coNP

La classe coNP contiene i complementi dei linguaggi in NP.

$$\text{coNP} = \{ \neg L \mid L \text{ è in NP} \}$$

Esempi di problemi in coNP:

UnSAT = {ϕ | ϕ è falsa per ogni assegnamento alle variabili }

NoCLIQUE = {<G,k> | G è un grafo non diretto senza un k-clique}

coNP: altro esempio

TAUT = {ϕ | ϕ è vera per ogni assegnamento alle variabili }

TAUT è in coNP?

sì, perché unTAUT è in NP (prova analoga a quella per SAT!)

Verificatore per unTAUT:

Input <<φ>,c>

1. **verifica che c sia un assegnamento di valori di verità per tutte le variabili di φ**
2. **verifica se φ risulta falsa e in tal caso accetta, altrimenti rifiuta.**

coNP-completi: un esempio

TAUT = {ϕ | ϕ è vera per ogni assegnamento alle variabili }

TAUT è coNP-completo?

SAT è NP-completo e quindi anche NP-hard, quindi unSAT è coNP-hard

Possiamo provare che unSAT $\leq_p$ TAUT, così anche TAUT è coNP-hard, inoltre abbiamo visto che TAUT è in coNP, quindi possiamo concludere che TAUT è NP-completo.

La riduzione:

se consideriamo f: $\phi \mapsto \neg\phi$ osserviamo che

$$\phi \text{ è in unSAT} \iff \neg\phi \text{ è in TAUT}$$

Se NP=coNP allora TAUT e unSAT sarebbero in NP.

In particolare avremmo un verificatore polinomiale per le tautologie.

PRIMES and COMPOSITES

E' facile provare che COMPOSITES è in NP.
COMPOSITES={x | x=p*q, per p,q interi e p,q≠1}
Basta dare in input a un verificatore gli interi x e p come istanza del problema e certificato.

not COMPOSITES = PRIMES = {x | x è un numero primo}.
Quindi PRIMES è in coNP

La prova che PRIMES è in NP è meno semplice (teorema di PRATT, 1975)
Quindi PRIMES NP ∩ coNP .

Non si dimentichi che PRIMES è in P.

P, NP, coNP e EXPTIME

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME}$$

dove

$$\mathbf{EXPTIME} = \bigcup \mathbf{TIME}(2^{n^k})$$

e

$$\mathbf{P} \subseteq \mathbf{coNP} \subseteq \mathbf{EXPTIME}$$

Prova P ⊆ coNP. Infatti P=coP (l'insieme dei complementi dei linguaggi in P) e P⊆NP ⇒ coP⊆coNP, infatti se P⊆NP allora L∈ P⇒ L ∈ NP⇒¬ L ∈ coNP, quindi P=coP⊆coNP.

P, NP e coNP

$P \subseteq NP \cap coNP$ ma $P = NP \cap coNP$?

Non noto (si suppone di no)

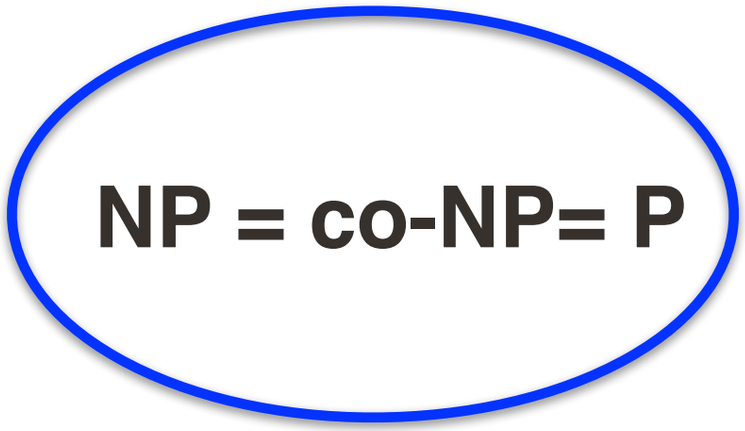
Se $P = NP$ allora $NP = coNP$, perché $P = coP$, o equivalentemente è vero che

se $NP \neq coNP$ allora $NP \neq P$

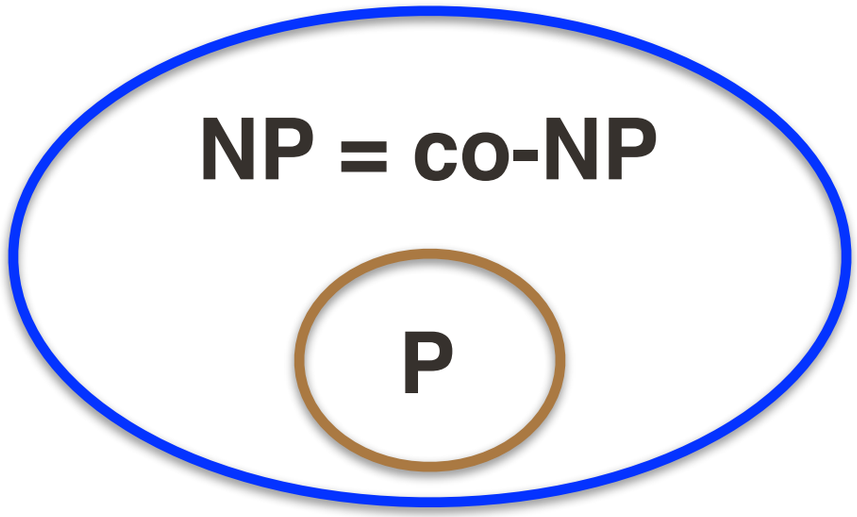
Ma può essere che sia vero sia $NP = coNP$ che $NP \neq P$!

Se un linguaggio in $NP \cap coNP$ fosse NP-completo, allora $NP = coNP$

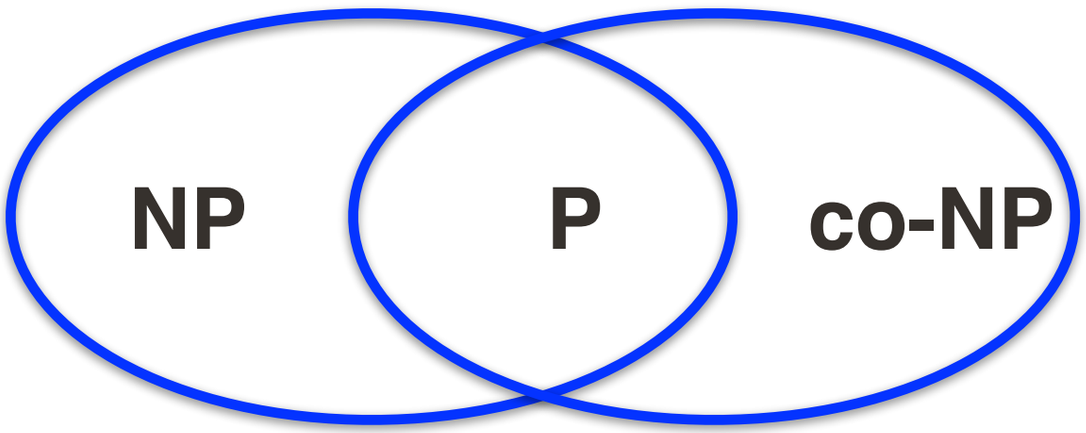
P, NP e coNP : i possibili scenari



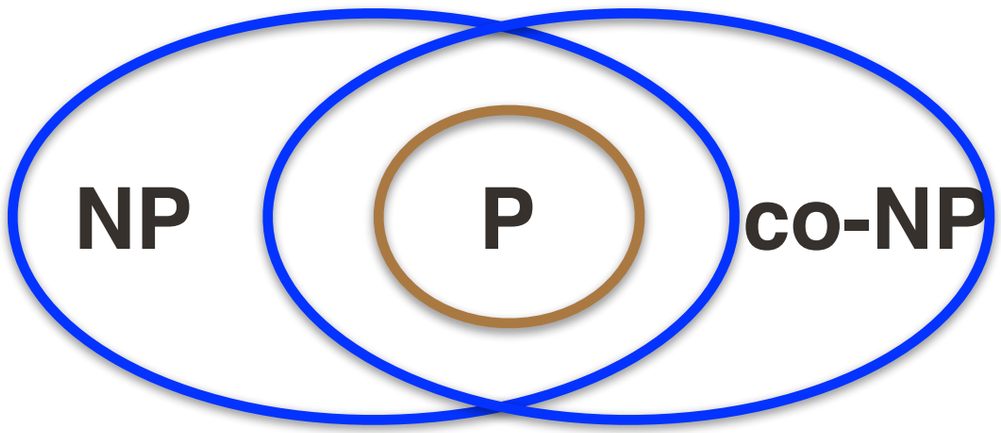
a



b



c



d (il più probabile)

NP ∩ coNP

Per i problemi in NP ∩ coNP disponiamo di un verificatore polinomiale sia per le istanze sì che per le istanze no.

Quindi sono problemi per i quali può essere ragionevole aspettarsi l'appartenenza a P e che difficilmente possono essere NP-completi.

Per esempio FACTOR.

FACTOR

FACTOR: Dati due interi positivi z e U, c'è un divisore intero non banale di z minore di U?

Istanza 1. z = 23.536.481.273 e U = 110.000.
risposta SI: z = 104.729 x 224.737.

Istanza 2. z = 23.536.481.273 e U = 100.000.
risposta NO : 104.729 x 224,737 è la fattorizzazione in fattori primi di z.

Istanza 3. z = 23.536.481.277 e U = 23.536.481.277
risposta NO : z è un numero primo.

Algoritmo deterministico per FACTOR

CercaFattori(x,U)
prec: (U≤√x)

Perché un numero z è composto se e soltanto se ha un fattore minore o uguale di $\sqrt{z}$.

```
k = 2
while k ≤ U
      if x mod k = 0 then esci e  "SI"
     else k = k +1
"NO"
```

L'algoritmo CercaFattori nel caso peggiore esegue $\sqrt{x}$ volte il ciclo. Quindi se n è la lunghezza della rappresentazione binaria di x (quindi x è minore di 2^{n+1}) allora l'algoritmo ha una complessità in $O(2^{n/2})$. Infatti considera tutti i numeri binari tra 2 e $2^{n/2}$.
Per esempio se $x = 2^{64} = 18\ 446\ 744\ 073\ 709\ 551\ 616$ ci sono $2^{32} = 4\ 294\ 967\ 296$ numeri da verificare,
se $x = 2^{1024}$, cioè circa 10^{308}, ci sono circa 10^{154} numeri da verificare.
Si stima che l'età dell'universo sia circa 10^{13} e il numero di particella sia minore di 10^{100}.

Algoritmo non deterministico e polinomiale per FACTOR

CercaFattoriNonDet(x,U)

prec: (U≤√x)

“non deterministicamente scegli un numero k tra 2 e U”

if x mod k = 0 then “SI” else “NO”

L’algoritmo CercaFattoriNondet lavora in tempo polinomiale

Riduzioni a FACTOR

PRIMES $\equiv_P$ COMPOSITES $\leq_P$ FACTOR.

FACTOR $\leq_P$ PRIMES ?

Si pensa di no!
Il sistema crittografico RSA è basato sulla dicotomia tra le complessità dei due problemi:

- **per la codifica si moltiplicano grandi primi generati efficientemente**
- **per rompere il sistema si deve fattorizzare**
- **efficientemente**

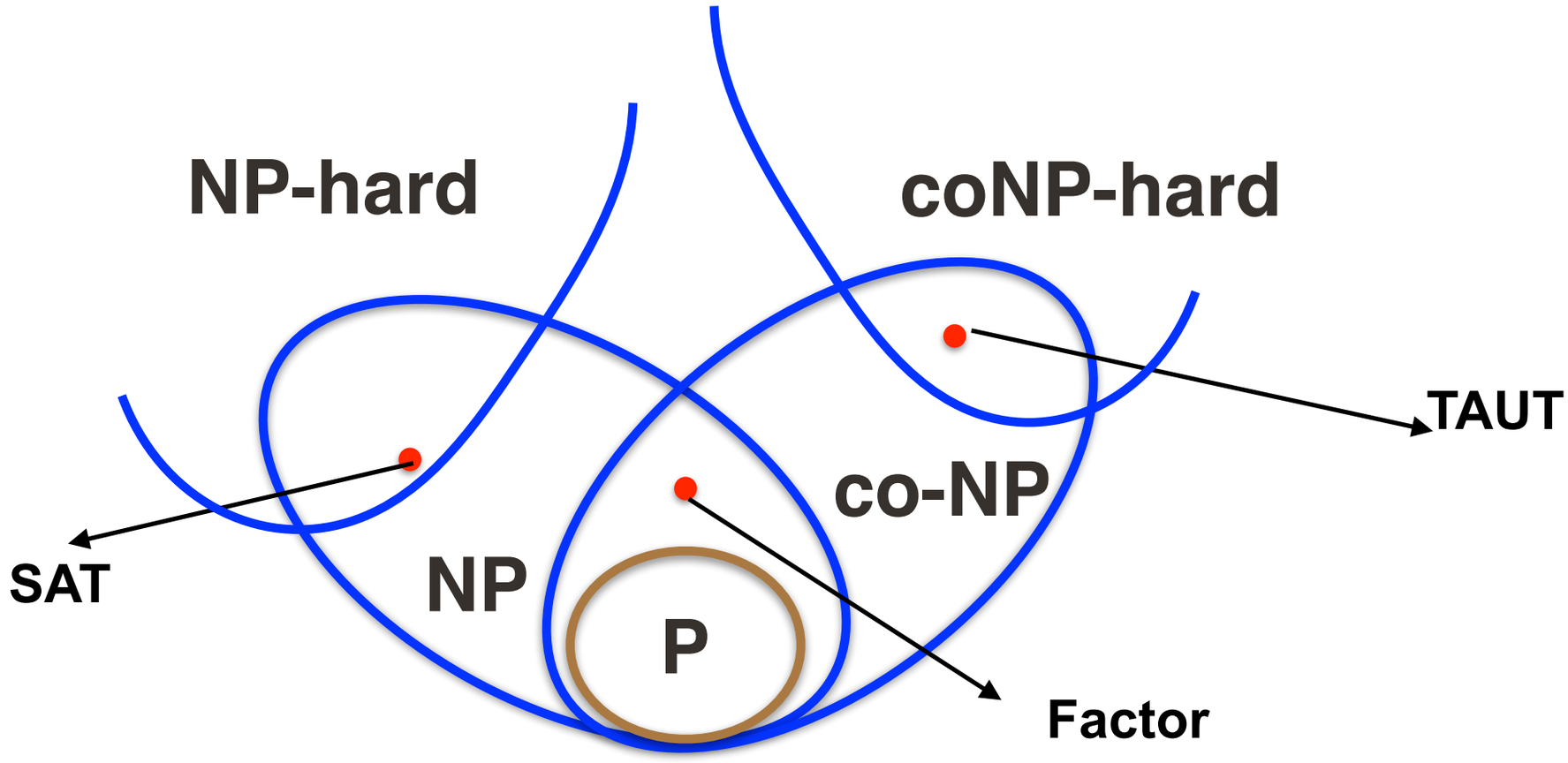
FACTOR in NP e in coNP

FACTOR: Dati due interi positivi z e U, c'è un divisore non banale di z minore di U?

FACTOR è in NP in virtù dell'algoritmo che abbiamo visto o equivalentemente perché si può costruire un verificatore polinomiale:
Il verificatore riceve in input z e il certificato, un fattore d ≥ 2 e d ≤ U, e verifica se z=d(N/d).

Un verificatore polinomiale per not FACTOR riceve in input z e una sequenza di numeri maggiori di U. La sequenza è lunga al più lg z perchè la più lunga scomposizione in fattori primi di un numero si ottiene considerando il più piccolo primo e $2^{\lg z} = z$. Questo garantisce la polinomialità della lunghezza del certificato. Con il test di primalità si verifica se i numeri dati sono primi, mentre basta moltiplicare i numeri e vedere se il risultato è z per verificare se si tratta della scomposizione in numeri primi di z.

FACTOR in NP e in coNP



Chiamiamo FACTORIZE il problema di trovare la fattorizzazione in interi primi.
Possiamo provare che FACTOR $\equiv_P$ FACTORIZE:
FACTORIZE $\leq_P$ FACTOR:
se esiste un divisore di x minore di U, perchè FACTOR(x,U) dà vero, usiamo la ricerca binaria per trovarlo e poi dividiamo x per il fattore trovato e ripetiamo
FACTOR $\leq_P$FACTORIZE: banale

numeri RSA: $n = p_1 p_2$

Un numero RSA è il prodotto di due primi. Il più grande RSA fattorizzato: RSA-768 ha 232 cifre decimali (768 bits), e era fattorizzato nel 2009, da Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Joppe W. Bos, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, e Paul Zimmermann.[35]

RSA-768 =
123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3

RSA-768 =
33478071698956898786044169848212690817704794983713768568912431388982883793 878002287614711652531743087737814467999489
×
36746043666799590428244633799627952632279158164343087642676032283815739666511279233734171433968102700927987363089 17

IL tempo di calcolo

Il tempo CPU speso per trovare i fattori di RSA-768 (768 bits), facendo lavorare in parallelo alcuni computer è pari a circa 2000 anni di calcolo su un singolo calcolatore veloce.

In pratica i numeri coinvolti nelle applicazioni basate sull' RSA sono lunghi da 1024 a 4096 bits.

Da Wikipedia:
https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

Algoritmo quantistico polinomiale per FACTOR

Nel 1994 Peter Shor (MIT) ha sviluppato un algoritmo in grado di risolvere FACTOR in tempo polinomiale con un calcolatore quantistico.

Dattani and Bryans nel 2012, usando l'algoritmo di Shor, hanno fattorizzato il più grande numero fino ad ora:
56153 = 241 x 233.

The beginning of the end for encryption schemes?

Questo era il titolo della notizia data dal MIT, nelle news, della pubblicazione nel marzo del 2016 su Science dell'articolo di

Thomas Monz, Daniel Nigg, Esteban A. Martinez, Matthias F. Brandl, Philipp Schindler, Richard Rines, Shannon X. Wang, Isaac L. Chuang, Rainer Blatt

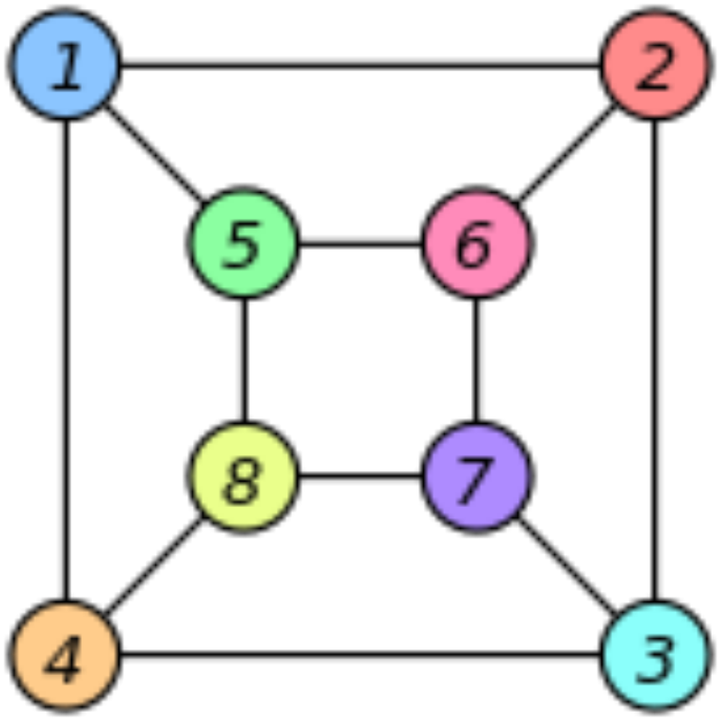
dal titolo

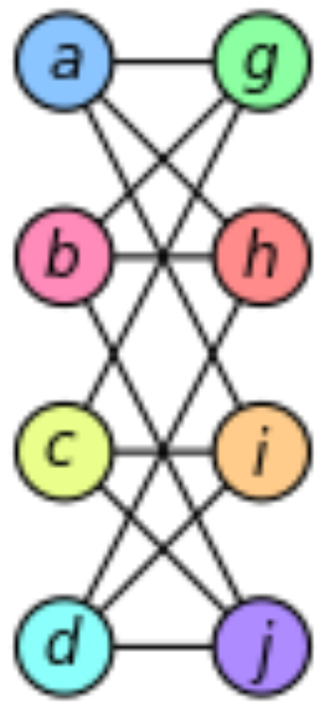
Realization of a scalable Shor algorithm

Isomorfismo tra grafi

GI è il problema di determinare se due grafi G=(V,E) e G'=(V',E') sono isomorfi, cioè se esiste una funzione biettiva f: V -> V' tale che se {u,v} è un arco di G, allora {f(u),f(v)} è un arco di G'. Informalmente il problema può porsi come il problema di stabilire se due grafi disegnati diversamente sono lo stesso grafo.

Esempio (preso da Wikipedia):





I due grafi sono isomorfi, basta associare vertici dell'uno a quelli dell'altro con lo stesso colore.

Isomorfismo tra grafi: un problema in NP

GI = {(G,G') l G e G' sono grafi non diretti e isomorfi tra loro} è in NP

Infatti non è difficile immaginare un verificatore A che prende in input due grafi G=(V,E) e G'=(V',E') e un potenziale isomorfimo. L'isomorfismo può essere rappresentato da una sequenza di n coppie in VxV', dove n è il numero dei vertici dei due grafi.
Si verifica se le coppie sono tutte distinte e senza ripetizioni, cioè se la funzione così rappresentata è una biezione. Si verifica se per ogni arco {u,v} in G c'è l'arco {u',v'} in G' dove (u,u') e (v,v') sono nella sequenza di coppie fornita in input. Se è così dopo la verifica che in G' non ci siano archi non controllati il verificatore può dare risposta positiva.
Questo lavoro è fatto in tempo polinomiale.

Isomorfismo tra grafi: algoritmi

L'algoritmo migliore, risultato recente, dicembre 2015 poi revisionato in gennaio 2017, e non ancora pubblicato di Làszlò Babai (premio Knuth 2015) è di complessità quasi polinomiale, cioè in $2^{O((\lg n)^c)}$, per una costante c>0.

L'algoritmo migliore precedente, sempre di Babai, con Luks, è stato presentato al FOCS (IEEE Symposium on Foundations of Computer Science) nel 1983 e ha complessità in $2^{O(\sqrt{n} \lg n)}$

Per le applicazioni c'è un'euristica che lavora molto bene i cui autori sono Brendan McKay e Adolfo Piperno, che la descrivono in un articolo apparso sul Journal of Symbolic Computation, nel 2014.
Altre informazioni: http://pallini.di.uniroma1.it

Isomorfismo tra grafi è NP-completo?

Non si sa.
Quindi GI è un problema non noto essere in P e nemmeno NP-completo.

Se fosse un problema non in P e non NP-completo sarebbe un esempio concreto e "naturale" di un problema intermedio la cui esistenza è stata dimostrata da Ladner (pubblicato nel Journal of the ACM (JACM) nel 1975): se P≠NP allora esiste un problema in NP che non è in P né in NP-completo.

Questi problemi sono chiamati NP-intermedi. C'è una lista di problemi considerati buoni candidati a entrare nella classe degli NP-intermedi; tra questi problemi c'è stato PRIMES, fino al 2002, ora comprende per esempio sia GI che FACTOR.
Ora ci si aspetta che GI esca dalla lista, per essere collocato in P.