R and S be relations, $\alpha = \text{DOM}(R)$, and $\gamma = \text{DOM}(S)$. If $\alpha \supseteq \beta \supseteq \alpha\cap\gamma$ then $\pi_{\beta\cup\gamma}(R \bowtie S) = \pi_\beta(R) \bowtie S$.*

Proof: For each function h' in the L.H.S., by lemma 3 it follows that $\exists h \in R \bowtie S$, such that $h|_{\beta\cup\gamma} = h'$, and (by lemma 5) $h|_{\beta\cup\gamma} \in \pi_\beta(R) \bowtie S$, so h' is in the R.H.S.

For each function h in the R.H.S., apply lemma 6 to show $\exists h' \in R \bowtie S$ such that $h'|_{\beta\cup\gamma} = h$. But $h'|_{\beta\cup\gamma} \in \pi_{\beta\cup\gamma}(R \bowtie S)$, so h is in the L.H.S.

Thus the L.H.S. equals the R.H.S. □

1.2 Definition of SIS

In the general case of the SIS problem, you are given:

$$S = (K, \Upsilon, \mathcal{E}, r)$$

where K is a positive integer defining the *points* set $[K] = \{1, 2, 3, \ldots, K\}$,
Υ is a set of *nodes* $\{\Upsilon_1, \Upsilon_2, \ldots,\}$,
$\mathcal{E}$ is a set of (hyper)*edges* $\{\epsilon_1, \epsilon_2, \epsilon_3, \ldots\}$,
each (hyper)edge ϵ_i is the name of a set of nodes, $\eta(\epsilon_i) \subseteq \Upsilon$,
r is an assignment of a set of tuples to each edge so that $r(\epsilon)$ is a relation on the attribute set $\eta(\epsilon)$ into the point set $[K]$. Each $r(\epsilon)$ is called a *primitive relation.*

The *giant join* of this system of relations is the join of all the primitive relations $\bowtie_{\epsilon_i \in \mathcal{E}} r(\epsilon_i)$, which for brevity we write as $\bowtie_{\mathcal{E}}$. Any tuple that is a member of this giant join is called a *solution* to the system. The computational problem is to find a solution or to prove that none exists.

(In the terminology of relational databases, the nodes Υ are attributes; the points, $[K]$, are the domain of the relations (this is unfortunate terminology since the points are actually the *range* of the tuples in the relation); the set of edges $\mathcal{E}$ is the database design, and each hyperedge is a relation scheme.)

I introduce this problem as a kind of generic constraint-satisfaction problem and for this it is useful to reinterpret the terms again. S is a system of variables (the nodes Υ), each of which can take on one of a set of values (the points $[K]$); there are specifications as to which variables interact with which others (the edges $\mathcal{E}$); and the details of exactly how they interact are given as a list of constraints, i.e. a list of compatible values for interacting variables ($r(\epsilon_i)$). The computational question is thus translated as "Is there an assignment to each variable in the system that satisfies all the constraints?" It is a system for which an efficient solution-finding algorithm would have very wide applicability. Of course its importance comes from the fact that it is *NP*-complete but this implies its intractability as well.

To get any kind of leverage into understanding this problem, I employ a technique of specifying subcases by restricting the form of the gross structure in the problem, and this takes the form of specifying a class of graphs that $(\Upsilon, \mathcal{E})$ must belong to. The first S in the acronym SIS is from the word "structured".

1.3 Random SIS problems

For reasons not relevant to this discussion, I want to model real-world instances of some *NP*-complete problems as composed of (1) gross structure, plus (2) noisy details. I look at the graph $(\Upsilon, \mathcal{E})$ as specifying the gross structure of the SIS problem, and I view r as specifying the noisy details. Clearly, the two components interact very deeply. Restrictions or peculiarities of the constraints in r can override any structure we might specify in

the gross graph. But I shall try to study the effect of gross structure on the complexity of the SIS problem by ensuring that no *systematic* structure arises from the relations, r. This shall be accomplished by studying the *statistical* nature of the problem when r is generated at random.

To keep things from becoming trivial, we need to ensure that at least one solution to these random SIS instances always exists. To generate them "at random" such that we guarantee a solution, we first flip a coin to select a solution, $\mathcal{S}$, which of course is an assignment of a value to each variable in the system:

$$\mathcal{S} : \Upsilon \rightarrow [K]$$

where for each $v \in \Upsilon$, $\Pr(\mathcal{S}(v) = j) = 1/K$ independently.. Then for each (hyper)edge, ϵ, whose nodes are $\eta(\epsilon) = \{v_1, v_2, \ldots, v_m\}$, we generate a primitive relation $r(\epsilon)$ where each possible tuple, $(t_1, t_2, \ldots, t_m) \in [K]^m$, is present with independent probability

$$\Pr((t_1, t_2, \ldots, t_m) \in r(\epsilon)) = \begin{cases} 1 & \text{if } t_i = \mathcal{S}(v_i) \;\; 1 \leq i \leq m \\ p & \text{otherwise} \end{cases}$$

An instance generated this way shall be called a p-dense instance of SIS.

It helps to view the generation in this way, but the questions that follow are actually oblivious to the particular $\mathcal{S}$ chosen and so σ could just as easily be assumed to be some particular assignment like "all 1's".

1.4 The Grid-SIS Problem

In the special case we consider, each ϵ_i is always a simple edge with exactly two ends and the nodes and edges form a graph that is a square 2-D grid. Using the convenience of giving the nodes two subscripts (to represent coordinates in the grid), the node set is $\Upsilon = \{\Upsilon_{x,y} : 0 \leq x, y < n\}$ and the set of edges is $\mathcal{E} = \{\{\Upsilon_{x,y}, \Upsilon_{x+1,y}\}, \{\Upsilon_{x,y}, \Upsilon_{x,y+1}\} : 0 \leq x, y < n\}$. We shall call this problem grid-SIS. To specify an instance, we need the integers n and K, and the function $r : \mathcal{E} \rightarrow \wp([K]^2)$. ($\wp$ denotes the powerset.) See figure ?? for an example instance.

Lemma 8 *Grid-SIS is NP-complete.* □

Lemma 9 *When $K \leq 2$, grid-SIS is polynomial.* □

For future mathematical ease, we must allow ourselves to speak of infinite instances of grid-SIS and to avoid edge effects. For this we will imagine the grid to extend infinitely far in both directions of both dimensions: $\Upsilon = \{\Upsilon_{x,y} : x, y \in Z\}$, $\mathcal{E} = \{\{\Upsilon_{x,y}, \Upsilon_{x+1,y}\}, \{\Upsilon_{x,y}, \Upsilon_{x,y+1}\} : x, y \in Z\}$.

In what follows, we shall often omit mention of K, it being implicitly some fixed number. And of course for infinite instances, n will also be omitted.

2 The Order Parameter

2.1 Dynamic Programming

Dynamic programming is a technique that could be used to solve SIS. It exploits lemma 7. First note that just by the associativity and commutativity of join, we have

Lemma 10 *For any set of edges, $H \subseteq \mathcal{E}$,* $\bowtie_{\mathcal{E}} = \bowtie_{H} \bowtie \bowtie_{\mathcal{E}-H}$. □

Let a $H \subseteq \mathcal{E}$ be a set of edges, and define the *boundary* of this set to be

$$\mathrm{bnd}(H) = (\bigcup_{F \in H} \eta(F)) \cap (\bigcup_{F \in \mathcal{E}-H} \eta(F)).$$

That is, the boundary is the minimal set of nodes that interfaces between the given set of edges, H, and all the other edges in the system, $\mathcal{E} - H$.

Lemma 11 *If H is a set of edges, $\beta = \mathrm{bnd}(H)$, and γ is the set of nodes each belonging to some edge not in H,*

$$\gamma = \bigcup_{F \in \mathcal{E}-H} \eta(F),$$

then

$$\pi_\gamma(\bowtie_{\mathcal{E}}) = \pi_\beta(\bowtie_{H}) \bowtie \bowtie_{\mathcal{E}-H}.$$

Proof: Let $\alpha = \text{DOM}(\bowtie_{H})$ and $\gamma = \text{DOM}(\bowtie_{\mathcal{E}-H})$, i.e. $\beta = \alpha \cap \gamma$.

$$\begin{aligned}
\pi_\gamma(\bowtie_{\mathcal{E}}) &= \pi_\gamma(\bowtie_{H} \bowtie \bowtie_{\mathcal{E}-H}) && \text{by lemma 10} \\
&= \pi_{\beta\cup\gamma}(\bowtie_{H} \bowtie \bowtie_{\mathcal{E}-H}) && \text{since } \gamma = \beta \cup \gamma \\
&= \pi_\beta(\bowtie_{H}) \bowtie \bowtie_{\mathcal{E}-H} && \text{by lemma 7}
\end{aligned}$$

□

The decision problem underlying SIS is to find whether or not the giant join is empty. The strategy of dynamic programming is to calculate the join of just part of the system, then throw away all the attributes that are irrelevant to further calculations and continue making joins with other parts of the system. By throwing away irrelevant attributes (i.e. projecting onto the boundary), the data structure being maintained can be shrunk, often enormously. See figure 2.1. The space and time complexity of the dynamic program is $\Theta(K^{|\beta|})$ so it will be more efficient if $|\beta|$ can be kept small. When the process is finished, the program has retained a set of functions over only the last few attributes. However, for every function left there exists at least one extension of it that is a member of the complete giant join. So the residual relation is empty iff the giant join is empty (by lemma 4).

2.2 A Model of Algorithms for *NP*

We require an algorithm to find a solution for SIS, i.e. it must take an arbitrary instance of SIS as input and emit a complete list of assignments to variables of the form "$v \mapsto i$" where $v \in \Upsilon$ and $i \in [K]$. The algorithm may also emit "There is no solution" at any time, but if the algorithm halts without having proved that, then it must have emitted exactly one assignment for each variable, and they must constitute a solution. The main purpose of this description of the algorithm is to specifically allow it the option of emitting assignments even before it has determined whether or not a total solution exists. With this margin of freedom, each emitted assignment "$v \mapsto i$" will in effect constitute a theorem of the form "If a

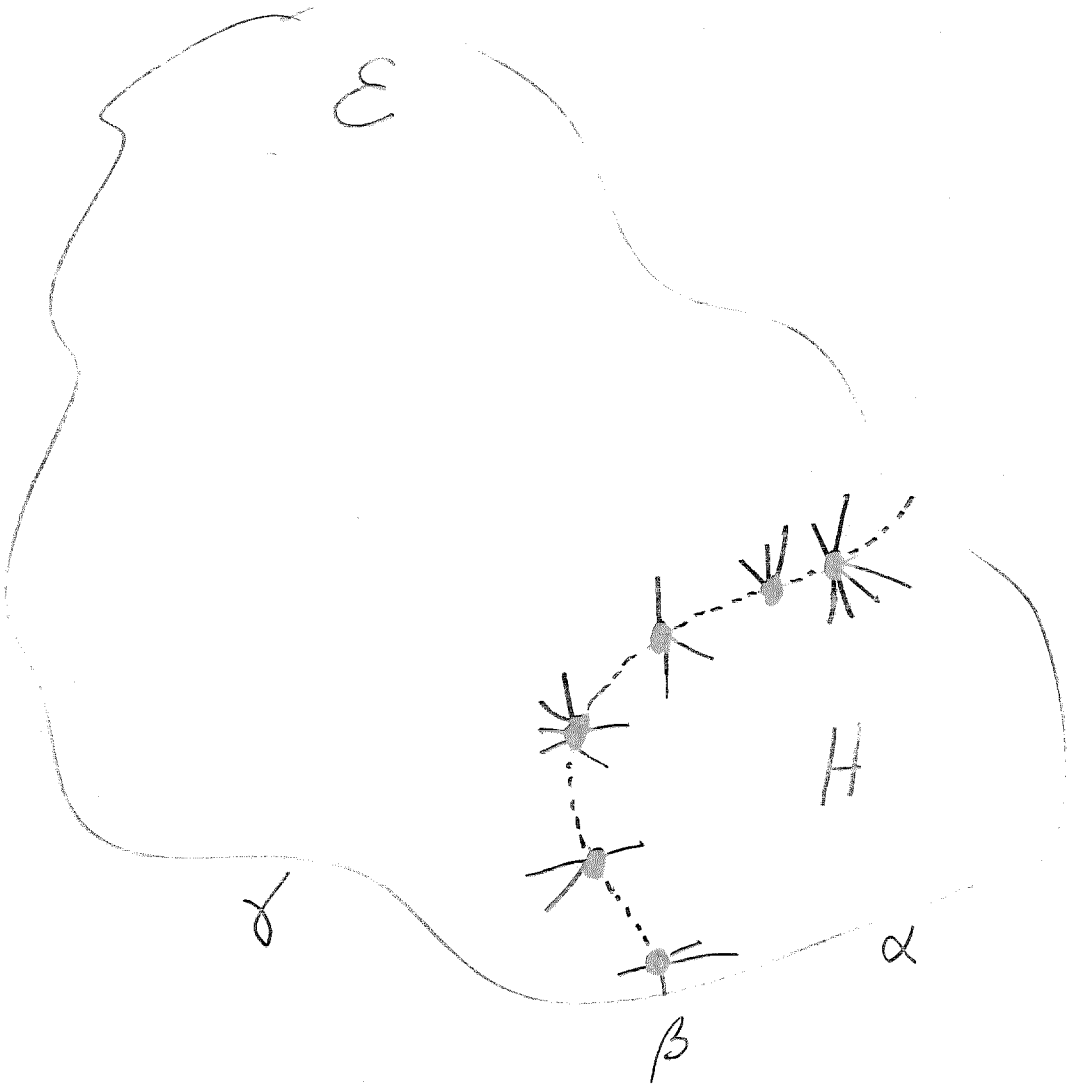

Figure 1: Illustration of dynamic programming.

solution exists, then there exists a solution wherein v has the value i."

The requirements above specify a *search* problem (rather than a *decision* problem) but we know that the search problem can be solved with just a linear number of decisions, so the search problem is not significantly harder than the decision problem.

(By the way, we might know from the way our instances will be generated that solutions are always going to exist but the algorithm is not to assume this.)

We shall be asking about the necessary running time of such an algorithm. Obviously, the time before the emission of a value for the first variable is a lower bound on the total time; so let us focus on the first emission. To minimize the running time before it is emitted, it would be very useful to consider only a limited amount of data. In an infinite-sized instance, if the first emission is to occur in a finite amount of time then it is clearly mandatory that only a finite amount of data be examined before the emission is made. And regardless of whether the instance is finite or infinite, it is necessary to be able to identify a *safe* assignment, wherein the value i given to the first variable is guaranteed not to preclude an eventual solution (if there is one).

2.3 Domination

Domination is an event in a dynamic program that can be used to detect a safe assignment of a value to a variable, even though the giant join is not completely calculated. First, observe

Lemma 12 *Let R_1, R_2 be relations on the same set of attributes. If $R_1 \supseteq R_2$, it follows that for any relation T*

$$R_1 \bowtie T \supseteq R_2 \bowtie T$$

and

$$R_1 \bowtie T \neq \emptyset \Longleftarrow R_2 \bowtie T \neq \emptyset. \qquad \square$$

We shall exploit this fact to identify values that must be part of a solution if any solutions exist. Let v be a node, H be a set of edges, $v \in \bigcup_{\epsilon \in H} \eta(\epsilon)$,

and define

$$R_i^v(H) = \pi_{\mathrm{bnd}(H)}(\sigma_{\dot{i}}^{v}(\bowtie_{H}))$$

$$R(H) = \pi_{\mathrm{bnd}(H)}(\bowtie_{H})$$

Domination of node v is the event defined w.r.t. a neighbourhood, H,

$$\mathrm{Domin}(v, H) \iff \exists i \in [K] \;\; \forall j \in [K] \;\; R_i^v(H) \supseteq R_j^v(H).$$

In this case, we say i is a *dominator* for v over H, or when the particular neighbourhood is irrelevant we say i *dominates* v.

Lemma 13 *If* S *and* T *are relations, and* $\mathrm{DOM}(S) = \mathrm{DOM}(T)$, *then*

$\forall j \in [K] \;\; S \supseteq \sigma_{\dot{j}}^{v}(T) \iff S = T.$ □

Lemma 14 *If* S *and* T *are relations, and* $\mathrm{DOM}(S) = \beta \subseteq \mathrm{DOM}(T)$ *then*

$\forall j \in [K] \;\; S \supseteq \pi_\beta(\sigma_{\dot{j}}^{v}(T)) \iff S = \pi_\beta(T).$ □

Consequently we have the equivalent definition of domination:

Lemma 15 $R_i^v(H) = R(H) \iff i$ *is a dominator for* v *over* H. □

Lemma 16 *Say* i *dominates* v. *Then if there is any solution tuple in the giant join, there is a solution in which attribute* v *has value* i. *That is, it is safe to emit "*$v \mapsto i$*".* □

Lemma 17 *Any infallible algorithm that emits the assignments "*$v_1 \mapsto i_1$*", "*$v_2 \mapsto i_2$*",..., "*$v_m \mapsto i_m$*", must by that stage in its computation have compiled a proof that* $(i_1, i_2, \ldots, i_m)$ *dominates* $(v_1, v_2, \ldots, v_m)$.

Proof: If not then there could be some relation on the boundary which is not compatible with the emitted assignments and yet which might have to be part of any solution to the system. □

Lemma 18 *If $(i_1, i_2, \ldots, i_m)$ dominates $(v_1, v_2, \ldots, v_m)$ over H, then for all $j \in \{1, 2, \ldots, m\}$, i_j dominates v_j over H.*

Proof: Let $S = \bowtie_H$ and $\beta = \mathrm{bnd}(H)$. For each $j \in \{1, 2, \ldots, m\}$,

$$\pi_\beta(\sigma_{\frac{v_j}{i_j}}(S)) \supseteq \pi_\beta(\sigma_{\frac{v_1 \ v_2 \ \cdots \ v_m}{i_1 \ i_2 \ \cdots \ i_m}}(S)) = \pi_\beta(S) \supseteq \pi_\beta(\sigma_{\frac{v_j}{i_j}}(S)).$$

(The inclusions are by lemma 1, the equality by domination.) Hence the last 2 quantities are equal, and $R_{i_j}^{v_j} \supseteq R(H)$, as required. □
Is the converse true?

These last lemmas reveal the motivation for studying domination. Summarizing them, we find something fundamental about all algorithms:

Theorem 19 *Proving domination is both necessary and sufficient for solving any problem in NP.* □

In cases where the system is small enough or where some regularity makes the calculation of the giant join feasible, then this claim reduces to the true but limp statement that to solve the problem you must be able to find a solution. But when the system is too big for that (and it easily can be), then the theorem becomes potent.

Lemma 20 *Let H_1, H_2 be sets of edges.*
If $H_1 \subseteq H_2$ then $\mathrm{Domin}(v, H_1) \Longrightarrow \mathrm{Domin}(v, H_2)$. □

2.4 The Order Parameter

We want to run a dynamic program on grid-SIS using successively larger neighbourhoods until we find one where domination occurs. Define the d-neighbourhood of a node as the set of edges that are within distance d of a node. For specificity in what follows, we shall be concerned about finding a dominator for the node $\Upsilon_{0,0}$ in infinite instances. Let H_d be the d-neighbourhood of $\Upsilon_{0,0}$, so for all $d \geq 1$

$$H_d = \{\{\Upsilon_{i,j-1}, \Upsilon_{i,j}\}, \{\Upsilon_{i-1,j}, \Upsilon_{i,j}\} : |i| + |j| \leq d\}$$

Let $\beta_d = \text{bnd}(H_d)$, and note $\beta_d = \{\Upsilon_{i,j} : |i| + |j| = d\}$ and $|\beta_d| = 4d$. We shall run a dynamic program on grid-SIS using the sequence of neighbourhoods defined by $\langle H_1, H_2, \ldots \rangle$. But when we do this, the boundaries $\langle \beta_1, \beta_2, \ldots \rangle$ will be growing successively larger. Recalling the reason for keeping the boundaries as small as possible, we are left with the question of how large d will become before domination occurs.

In an instance of p-dense grid-SIS, define the order parameter

$$D(p, K) = \text{Expected}(\min d \ni \text{Domin}(\Upsilon_{0,0}, H_d)).$$

3 The Statistical Nature of Grid-SIS

3.1 Two-Phase Conjecture

The primary feature of p-dense grid-SIS that I want to illuminate is stated in the following conjecture. It tells us that the great majority of instances of this particular *NP*-complete problem are so tangled up that no algorithm will be able to emit early assignments for most variables. There are some *single* variables that cannot be safely assigned a value until the algorithm has compiled all the data—i.e. until it can assign values to *every* variable.

Conjecture 21 *There exists a function, $K^* : [0,1] \to \Re^+$, which is continuous, concave upward, and approaching ∞ as $p \to 1$ or as $p \to 0$ such that*

$$\forall p \in [0,1] \;\; K > K^*(p) \iff D(p, K) = \infty.$$

Corollary 22 *For all sufficiently large K, there exist two constants $p_K^{(\ell)}, p_K^{(h)}$ where $0 < p_K^{(\ell)} < p_K^{(h)} < 1$ such that for all p,*

$$p_K^{(\ell)} < p < p_K^{(h)} \iff D(p, K) = \infty.$$

and

$$p_{K+1}^{(\ell)} < p_K^{(\ell)} < p_K^{(h)} < p_{K+1}^{(h)}.$$

□