Introduzione ad Algoritmi e Teoria della Complessità

Algoritmi Probabilistici

Piero A. Bonatti

Quantum Computation

9/2/2023

Introduzione agli algoritmi probabilistici

Idea: Il nondeterminismo può essere sfruttato anche in altro modo:

- invece di programmare la MdT in modo che abbia almeno un run di accettazione se e solo se la risposta al problema è “yes”
- programmarla in modo che la *maggioranza* dei run dia la risposta corretta
- così, invece di chiedere all'oracolo di guidarci alla soluzione corretta
- implementiamo i **choose** con una scelta casuale
- la probabilità di una risposta corretta è maggiore di quella di una risposta errata (se i run sono equiprobabili)
- per ridurre la probabilità di errore, possiamo eseguire l'algoritmo più volte e decidere sulla base della maggioranza delle risposte

Introduzione agli algoritmi probabilistici

Idea: Il nondeterminismo può essere sfruttato anche in altro modo:

- invece di programmare la MdT in modo che abbia almeno un run di accettazione se e solo se la risposta al problema è “yes”
- programmarla in modo che la *maggioranza* dei run dia la risposta corretta
- così, invece di chiedere all’oracolo di guidarci alla soluzione corretta
- implementiamo i **choose** con una scelta casuale
- la probabilità di una risposta corretta è maggiore di quella di una risposta errata (se i run sono equiprobabili)
- per ridurre la probabilità di errore, possiamo eseguire l’algoritmo più volte e decidere sulla base della maggioranza delle risposte

Introduzione agli algoritmi probabilistici

Idea: Il nondeterminismo può essere sfruttato anche in altro modo:

- invece di programmare la MdT in modo che abbia almeno un run di accettazione se e solo se la risposta al problema è "yes"
- programmarla in modo che la *maggioranza* dei run dia la risposta corretta
- così, invece di chiedere all'oracolo di guidarci alla soluzione corretta
- implementiamo i **choose** con una scelta casuale
- la probabilità di una risposta corretta è maggiore di quella di una risposta errata (se i run sono equiprobabili)
- per ridurre la probabilità di errore, possiamo eseguire l'algoritmo più volte e decidere sulla base della maggioranza delle risposte

Introduzione agli algoritmi probabilistici

Idea: Il nondeterminismo può essere sfruttato anche in altro modo:

- invece di programmare la MdT in modo che abbia almeno un run di accettazione se e solo se la risposta al problema è "yes"
- programmarla in modo che la *maggioranza* dei run dia la risposta corretta
- così, invece di chiedere all'oracolo di guidarci alla soluzione corretta
- implementiamo i **choose** con una scelta casuale
- la probabilità di una risposta corretta è maggiore di quella di una risposta errata (se i run sono equiprobabili)
- per ridurre la probabilità di errore, possiamo eseguire l'algoritmo più volte e decidere sulla base della maggioranza delle risposte

Introduzione agli algoritmi probabilistici

Idea: Il nondeterminismo può essere sfruttato anche in altro modo:

- invece di programmare la MdT in modo che abbia almeno un run di accettazione se e solo se la risposta al problema è "yes"
- programmarla in modo che la *maggioranza* dei run dia la risposta corretta
- così, invece di chiedere all'oracolo di guidarci alla soluzione corretta
- implementiamo i **choose** con una scelta casuale
- la probabilità di una risposta corretta è maggiore di quella di una risposta errata (se i run sono equiprobabili)
- per ridurre la probabilità di errore, possiamo eseguire l'algoritmo più volte e decidere sulla base della maggioranza delle risposte

Introduzione agli algoritmi probabilistici

Idea: Il nondeterminismo può essere sfruttato anche in altro modo:

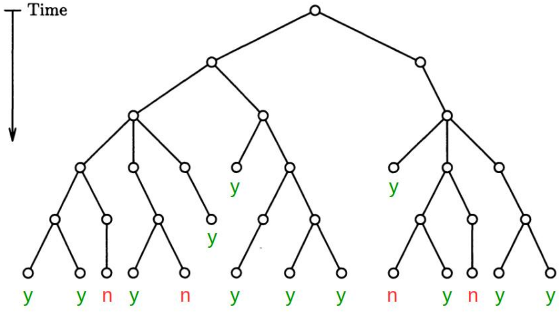
- invece di programmare la MdT in modo che abbia almeno un run di accettazione se e solo se la risposta al problema è "yes"
- programmarla in modo che la *maggioranza* dei run dia la risposta corretta
- così, invece di chiedere all'oracolo di guidarci alla soluzione corretta
- implementiamo i **choose** con una scelta casuale
- la probabilità di una risposta corretta è maggiore di quella di una risposta errata (se i run sono equiprobabili)
- per ridurre la probabilità di errore, possiamo eseguire l'algoritmo più volte e decidere sulla base della maggioranza delle risposte

Introduzione agli algoritmi probabilistici

Idea: Il nondeterminismo può essere sfruttato anche in altro modo:

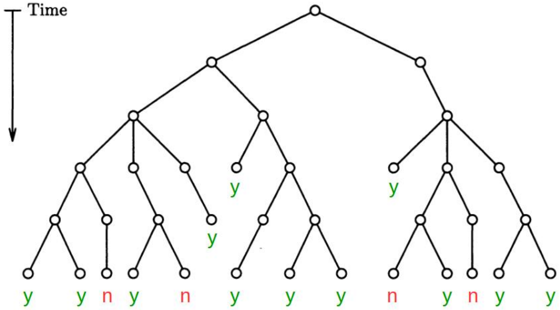
- invece di programmare la MdT in modo che abbia almeno un run di accettazione se e solo se la risposta al problema è "yes"
- programmarla in modo che la *maggioranza* dei run dia la risposta corretta
- così, invece di chiedere all'oracolo di guidarci alla soluzione corretta
- implementiamo i **choose** con una scelta casuale
- la probabilità di una risposta corretta è maggiore di quella di una risposta errata (se i run sono equiprobabili)
- per ridurre la probabilità di errore, possiamo eseguire l'algoritmo più volte e decidere sulla base della maggioranza delle risposte

Esempio 1



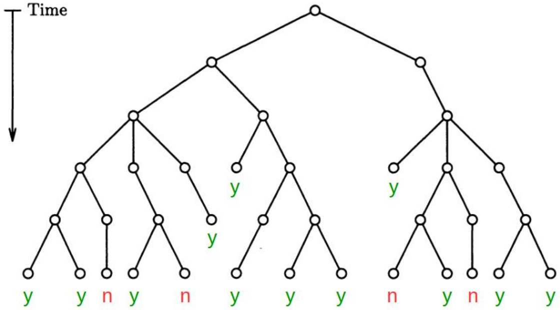
- Per questa MdT e questo input, 12 computazioni su 16 restituiscono “yes”
- Accettazione “a maggioranza”: “si”

Esempio 1



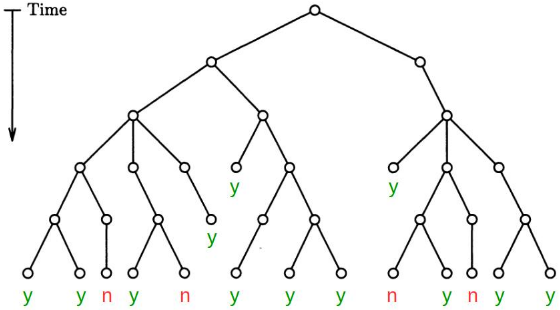
- Verifica probabilistica: con selezione dei run equiprobabile
  - la probabilità che al primo tentativo esca "yes" è $12/16 = 75\%$
  - la probabilità che in 3 tentativi escano almeno 2 "yes" è $> 84\%$
  - la probabilità che in 5 tentativi escano almeno 3 "yes" è $> 89\%$
- Quanti tentativi servirebbero per superare - che so - il 99%?
  - polinomiali o esponenziali nell'input?

Esempio 1



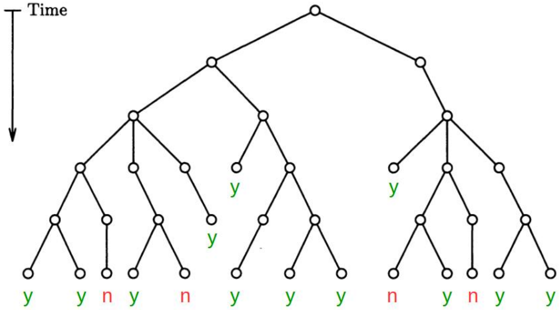
- Verifica probabilistica: con selezione dei run equiprobabile
  - la probabilità che al primo tentativo esca “yes” è $12/16 = 75\%$
  - la probabilità che in 3 tentativi escano almeno 2 “yes” è $> 84\%$
  - la probabilità che in 5 tentativi escano almeno 3 “yes” è $> 89\%$
- Quanti tentativi servirebbero per superare - che so - il 99%?
  - polinomiali o esponenziali nell’input?

Esempio 2



- Caso più generale: ogni run ha una probabilità diversa di essere generato (succederà coi computer quantistici)
- Se ogni run che termina in "no" ha probabilità 5 volte maggiore di ogni run che termina in "yes",
- allora la probabilità di estrarre un "no" al primo tentativo diventa il 62% (e cresce col numero di tentativi).

Esempio 2



- Caso più generale: ogni run ha una probabilità diversa di essere generato (succederà coi computer quantistici)
- Se ogni run che termina in "no" ha probabilità 5 volte maggiore di ogni run che termina in "yes",
- allora la probabilità di estrarre un "no" al primo tentativo diventa il 62% (e cresce col numero di tentativi).

Riassumendo

Sia che i run siano equiprobabili, sia che abbiano probabilità diverse

- Programmare un algoritmo probabilistico (MdT nondeterministica) significa fare in modo che le risposte corrette siano più probabili di quelle errate
- Ripetendo l'algoritmo più volte e prendendo come risposta finale quella più frequente abbassiamo la probabilità di errore
- Naturalmente il tempo totale impiegato dall'algoritmo è quello del singolo run moltiplicato per il numero di ripetizioni

Riassumendo

Sia che i run siano equiprobabili, sia che abbiano probabilità diverse

- Programmare un algoritmo probabilistico (MdT nondeterministica) significa fare in modo che le risposte corrette siano più probabili di quelle errate
- Ripetendo l'algoritmo più volte e prendendo come risposta finale quella più frequente abbassiamo la probabilità di errore
- Naturalmente il tempo totale impiegato dall'algoritmo è quello del singolo run moltiplicato per il numero di ripetizioni

Tema di queste slides

- Formalizzeremo queste intuizioni
  - introducendo le classi probabilistiche **PP**, **BPP**, **BQP**
  - **BQP** formalizza la classe di problemi risolubili “velocemente” da computer quantistici
- Studieremo le relazioni tra queste classi e quelle più tradizionali per prevedere quali problemi verranno resi “facili” dai computer quantistici e quali no

MdT nondeterministiche "standardizzate"

- In questo ramo della Complexity Theory, si adottano alcune assunzioni sulle MdT nondeterministiche
  - non cambiano la potenza espressiva delle MdT
  - non cambiano la complessità dei problemi (almeno da **P** in su)

1. Le assumiamo *precise*
   - tutte le computazioni terminano esattamente nello stesso numero di passi
2. ogni stato ha esattamente 2 scelte nondeterministiche

- Chiaramente questo facilita il calcolo delle probabilità dei run
  - gli alberi di computazione sono alberi binari perfettamente bilanciati
  - i run (rami) sono $2^{f(n)}$ dove $f(n)$ è la profondità dell'albero / lunghezza dei run / il tempo di computazione della MdT nondeterministica

MdT nondeterministiche "standardizzate"

- In questo ramo della Complexity Theory, si adottano alcune assunzioni sulle MdT nondeterministiche
  - non cambiano la potenza espressiva delle MdT
  - non cambiano la complessità dei problemi (almeno da **P** in su)

1. Le assumiamo *precise*
   - tutte le computazioni terminano esattamente nello stesso numero di passi
2. ogni stato ha esattamente 2 scelte nondeterministiche

- Chiaramente questo facilita il calcolo delle probabilità dei run
  - gli alberi di computazione sono alberi binari perfettamente bilanciati
  - i run (rami) sono $2^{f(n)}$ dove $f(n)$ è la profondità dell'albero / lunghezza dei run / il tempo di computazione della MdT nondeterministica

MdT nondeterministiche “standardizzate”

- In questo ramo della Complexity Theory, si adottano alcune assunzioni sulle MdT nondeterministiche
  - non cambiano la potenza espressiva delle MdT
  - non cambiano la complessità dei problemi (almeno da **P** in su)

1. Le assumiamo *precise*
   - tutte le computazioni terminano esattamente nello stesso numero di passi
2. ogni stato ha esattamente 2 scelte nondeterministiche

- Chiaramente questo facilita il calcolo delle probabilità dei run
  - gli alberi di computazione sono alberi binari perfettamente bilanciati
  - i run (rami) sono $2^{f(n)}$ dove $f(n)$ è la profondità dell'albero / lunghezza dei run / il tempo di computazione della MdT nondeterministica

MdT nondeterministiche “standardizzate”

- In questo ramo della Complexity Theory, si adottano alcune assunzioni sulle MdT nondeterministiche
  - non cambiano la potenza espressiva delle MdT
  - non cambiano la complessità dei problemi (almeno da **P** in su)

1. Le assumiamo *precise*
   - tutte le computazioni terminano esattamente nello stesso numero di passi
2. ogni stato ha esattamente 2 scelte nondeterministiche

- Chiaramente questo facilita il calcolo delle probabilità dei run
  - gli alberi di computazione sono alberi binari perfettamente bilanciati
  - i run (rami) sono $2^{f(n)}$ dove $f(n)$ è la profondità dell'albero / lunghezza dei run / il tempo di computazione della MdT nondeterministica

La classe PP

Probabilistic Polynomial Time

Definizione

$L \in$ **PP** sse esiste una MdT N (standardizzata) che termina in tempo polinomiale e tale che

$$x \in L \text{ sse } N \text{ accetta } x \text{ “a maggioranza”}$$

ovvero più della metà delle computazioni di N per l'input x terminano in “yes”

Consideriamo i run equiprobabili, quindi l'accettazione a maggioranza implica che la risposta corretta è più probabile

La classe **PP**

- Un problema completo per **PP**:

MAJSAT

Data una espressione booleana ϕ, è vero che la maggioranza dei suoi truth assignment la soddisfa?

Relazioni con **NP**

Teorema 11.3

NP $\subseteq$ **PP**

- Si pensa che il contrario non sia vero
  - la ragione si fonda su caratterizzazione di **NP** con certificati di lunghezza polinomiale

Dimostrazione che **NP** $\subseteq$ **PP** (Teorema 11.3)

- **(cenni)** Prendete un qualunque $L \in$ **NP**, quindi deciso da qualche MdT N (standardizzata) in tempo polinomiale $p(n)$,
- per dimostrare $L \in$ **PP**, trasformate N in una N' che
  - accetta a maggioranza.
  - risponde come N

1. Introducete un nuovo stato iniziale con 2 scelte nondeterministiche:
   - eseguire N
   - continuare con scelte nondeterministiche binarie per $p(n)$ passi, terminando sempre in "yes"
2. Ciascuna scelta iniziale porta a $2^{p(n)}$ computazioni
3. N' accetta a maggioranza sse almeno $2^{p(n)} + 1$ run accettano x, sse almeno 1 run di N accetta

QED

Dimostrazione che **NP** $\subseteq$ **PP** (Teorema 11.3)

- **(cenni)** Prendete un qualunque $L \in$ **NP**, quindi deciso da qualche MdT N (standardizzata) in tempo polinomiale $p(n)$,
- per dimostrare $L \in$ **PP**, trasformate N in una N' che
  - accetta a maggioranza.
  - risponde come N

1. Introducete un nuovo stato iniziale con 2 scelte nondeterministiche:
   - eseguire N
   - continuare con scelte nondeterministiche binarie per $p(n)$ passi, terminando sempre in "yes"
2. Ciascuna scelta iniziale porta a $2^{p(n)}$ computazioni
3. N' accetta a maggioranza sse almeno $2^{p(n)} + 1$ run accettano x, sse almeno 1 run di N accetta

QED

Dimostrazione che NP ⊆ PP (Teorema 11.3)

- **(cenni)** Prendete un qualunque $L \in$ **NP**, quindi deciso da qualche MdT N (standardizzata) in tempo polinomiale $p(n)$,
- per dimostrare $L \in$ **PP**, trasformate N in una N' che
  - accetta a maggioranza.
  - risponde come N

1. Introducete un nuovo stato iniziale con 2 scelte nondeterministiche:
   - eseguire N
   - continuare con scelte nondeterministiche binarie per $p(n)$ passi, terminando sempre in "yes"
2. Ciascuna scelta iniziale porta a $2^{p(n)}$ computazioni
3. N' accetta a maggioranza sse almeno $2^{p(n)} + 1$ run accettano x, sse almeno 1 run di N accetta

QED

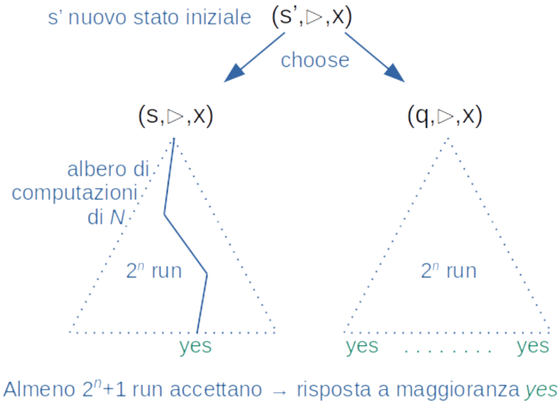
Dimostrazione che **NP** $\subseteq$ **PP** (Teorema 11.3)

- **(cenni)** Prendete un qualunque $L \in$ **NP**, quindi deciso da qualche MdT N (standardizzata) in tempo polinomiale $p(n)$,
- per dimostrare $L \in$ **PP**, trasformate N in una N' che
  - accetta a maggioranza.
  - risponde come N

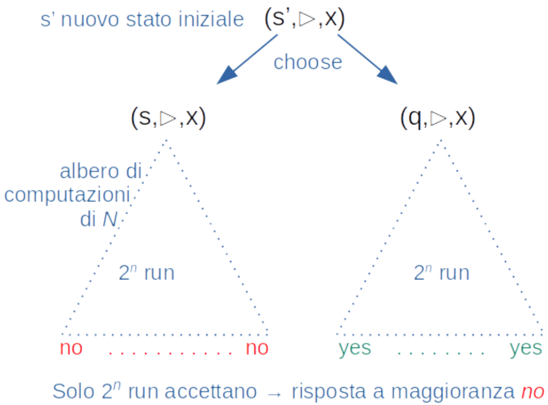
1. Introducete un nuovo stato iniziale con 2 scelte nondeterministiche:
   - eseguire N
   - continuare con scelte nondeterministiche binarie per $p(n)$ passi, terminando sempre in "yes"
2. Ciascuna scelta iniziale porta a $2^{p(n)}$ computazioni
3. N' accetta a maggioranza sse almeno $2^{p(n)} + 1$ run accettano x, sse almeno 1 run di N accetta

QED

Dimostrazione che **NP** ⊆ **PP** (Teorema 11.3)



Dimostrazione che **NP** $\subseteq$ **PP** (Teorema 11.3)



PP vs PSPACE

Teorema

PP $\subseteq$ **PSPACE**

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli “yes” e i “no”. Uso del nastro:

$\triangleright x$ (input)	# “yes” ($O(n^c)$)	# “no” ($O(n^c)$)	dati per scegliere prossimo run ($O(n^c)$)	spazio per i run ($O(n^c)$)

- I numeri di “yes” e “no” sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \mathrm{SPACE}(n^c) \subset$ **PSPACE** (QED)

PP vs PSPACE

Teorema

PP $\subseteq$ **PSPACE**

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli "yes" e i "no". Uso del nastro:

$\rhd x$ (input)	# "yes" ($O(n^c)$)	# "no" ($O(n^c)$)	dati per scegliere prossimo run ($O(n^c)$)	spazio per i run ($O(n^c)$)

- I numeri di "yes" e "no" sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \mathrm{SPACE}(n^c) \subseteq$ **PSPACE** (QED)

PP vs PSPACE

Teorema

PP $\subseteq$ **PSPACE**

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli "yes" e i "no". Uso del nastro:

$\rhd x$ (input)	# "yes" $(O(n^c))$	# "no" $(O(n^c))$	dati per scegliere prossimo run $(O(n^c))$	spazio per i run $(O(n^c))$

- I numeri di "yes" e "no" sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \text{SPACE}(n^c) \subset$ **PSPACE** (QED)

PP vs PSPACE

Teorema

PP $\subseteq$ **PSPACE**

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli "yes" e i "no". Uso del nastro:

$\rhd x$ (input)	# "yes" ($O(n^c)$)	# "no" ($O(n^c)$)	dati per scegliere prossimo run ($O(n^c)$)	spazio per i run ($O(n^c)$)

- I numeri di "yes" e "no" sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \mathrm{SPACE}(n^c) \subset$ **PSPACE** (QED)

PP vs PSPACE

Teorema

PP $\subseteq$ PSPACE

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli "yes" e i "no". Uso del nastro:

$\triangleright x$ (input)	# "yes" ($O(n^c)$)	# "no" ($O(n^c)$)	dati per scegliere prossimo run ($O(n^c)$)	spazio per i run ($O(n^c)$)

- I numeri di "yes" e "no" sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \text{SPACE}(n^c) \subset$ **PSPACE** (QED)

PP vs PSPACE

Teorema

PP $\subseteq$ PSPACE

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli "yes" e i "no". Uso del nastro:

$\rhd x$ (input)	# "yes" ($O(n^c)$)	# "no" ($O(n^c)$)	dati per scegliere prossimo run ($O(n^c)$)	spazio per i run ($O(n^c)$)

- I numeri di "yes" e "no" sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \mathrm{SPACE}(n^c) \subset$ **PSPACE** (QED)

PP vs PSPACE

Teorema

PP $\subseteq$ PSPACE

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli "yes" e i "no". Uso del nastro:

$\triangleright x$ (input)	# "yes" ($O(n^c)$)	# "no" ($O(n^c)$)	dati per scegliere prossimo run ($O(n^c)$)	spazio per i run ($O(n^c)$)

- I numeri di "yes" e "no" sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \text{SPACE}(n^c) \subset$ **PSPACE** (QED)

PP vs PSPACE

Teorema

PP $\subseteq$ **PSPACE**

Cenni alla dimostrazione:

- Dato un qualsiasi $L \in$ **PP** esiste una MdT N che lo accetta a maggioranza in tempo n^c, per qualche $c \in \mathbb{N}$. Trasformiamo N in M deterministica che opera in spazio $O(n^c)$
- M esegue deterministicamente *tutti* i 2^{n^c} run di N, uno alla volta (invece di scegliere a caso) e conta gli "yes" e i "no". Uso del nastro:

$\rhd x$ (input)	# "yes" ($O(n^c)$)	# "no" ($O(n^c)$)	dati per scegliere prossimo run ($O(n^c)$)	spazio per i run ($O(n^c)$)

- I numeri di "yes" e "no" sono $\leq 2^{n^c}$, rappresentabili in binario con n^c celle
- Per scegliere il prossimo run si usano sequenze di $\leq n^c$ numeri
- Ogni run richiede al massimo n^c passi (per ipotesi) quindi usa al massimo n^c celle del nastro. Riutilizziamo lo stesso spazio per tutti i run
- Quindi $L \in \mathrm{SPACE}(n^c) \subset$ **PSPACE** (QED)

La classe **BPP**
(Bounded error Probabilistic Polynomial time)

Ridurre la probabilità di errore *velocemente*

Torniamo alla domanda: *Quante volte devo ripetere un algoritmo probabilistico prima che la probabilità di errore*[1] *sia minore di una soglia data?*

- **Problema di PP:** Le ripetizioni possono essere troppe (esponenziali) con la semplice accettazione a maggioranza
  - al crescere del numero di run, le probabilità di risposta corretta o errata potrebbero essere quasi uguali ($2^{n^k} + 1$ contro $2^{n^k} - 1$)
- La soluzione è ovvia:
  - Richiedere che le risposte corrette abbiano una "chiara" maggioranza

  Questo porta direttamente a **BPP**

[1] errore = la maggioranza delle risposte è errata

Ridurre la probabilità di errore *velocemente*

Torniamo alla domanda: *Quante volte devo ripetere un algoritmo probabilistico prima che la probabilità di errore*[1] *sia minore di una soglia data?*

- **Problema di PP:** Le ripetizioni possono essere troppe (esponenziali) con la semplice accettazione a maggioranza
  - al crescere del numero di run, le probabilità di risposta corretta o errata potrebbero essere quasi uguali ($2^{n^k} + 1$ contro $2^{n^k} - 1$)
- La soluzione è ovvia:
  - Richiedere che le risposte corrette abbiano una "chiara" maggioranza

  Questo porta direttamente a **BPP**

[1] errore = la maggioranza delle risposte è errata

Ridurre la probabilità di errore *velocemente*

Torniamo alla domanda: *Quante volte devo ripetere un algoritmo probabilistico prima che la probabilità di errore*[1] *sia minore di una soglia data?*

- **Problema di PP:** Le ripetizioni possono essere troppe (esponenziali) con la semplice accettazione a maggioranza
  - al crescere del numero di run, le probabilità di risposta corretta o errata potrebbero essere quasi uguali ($2^{n^k}+1$ contro $2^{n^k}-1$)
- La soluzione è ovvia:
  - Richiedere che le risposte corrette abbiano una "chiara" maggioranza

  Questo porta direttamente a **BPP**

[1] errore = la maggioranza delle risposte è errata

La classe **BPP**

Definizione

$L \in$ **BPP** sse esiste una MdT N nondeterministica (standardizzata) che opera in tempo polinomiale tale che

- se $x \in L$ allora almeno $\frac{3}{4}$ dei run di N accettano x
- se $x \notin L$ allora almeno $\frac{3}{4}$ dei run di N rigettano x

- **Nota**: la scelta di $\frac{3}{4}$ è convenzionale: qualunque altra soglia s t.c. $\frac{1}{2} < s \leq 1$ produrrebbe la stessa classe
- L'importante è che sia espressa come *percentuale* dei run
  - la differenza tra risposte corrette ed errate aumenta con l'aumentare dei run

La classe **BPP**

Definizione

$L \in$ **BPP** sse esiste una MdT N nondeterministica (standardizzata) che opera in tempo polinomiale tale che

- se $x \in L$ allora almeno $\frac{3}{4}$ dei run di N accettano x
- se $x \notin L$ allora almeno $\frac{3}{4}$ dei run di N rigettano x

- **Nota**: la scelta di $\frac{3}{4}$ è convenzionale: qualunque altra soglia s t.c. $\frac{1}{2} < s \leq 1$ produrrebbe la stessa classe
- L'importante è che sia espressa come *percentuale* dei run
  - la differenza tra risposte corrette ed errate aumenta con l'aumentare dei run

Natura convenzionale della soglia

- Supponiamo che N decida L con una maggioranza $\frac{1}{2} + \epsilon \neq \frac{3}{4}$
- Si può dimostrare che dopo $2k + 1$ ripetizioni la probabilità che la maggioranza dei risultati dia un risultato errato è

$$e^{-2\epsilon^2 k}$$

- Quindi per ridurla a meno di $1/4$ (come richiesto da **BPP**) basta iterare N per $2k + 1$ volte dove

$$k = \left\lceil \frac{\ln 2}{\epsilon^2} \right\rceil$$

  Notare che k è costante (dipende solo da N)
- Di conseguenza la nuova macchina con soglia a $\frac{3}{4}$ rallenta solo di un fattore costante $2k + 1$

Natura convenzionale della soglia

- Supponiamo che N decida L con una maggioranza $\frac{1}{2} + \epsilon \neq \frac{3}{4}$
- Si può dimostrare che dopo $2k+1$ ripetizioni la probabilità che la maggioranza dei risultati dia un risultato errato è

$$e^{-2\epsilon^2 k}$$

- Quindi per ridurla a meno di 1/4 (come richiesto da **BPP**) basta iterare N per $2k+1$ volte dove

$$k = \left\lceil \frac{\ln 2}{\epsilon^2} \right\rceil$$

Notare che k è costante (dipende solo da N)

- Di conseguenza la nuova macchina con soglia a $\frac{3}{4}$ rallenta solo di un fattore costante $2k+1$

Natura convenzionale della soglia

- Supponiamo che N decida L con una maggioranza $\frac{1}{2} + \epsilon \neq \frac{3}{4}$
- Si può dimostrare che dopo $2k+1$ ripetizioni la probabilità che la maggioranza dei risultati dia un risultato errato è

$$e^{-2\epsilon^2 k}$$

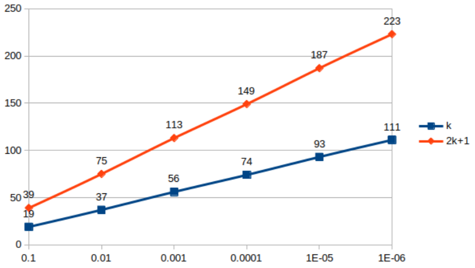
- Quindi per ridurla a meno di $1/4$ (come richiesto da **BPP**) basta iterare N per $2k+1$ volte dove

$$k = \left\lceil \frac{\ln 2}{\epsilon^2} \right\rceil$$

  Notare che k è costante (dipende solo da N)
- Di conseguenza la nuova macchina con soglia a $\frac{3}{4}$ rallenta solo di un fattore costante $2k+1$

Costo della riduzione di errore (# iterazioni)

- Con calcoli analoghi si mostra che la soglia desiderata di errore in **BPP** si raggiunge con un numero polinomiale di ripetizioni
- Se la probabilità di risposta corretta è $3/4$, dopo $2k+1$ ripetizioni la probabilità di errore è $e^{-2(1/4)^2 k} = e^{-k/8}$
- Per ottenere probabilità di errore ε porre $k = \lceil -8 \ln \varepsilon \rceil$



Relazioni tra **BPP** e le altre classi

Proposizione

P $\subseteq$ **BPP**

- Banale: Le MdT deterministiche sono casi particolari di MdT nondeterministiche che generano sempre solo un run
- la risposta del run unico rappresenta il 100% delle risposte

Proposizione

BPP $\subseteq$ **PP**

- Banale: l'accettazione a maggioranza qualificata di **BPP** implica quella a maggioranza semplice

Relazioni tra **BPP** e le altre classi

Proposizione

P $\subseteq$ **BPP**

- Banale: Le MdT deterministiche sono casi particolari di MdT nondeterministiche che generano sempre solo un run
- la risposta del run unico rappresenta il 100% delle risposte

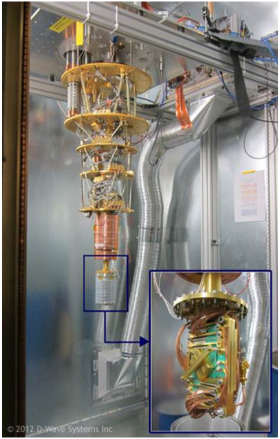
Proposizione

BPP $\subseteq$ **PP**

- Banale: l'accettazione a maggioranza qualificata di **BPP** implica quella a maggioranza semplice

Calcolo quantistico

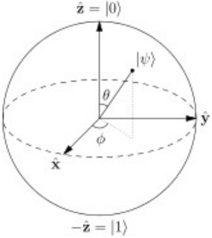
- I computer quantistici sono sostanzialmente un hardware particolare per implementare algoritmi probabilistici



I qubit e il calcolo probabilistico

- I qubit hanno una rappresentazione vettoriale
  - due vettori, denotati $|0\rangle$ e $|1\rangle$ corrispondono a 0 e 1 tradizionali
  - un qubit può anche essere una combinazione lineare di $|0\rangle$ e $|1\rangle$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$



α e β possono essere numeri complessi

$$\begin{aligned}\alpha &= \cos\left(\frac{\theta}{2}\right)\\ \beta &= e^{i\phi}\sin\left(\frac{\theta}{2}\right)\end{aligned}$$

- α e β sono detti *probability amplitudes*
- Quando il valore di $|\psi\rangle$ viene misurato, lo stato del qubit collassa a $|0\rangle$ con probabilità $|\alpha|^2$ e a $|1\rangle$ con probabilità $|\beta|^2$
  - così si implementano le scelte probabilistiche

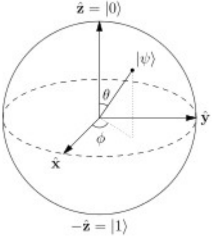
I qubit e il calcolo probabilistico

- I qubit hanno una rappresentazione vettoriale
  - due vettori, denotati $|0\rangle$ e $|1\rangle$ corrispondono a 0 e 1 tradizionali
  - un qubit può anche essere una combinazione lineare di $|0\rangle$ e $|1\rangle$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

α e β possono essere numeri complessi

$$\begin{aligned} \alpha &= \cos\left(\frac{\theta}{2}\right) \\ \beta &= e^{i\phi}\sin\left(\frac{\theta}{2}\right) \end{aligned}$$



- α e β sono detti *probability amplitudes*
- Quando il valore di $|\psi\rangle$ viene misurato, lo stato del qubit collassa a $|0\rangle$ con probabilità $|\alpha|^2$ e a $|1\rangle$ con probabilità $|\beta|^2$
  - così si implementano le scelte probabilistiche

Circuiti quantistici come algoritmi probabilistici

- I qubit possono essere elaborati mediante *gate quantistici* che generalizzano i classici *and*, *or*, *nand*...
  - matematicamente: sono delle matrici che trasformano i valori dei qubit in ingresso
- l'output sarà un array di qubit la cui misura dovrà restituire la risposta esatta con probabilità "nettamente" maggiore di 1/2
  - così ripetendo il calcolo k volte ridurremo la probabilità di errore alla soglia desiderata
- Tuttavia i circuiti quantistici hanno una struttura molto diversa dagli algoritmi classici (MdT) e non sono adatti a un confronto diretto

Circuiti quantistici come algoritmi probabilistici

- I qubit possono essere elaborati mediante *gate quantistici* che generalizzano i classici *and*, *or*, *nand*...
  - matematicamente: sono delle matrici che trasformano i valori dei qubit in ingresso
- l'output sarà un array di qubit la cui misura dovrà restituire la risposta esatta con probabilità "nettamente" maggiore di $1/2$
  - così ripetendo il calcolo k volte ridurremo la probabilità di errore alla soglia desiderata
- Tuttavia i circuiti quantistici hanno una struttura molto diversa dagli algoritmi classici (MdT) e non sono adatti a un confronto diretto

Circuiti quantistici come algoritmi probabilistici

- I qubit possono essere elaborati mediante *gate quantistici* che generalizzano i classici *and*, *or*, *nand*...
  - matematicamente: sono delle matrici che trasformano i valori dei qubit in ingresso
- l'output sarà un array di qubit la cui misura dovrà restituire la risposta esatta con probabilità "nettamente" maggiore di $1/2$
  - così ripetendo il calcolo k volte ridurremo la probabilità di errore alla soglia desiderata
- Tuttavia i circuiti quantistici hanno una struttura molto diversa dagli algoritmi classici (MdT) e non sono adatti a un confronto diretto

Quantum Turing Machines

- Sono più convenienti dei circuiti per le analisi di complessità
  - e sarebbero più facili da programmare dei circuiti quantistici...
- È stato provato che hanno la stessa potenza delle *famiglie uniformi* di circuiti quantistici
  - Esiste un algoritmo polinomiale che dato l'input costruisce il circuito che lo deve analizzare
  - Così si adatta la dimensione del circuito a quella dell'input
- Poichè stati e nastro sono memorizzati con qubit, la QTM si troverà in ogni istante in una *sovrapposizione* di configurazioni
- L'output (probabilistico) si ottiene misurando la configurazione finale

Quantum Turing Machines

- Sono più convenienti dei circuiti per le analisi di complessità
  - e sarebbero più facili da programmare dei circuiti quantistici...
- È stato provato che hanno la stessa potenza delle *famiglie uniformi* di circuiti quantistici
  - Esiste un algoritmo polinomiale che dato l'input costruisce il circuito che lo deve analizzare
  - Così si adatta la dimensione del circuito a quella dell'input
- Poichè stati e nastro sono memorizzati con qubit, la QTM si troverà in ogni istante in una *sovrapposizione* di configurazioni
- L'output (probabilistico) si ottiene misurando la configurazione finale

Quantum Turing Machines

- Sono più convenienti dei circuiti per le analisi di complessità
  - e sarebbero più facili da programmare dei circuiti quantistici...
- È stato provato che hanno la stessa potenza delle *famiglie uniformi* di circuiti quantistici
  - Esiste un algoritmo polinomiale che dato l'input costruisce il circuito che lo deve analizzare
  - Così si adatta la dimensione del circuito a quella dell'input
- Poichè stati e nastro sono memorizzati con qubit, la QTM si troverà in ogni istante in una *sovrapposizione* di configurazioni
- L'output (probabilistico) si ottiene misurando la configurazione finale

Quantum Turing Machines

Definizioni formali

Definizione (QTM)

È una quadrupla $\langle K, \Sigma, \delta, s\rangle$ dove K, Σ, s sono come nelle MdT ma

$$\delta : K \times \Sigma \rightarrow \bar{\mathbb{C}}^{K\times\Sigma\times\{\leftarrow,\rightarrow,-\}}$$

- $\bar{\mathbb{C}}$ è un insieme di numeri complessi *calcolabile velocemente*
  - $\alpha \in \bar{\mathbb{C}}$ sse esiste una MdT che dato k calcola il k-esimo bit delle parti reale e immaginaria di α in tempo polinomiale in k
- $\bar{\mathbb{C}}^{K\times\Sigma\times\{\leftarrow,\rightarrow,-\}}$ è l'insieme delle funzioni

$$f : K \times \Sigma \times \{\leftarrow,\rightarrow,-\} \rightarrow \bar{\mathbb{C}}$$

  $\delta(q,\sigma)$ associa una *probability amplitude* ad ogni azione applicabile in (q,σ)

Quantum Turing Machines

Definizioni formali

Definizione (QTM)

È una quadrupla $\langle K, \Sigma, \delta, s\rangle$ dove K, Σ, s sono come nelle MdT ma

$$\delta : K \times \Sigma \to \tilde{\mathbb{C}}^{K\times\Sigma\times\{\leftarrow,\rightarrow,-\}}$$

- $\tilde{\mathbb{C}}$ è un insieme di numeri complessi *calcolabile velocemente*
  - $\alpha \in \tilde{\mathbb{C}}$ sse esiste una MdT che dato k calcola il k-esimo bit delle parti reale e immaginaria di α in tempo polinomiale in k
- $\tilde{\mathbb{C}}^{K\times\Sigma\times\{\leftarrow,\rightarrow,-\}}$ è l'insieme delle funzioni

$$f : K \times \Sigma \times \{\leftarrow,\rightarrow,-\} \to \tilde{\mathbb{C}}$$

$\delta(q,\sigma)$ associa una *probability amplitude* ad ogni azione applicabile in (q,σ)

Quantum Turing Machines

Configurazioni e transizioni

- Configurazioni di una QTM Q: come per le MdT: $c = (q, w, u)$
- In ogni istante Q si trova in una *sovrapposizione di configurazioni*

$$\sum_c \alpha_c \cdot |c\rangle \qquad (1)$$

- Se inizialmente Q è nella configurazione $c_0 = (s, w\sigma, u)$, al passo successivo sarà nella sovrapposizione (1) tale che:
  - c spazia sulle configurazioni ottenute applicando a c_0 una qualunque delle possibili azioni $a = (q', \sigma', \{\leftarrow, \rightarrow, -\})$
  - $\alpha_c = \delta(s, \sigma)(a)$ (la probability amplitude dell'azione)
- Iterando, si ottengono le sovrapposizioni ai passi successivi

Quantum Turing Machines

Configurazioni e transizioni

- Configurazioni di una QTM Q: come per le MdT: $c = (q, w, u)$
- In ogni istante Q si trova in una *sovrapposizione di configurazioni*

$$\sum_c \alpha_c \cdot |c\rangle \tag{1}$$

- Se inizialmente Q è nella configurazione $c_0 = (s, w\sigma, u)$, al passo successivo sarà nella sovrapposizione (1) tale che:
  - c spazia sulle configurazioni ottenute applicando a c_0 una qualunque delle possibili azioni $a = (q', \sigma', \{\leftarrow, \rightarrow, -\})$
  - $\alpha_c = \delta(s, \sigma)(a)$ (la probability amplitude dell'azione)
- Iterando, si ottengono le sovrapposizioni ai passi successivi

Quantum Turing Machines

Configurazioni e transizioni

- Configurazioni di una QTM Q: come per le MdT: $c = (q, w, u)$
- In ogni istante Q si trova in una *sovrapposizione di configurazioni*

$$\sum_{c} \alpha_c \cdot |c\rangle \tag{1}$$

- Se inizialmente Q è nella configurazione $c_0 = (s, w\sigma, u)$, al passo successivo sarà nella sovrapposizione (1) tale che:
  - c spazia sulle configurazioni ottenute applicando a c_0 una qualunque delle possibili azioni $a = (q', \sigma', \{\leftarrow, \rightarrow, -\})$
  - $\alpha_c = \delta(s, \sigma)(a)$ (la probability amplitude dell'azione)
- Iterando, si ottengono le sovrapposizioni ai passi successivi

Quantum Turing Machines

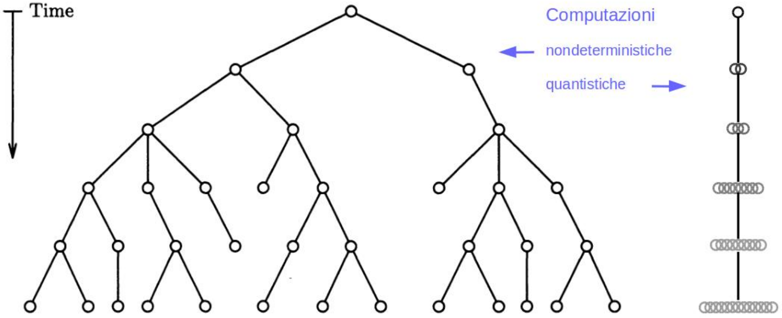
Configurazioni e transizioni

- Configurazioni di una QTM Q: come per le MdT: $c = (q, w, u)$
- In ogni istante Q si trova in una *sovrapposizione di configurazioni*

$$\sum_{c} \alpha_c \cdot |c\rangle \tag{1}$$

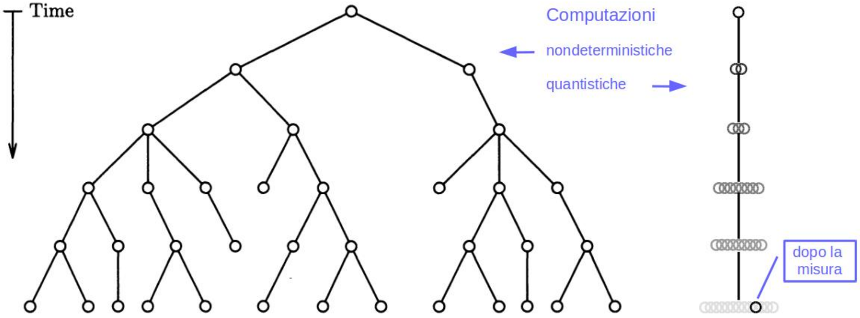
- Se inizialmente Q è nella configurazione $c_0 = (s, w\sigma, u)$, al passo successivo sarà nella sovrapposizione (1) tale che:
  - c spazia sulle configurazioni ottenute applicando a c_0 una qualunque delle possibili azioni $a = (q', \sigma', \{\leftarrow, \rightarrow, -\})$
  - $\alpha_c = \delta(s, \sigma)(a)$ (la probability amplitude dell'azione)
- Iterando, si ottengono le sovrapposizioni ai passi successivi

Quantum Turing Machines



- La misura della sovrapposizione finale fa collassare la configurazione quantistica su una delle configurazioni tradizionali finali
- rispettando le probabilità associate a ogni configurazione

Quantum Turing Machines



- La misura della sovrapposizione finale fa collassare la configurazione quantistica su una delle configurazioni tradizionali finali
- rispettando le probabilità associate a ogni configurazione

Quantum Turing Machines

Misura della configurazione

- Formalmente, se la QTM si trova nella sovrapposizione

$$\sum_i \alpha_i \cdot |c_i\rangle$$

- allora la probabilità di osservare $|c_i\rangle$ è $|\alpha_i|^2$
- se succede, allora la nuova sovrapposizione diventa $|c_i\rangle$
- quindi la misurazione *deve* essere fatta *solo alla fine*
- È quindi importante che la QTM sia *precisa*, cioè che tutte le sue computazioni abbiano esattamente la stessa lunghezza

Quantum Turing Machines

Misura della configurazione

- Formalmente, se la QTM si trova nella sovrapposizione

$$\sum_i \alpha_i \cdot |c_i\rangle$$

- allora la probabilità di osservare $|c_i\rangle$ è $|\alpha_i|^2$
- se succede, allora la nuova sovrapposizione diventa $|c_i\rangle$
- quindi la misurazione *deve* essere fatta *solo alla fine*
- È quindi importante che la QTM sia *precisa*, cioè che tutte le sue computazioni abbiano esattamente la stessa lunghezza

Quantum Turing Machines

Misura della configurazione

- Formalmente, se la QTM si trova nella sovrapposizione

$$\sum_i \alpha_i \cdot |c_i\rangle$$

- allora la probabilità di osservare $|c_i\rangle$ è $|\alpha_i|^2$
- se succede, allora la nuova sovrapposizione diventa $|c_i\rangle$
- quindi la misurazione *deve* essere fatta *solo alla fine*
- È quindi importante che la QTM sia *precisa*, cioè che tutte le sue computazioni abbiano esattamente la stessa lunghezza

Quantum Turing Machines

Misura della configurazione

- Formalmente, se la QTM si trova nella sovrapposizione

$$\sum_i \alpha_i \cdot |c_i\rangle$$

- allora la probabilità di osservare $|c_i\rangle$ è $|\alpha_i|^2$
- se succede, allora la nuova sovrapposizione diventa $|c_i\rangle$
- quindi la misurazione *deve* essere fatta *solo alla fine*
- È quindi importante che la QTM sia *precisa*, cioè che tutte le sue computazioni abbiano esattamente la stessa lunghezza

La classe **BQP** (Bounded-error Quantum Polynomial-time)

Definizione

$L \in$ **BQP** se e solo se esiste una QTM Q tale che per ogni input x la probabilità di osservare una configurazione con "yes" nella misura finale è

- $\geq \frac{2}{3}$ se $x \in L$
- $\leq \frac{1}{3}$ se $x \notin L$ (x viene rigettato con probabilità $\geq \frac{2}{3}$)

- Notare l'analogia con **BPP**
- Cambia il modo in cui vengono attribuite le probabilità alle configurazioni finali
  - in **BPP** sono equiprobabili, in **BQP** dipendono dalla δ di Q

La classe **BQP** (Bounded-error Quantum Polynomial-time)

Definizione

$L \in$ **BQP** se e solo se esiste una QTM Q tale che per ogni input x la probabilità di osservare una configurazione con "yes" nella misura finale è

- $\geq \frac{2}{3}$ se $x \in L$
- $\leq \frac{1}{3}$ se $x \notin L$ (x viene rigettato con probabilità $\geq \frac{2}{3}$)

- Notare l'analogia con **BPP**
- Cambia il modo in cui vengono attribuite le probabilità alle configurazioni finali
  - in **BPP** sono equiprobabili, in **BQP** dipendono dalla δ di Q

La classe **BQP**

scelta della soglia

- La soglia di $\frac{2}{3}$ è convenzionale, come per **BPP**
- Dato un limite desiderato all'errore, si può ottenere una risposta che lo soddisfa
- iterando Q un numero polinomiale di volte

La classe **BQP**

scelta della soglia

- La soglia di $\frac{2}{3}$ è convenzionale, come per **BPP**
- Dato un limite desiderato all'errore, si può ottenere una risposta che lo soddisfa
- iterando Q un numero polinomiale di volte

Relazioni tra **BQP** e le altre classi di complessità

- Servono a capire quali problemi difficili per il calcolo classico diventeranno facili con quello quantistico
- e quali resteranno difficili
- *anche se tutti i problemi tecnologici venissero risolti* (decoherencing, errori, limiti di memoria, ...)
  - le QTM non hanno questi problemi, sono una idealizzazione
- Quindi nessuna delle analisi di complessità che seguiranno dipende dagli attuali limiti tecnologici
  - derivano unicamente dalle proprietà intrinseche delle computazioni

Relazioni tra **BQP** e le altre classi di complessità

- Servono a capire quali problemi difficili per il calcolo classico diventeranno facili con quello quantistico
- e quali resteranno difficili
- *anche se tutti i problemi tecnologici venissero risolti* (decoherencing, errori, limiti di memoria, ...)
  - le QTM non hanno questi problemi, sono una idealizzazione
- Quindi nessuna delle analisi di complessità che seguiranno dipende dagli attuali limiti tecnologici
  - derivano unicamente dalle proprietà intrinseche delle computazioni

Relazioni tra **BQP** e le altre classi di complessità

- Servono a capire quali problemi difficili per il calcolo classico diventeranno facili con quello quantistico
- e quali resteranno difficili
- *anche se tutti i problemi tecnologici venissero risolti* (decoherencing, errori, limiti di memoria, ...)
  - le QTM non hanno questi problemi, sono una idealizzazione
- Quindi nessuna delle analisi di complessità che seguiranno dipende dagli attuali limiti tecnologici
  - derivano unicamente dalle proprietà intrinseche delle computazioni

Relazioni tra **BQP** e le altre classi di complessità

Teorema

BPP $\subseteq$ **BQP**

- Non sorprende, vista la maggiore generalità delle distribuzione di probabilità sulle configurazioni finali delle QTM
- Si *pensa* che **BPP** $\neq$ **BQP** (ennesima questione aperta)
- Corollario: **P** $\subseteq$ **BQP** (perchè **P** $\subseteq$ **BPP**)

Relazioni tra **BQP** e le altre classi di complessità

Teorema

BPP $\subseteq$ **BQP**

- Non sorprende, vista la maggiore generalità delle distribuzione di probabilità sulle configurazioni finali delle QTM
- Si *pensa* che **BPP** $\neq$ **BQP** (ennesima questione aperta)
- Corollario: **P** $\subseteq$ **BQP** (perchè **P** $\subseteq$ **BPP**)

Relazioni tra **BQP** e le altre classi di complessità

Teorema

$\mathbf{BPP} \subseteq \mathbf{BQP}$

- Non sorprende, vista la maggiore generalità delle distribuzione di probabilità sulle configurazioni finali delle QTM
- Si *pensa* che $\mathbf{BPP} \neq \mathbf{BQP}$ (ennesima questione aperta)
- Corollario: $\mathbf{P} \subseteq \mathbf{BQP}$ (perchè $\mathbf{P} \subseteq \mathbf{BPP}$)

Relazioni tra **BQP** e le altre classi di complessità

Teorema

BQP $\subseteq$ **PP**

- Di nuovo, non sorprendente visto che la condizione di accettazione di **PP** è più debole (non richiede maggioranze di 2/3)
- Corollari/conseguenze:
  1. **BQP** $\subseteq$ **PSPACE** (perchè **PP** $\subseteq$ **PSPACE**)
  2. Si congettura che **PP** $\subsetneq$ **PSPACE** quindi anche **BQP** $\subsetneq$ **PSPACE**

Relazioni tra **BQP** e le altre classi di complessità

Teorema

BQP $\subseteq$ **PP**

- Di nuovo, non sorprendente visto che la condizione di accettazione di **PP** è più debole (non richiede maggioranze di 2/3)
- Corollari/conseguenze:
  1. **BQP** $\subseteq$ **PSPACE** (perchè **PP** $\subseteq$ **PSPACE**)
  2. Si congettura che **PP** $\subsetneq$ **PSPACE** quindi anche **BQP** $\subsetneq$ **PSPACE**

Relazioni tra **BQP** e le altre classi di complessità

- Si pensa che **NP** $\not\subseteq$ **BQP** (!) e che **BQP** $\not\subseteq$ **NP**
- Evidenza: alcune computazioni nondeterministiche non sono simulabili quantisticamente e viceversa
  - dimostrato con opportuni oracoli
- È una evidenza forte ma non definitiva
  - gli algoritmi non simulabili potrebbero non essere essenziali
  - (potrebbero risolvere problemi risolubili anche in altro modo)
- Se **NP** $\not\subseteq$ **BQP** allora anche **BQP** $\subsetneq$ **PSPACE**
  - perché **NP** $\subseteq$ **PSPACE**

Relazioni tra **BQP** e le altre classi di complessità

- Si pensa che **NP** $\not\subseteq$ **BQP** (!) e che **BQP** $\not\subseteq$ **NP**
- Evidenza: alcune computazioni nondeterministiche non sono simulabili quantisticamente e viceversa
  - dimostrato con opportuni oracoli
- È una evidenza forte ma non definitiva
  - gli algoritmi non simulabili potrebbero non essere essenziali
  - (potrebbero risolvere problemi risolubili anche in altro modo)
- Se **NP** $\not\subseteq$ **BQP** allora anche **BQP** $\subsetneq$ **PSPACE**
  - perché **NP** $\subseteq$ **PSPACE**

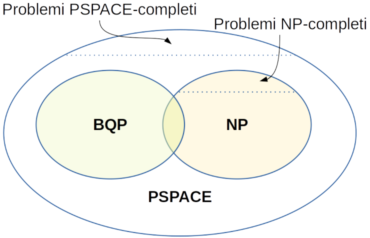
Relazioni tra **BQP** e le altre classi di complessità

- Si pensa che **NP** $\not\subseteq$ **BQP** (!) e che **BQP** $\not\subseteq$ **NP**
- Evidenza: alcune computazioni nondeterministiche non sono simulabili quantisticamente e viceversa
  - dimostrato con opportuni oracoli
- È una evidenza forte ma non definitiva
  - gli algoritmi non simulabili potrebbero non essere essenziali
  - (potrebbero risolvere problemi risolubili anche in altro modo)
- Se **NP** $\not\subseteq$ **BQP** allora anche **BQP** $\subsetneq$ **PSPACE**
  - perché **NP** $\subseteq$ **PSPACE**

Relazioni tra **BQP** e le altre classi di complessità

- Si pensa che **NP** $\not\subseteq$ **BQP** (!) e che **BQP** $\not\subseteq$ **NP**
- Evidenza: alcune computazioni nondeterministiche non sono simulabili quantisticamente e viceversa
  - dimostrato con opportuni oracoli
- È una evidenza forte ma non definitiva
  - gli algoritmi non simulabili potrebbero non essere essenziali
  - (potrebbero risolvere problemi risolubili anche in altro modo)
- Se **NP** $\not\subseteq$ **BQP** allora anche **BQP** $\subsetneq$ **PSPACE**
  - perché **NP** $\subseteq$ **PSPACE**

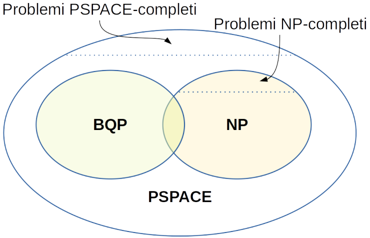
Relazioni tra **BQP** e le altre classi di complessità



Conseguenze immediate della congettura **NP** $\not\subseteq$ **BQP**

- Nessuno dei problemi **NP**-completi può essere risolto in tempo polinomiale con un calcolatore quantistico
  - perchè basterebbe risolverne 1 per risolvere tutti i problemi in **NP** e avere **NP** $\subseteq$ **BQP**
  - esempi: KNAPSACK, traveling salesman, e molti altri problemi di ottimizzazione (anche applicati a machine learning/clustering)
  - ragionamento in logica proposizionale (AI), risoluzione di sistemi di vincoli

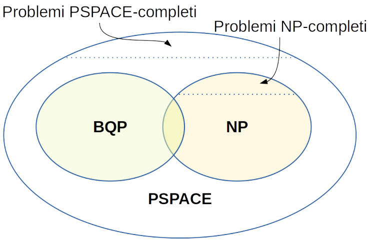
Relazioni tra **BQP** e le altre classi di complessità



Conseguenze immediate della congettura **NP** $\not\subseteq$ **BQP**

- Nessuno dei problemi **NP**-completi può essere risolto in tempo polinomiale con un calcolatore quantistico
  - perchè basterebbe risolverne 1 per risolvere tutti i problemi in **NP** e avere **NP** $\subseteq$ **BQP**
  - esempi: KNAPSACK, traveling salesman, e molti altri problemi di ottimizzazione (anche applicati a machine learning/clustering)
  - ragionamento in logica proposizionale (AI), risoluzione di sistemi di vincoli

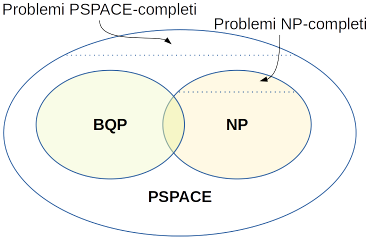
Relazioni tra **BQP** e le altre classi di complessità



Conseguenze immediate della congettura **NP** $\not\subseteq$ **BQP**

- Nessuno dei problemi **NP**-completi può essere risolto in tempo polinomiale con un calcolatore quantistico
  - perchè basterebbe risolverne 1 per risolvere tutti i problemi in **NP** e avere **NP** $\subseteq$ **BQP**
  - esempi: KNAPSACK, traveling salesman, e molti altri problemi di ottimizzazione (anche applicati a machine learning/clustering)
  - ragionamento in logica proposizionale (AI), risoluzione di sistemi di vincoli

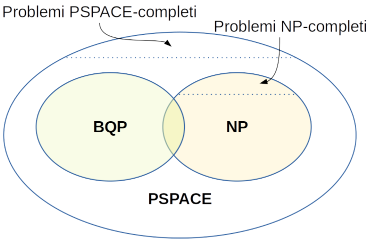
Relazioni tra **BQP** e le altre classi di complessità



Conseguenze immediate della congettura **BQP** $\subsetneq$ **PSPACE**

- Nessuno dei problemi **PSPACE**-completi può essere risolto in tempo polinomiale con un calcolatore quantistico
  - altrimenti **PSPACE** $\subseteq$ **BQP** (vedi sopra)
  - ad es. dire se esiste una strategia vincente per GO, GEOGRAPHY, ...
  - ragionamento in logica proposizionale quantificata (AI)

Relazioni tra **BQP** e le altre classi di complessità



Conseguenze immediate della congettura **BQP** $\subsetneq$ **PSPACE**

- Nessuno dei problemi **PSPACE**-completi può essere risolto in tempo polinomiale con un calcolatore quantistico
  - altrimenti **PSPACE** $\subseteq$ **BQP** (vedi sopra)
  - ad es. dire se esiste una strategia vincente per GO, GEOGRAPHY, ...
  - ragionamento in logica proposizionale quantificata (AI)

Relazioni tra **BQP** e le altre classi di complessità

Inoltre abbiamo risultati negativi “sicuri” (non congetture)

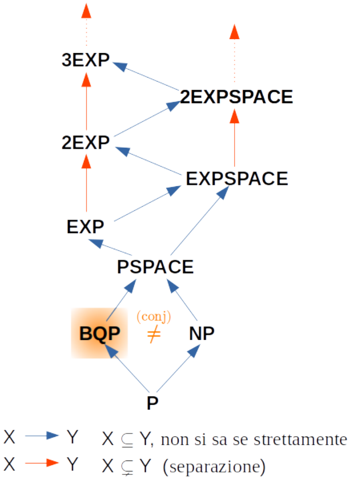
- **BQP** $\subsetneq$ 2**EXP**
  - poichè **BQP** $\subseteq$ **PSPACE** $\subseteq$ **EXP** $\subset$ 2**EXP**
- quindi i problemi 2**EXP**-completi non sono risolubili in tempo polinomiale da un calcolatore quantistico
  - ad es. il ragionamento con ontologie scritte in OWL2 (AI, Semantic Web)

Relazioni tra **BQP** e le altre classi di complessità

Inoltre abbiamo risultati negativi “sicuri” (non congetture)

- **BQP** $\subsetneq$ 2**EXP**
  - poichè **BQP** $\subseteq$ **PSPACE** $\subseteq$ **EXP** $\subset$ 2**EXP**
- quindi i problemi 2**EXP**-completi non sono risolubili in tempo polinomiale da un calcolatore quantistico
  - ad es. il ragionamento con ontologie scritte in OWL2 (AI, Semantic Web)

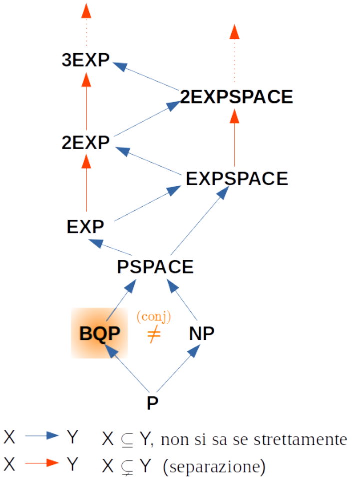
Mappa ricapitolativa delle relazioni tra computazioni tradizionali e quantistiche



BQP si trova piuttosto in basso nella gerarchia

- La maggior parte dei problemi difficili resta tale nel quantum computing
- allora perchè tanto interesse?

Mappa ricapitolativa delle relazioni tra computazioni tradizionali e quantistiche



BQP si trova piuttosto in basso nella gerarchia

- La maggior parte dei problemi difficili resta tale nel quantum computing
- allora perchè tanto interesse?

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

1. Calcolo della Trasformata di Fourier
   - per spazi n-dimensionali passa da $O(n \log n)$ a $O(\log^2 n)$
   - accelerazione esponenziale di un problema che è già in **P**
   - può essere usato per ...
2. Fattorizzazione di interi
   - problema in **NP**, probabilmente *non* NP-completo
   - non noti algoritmi polinomiali, non si sa se in **P**
   - l'*algoritmo di Shor* fattorizza in tempo polinomiale
   - usando l'algoritmo per la trasformata di Fourier

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

1. Calcolo della Trasformata di Fourier
   - per spazi n-dimensionali passa da $O(n \log n)$ a $O(\log^2 n)$
   - accelerazione esponenziale di un problema che è già in **P**
   - può essere usato per ...
2. Fattorizzazione di interi
   - problema in **NP**, probabilmente *non* NP-completo
   - non noti algoritmi polinomiali, non si sa se in **P**
   - l'*algoritmo di Shor* fattorizza in tempo polinomiale
   - usando l'algoritmo per la trasformata di Fourier

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

1. Calcolo della Trasformata di Fourier
   - per spazi n-dimensionali passa da $O(n \log n)$ a $O(\log^2 n)$
   - accelerazione esponenziale di un problema che è già in **P**
   - può essere usato per ...

2. Fattorizzazione di interi
   - problema in **NP**, probabilmente *non* NP-completo
   - non noti algoritmi polinomiali, non si sa se in **P**
   - l'*algoritmo di Shor* fattorizza in tempo polinomiale
   - usando l'algoritmo per la trasformata di Fourier

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

1. Calcolo della Trasformata di Fourier
   - per spazi n-dimensionali passa da $O(n \log n)$ a $O(\log^2 n)$
   - accelerazione esponenziale di un problema che è già in **P**
   - può essere usato per ...
2. Fattorizzazione di interi
   - problema in **NP**, probabilmente *non* NP-completo
   - non noti algoritmi polinomiali, non si sa se in **P**
   - l'*algoritmo di Shor* fattorizza in tempo polinomiale
   - usando l'algoritmo per la trasformata di Fourier

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

1. Calcolo della Trasformata di Fourier
   - per spazi n-dimensionali passa da $O(n \log n)$ a $O(\log^2 n)$
   - accelerazione esponenziale di un problema che è già in **P**
   - può essere usato per ...
2. Fattorizzazione di interi
   - problema in **NP**, probabilmente *non* NP-completo
   - non noti algoritmi polinomiali, non si sa se in **P**
   - l'*algoritmo di Shor* fattorizza in tempo polinomiale
   - usando l'algoritmo per la trasformata di Fourier

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

3. Ricerca in una lista non ordinata S (*algoritmo di Grover*):

$$\text{data } f : S \to \{0,1\} \text{ trovare un } x \in S \text{ tale che } f(x) = 1$$

- notare la relazione con i problemi in **NP**
  - dove $|S|$ è esponenziale, i suoi elementi x hanno dimensione polinomiale e f è calcolabile in tempo polinomiale
- Il calcolo deterministico a forza bruta impiega tempo $O(m \cdot |S|)$, se il costo di $f(x)$ è $O(m)$
- con Grover si riduce a $O(m \cdot \sqrt{|S|})$

I tre algoritmi quantistici citati hanno conseguenze interessanti in pratica

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

3. Ricerca in una lista non ordinata S (*algoritmo di Grover*):

$$\text{data } f : S \to \{0,1\} \text{ trovare un } x \in S \text{ tale che } f(x) = 1$$

- notare la relazione con i problemi in **NP**
  - dove $|S|$ è esponenziale, i suoi elementi x hanno dimensione polinomiale e f è calcolabile in tempo polinomiale
- Il calcolo deterministico a forza bruta impiega tempo $O(m \cdot |S|)$, se il costo di $f(x)$ è $O(m)$
- con Grover si riduce a $O(m \cdot \sqrt{|S|})$

I tre algoritmi quantistici citati hanno conseguenze interessanti in pratica

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

3 Ricerca in una lista non ordinata S (*algoritmo di Grover*):

$$\text{data } f : S \to \{0,1\} \text{ trovare un } x \in S \text{ tale che } f(x) = 1$$

- notare la relazione con i problemi in **NP**
  - dove $|S|$ è esponenziale, i suoi elementi x hanno dimensione polinomiale e f è calcolabile in tempo polinomiale
- Il calcolo deterministico a forza bruta impiega tempo $O(m \cdot |S|)$, se il costo di $f(x)$ è $O(m)$
- con Grover si riduce a $O(m \cdot \sqrt{|S|})$

I tre algoritmi quantistici citati hanno conseguenze interessanti in pratica

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

3. Ricerca in una lista non ordinata S (*algoritmo di Grover*):

$$\text{data } f : S \to \{0,1\} \text{ trovare un } x \in S \text{ tale che } f(x) = 1$$

- notare la relazione con i problemi in **NP**
  - dove $|S|$ è esponenziale, i suoi elementi x hanno dimensione polinomiale e f è calcolabile in tempo polinomiale
- Il calcolo deterministico a forza bruta impiega tempo $O(m \cdot |S|)$, se il costo di $f(x)$ è $O(m)$
- con Grover si riduce a $O(m \cdot \sqrt{|S|})$

I tre algoritmi quantistici citati hanno conseguenze interessanti in pratica

Problemi con speedup interessanti

Nota: **non soluzioni precise ma errore limitato**

3. Ricerca in una lista non ordinata S (*algoritmo di Grover*):

   $$\text{data } f : S \to \{0,1\} \text{ trovare un } x \in S \text{ tale che } f(x) = 1$$

   - notare la relazione con i problemi in **NP**
     - dove $|S|$ è esponenziale, i suoi elementi x hanno dimensione polinomiale e f è calcolabile in tempo polinomiale
   - Il calcolo deterministico a forza bruta impiega tempo $O(m \cdot |S|)$, se il costo di $f(x)$ è $O(m)$
   - con Grover si riduce a $O(m \cdot \sqrt{|S|})$

I tre algoritmi quantistici citati hanno conseguenze interessanti in pratica

Applicazioni dell'algoritmo quantistico di Grover

Applicazione di Grover a problemi NP-completi

Accelerazione esponenziale rispetto ad algoritmi naive: da $O(c^{n^k})$ a $O(c^{n^k/2})$

Proposizione

Ogni problema in **NP** può essere risolto quantisticamente con errore limitato in tempo $O(\sqrt{c^{n^k}}) = O(c^{n^k/2})$ (per qualche $c, k \in \mathbb{N}$)

- **Prova per** KNAPSACK (cenni): Usare l'algoritmo di Grover per trovare un sottoinsieme degli n oggetti che ha peso $\leq W$ e valore $\geq V$

$$f(x) = 1 \text{ se e solo se } \sum_{i \in x} w_i \leq W \text{ e } \sum_{i \in x} v_i \geq V$$

- Iterare k_ε volte per abbassare la probabilità di errore fino alla soglia desiderata.
  - k_ε è costante, dipende solo dalla probabilità di errore ε desiderata
- Costo totale: $k_\varepsilon \cdot$ costo di f $(O(n)) \cdot \sqrt{2^n} = O(4^{n/2})$ QED

Applicazione di Grover a problemi NP-completi

Accelerazione esponenziale rispetto ad algoritmi naive: da $O(c^{n^k})$ a $O(c^{n^k/2})$

Proposizione

Ogni problema in **NP** può essere risolto quantisticamente con errore limitato in tempo $O(\sqrt{c^{n^k}}) = O(c^{n^k/2})$ (per qualche $c, k \in \mathbb{N}$)

- **Prova per** KNAPSACK (cenni): Usare l'algoritmo di Grover per trovare un sottoinsieme degli n oggetti che ha peso $\leq W$ e valore $\geq V$

$$f(x) = 1 \text{ se e solo se } \sum_{i \in x} w_i \leq W \text{ e } \sum_{i \in x} v_i \geq V$$

- Iterare k_ε volte per abbassare la probabilità di errore fino alla soglia desiderata.
  - k_ε è costante, dipende solo dalla probabilità di errore ε desiderata
- Costo totale: $k_\varepsilon \cdot$ costo di f $(O(n)) \cdot \sqrt{2^n} = O(4^{n/2})$ QED

Applicazioni di Grover a problemi NP-completi (II)

- Soluzione più veloce degli algoritmi tradizionali di un fattore $c^{n^k}/2$ (esponenziale)
  - si riescono a risolvere istanze più grandi
- Ma il problema resta esponenziale. Ad esempio KNAPSACK

Algoritmo tradizionale naive $O(c^n)$

- $n < 40$: < 11 μsec
- $n = 60$: > 11 sec
- $n = 70$: $\sim$ 3h 20'
- $n = 80$: $\sim$ 140gg
- $n = 90$: $\sim$ 4 secoli
- $n = 100$: $>$ 401 millenni

Algoritmo di Grover $O(c^{n/2})$

- $n < 80$: < 11 μsec
- $n = 120$: $\sim$ 12 sec
- $n = 140$: $\sim$ 3h 20'
- $n = 150$: $\sim$ 4gg 10h
- $n = 160$: $\sim$ 140gg
- $n = 170$: $>$ 12 anni e 3 mesi

(restiamo comunque lontani dagli ordini di grandezza 10^4 dell'esempio dei container e dei task più grandi di apprendimento del machine learning)

Applicazioni di Grover a problemi NP-completi (II)

- Soluzione più veloce degli algoritmi tradizionali di un fattore $c^{n^k}/2$ (esponenziale)
  - si riescono a risolvere istanze più grandi
- Ma il problema resta esponenziale. Ad esempio KNAPSACK

Algoritmo tradizionale naive $O(c^n)$

- $n < 40$: < 11 μsec
- $n = 60$: > 11 sec
- $n = 70$: $\sim$ 3h 20'
- $n = 80$: $\sim$ 140gg
- $n = 90$: $\sim$ 4 secoli
- $n = 100$: > 401 millenni

Algoritmo di Grover $O(c^{n/2})$

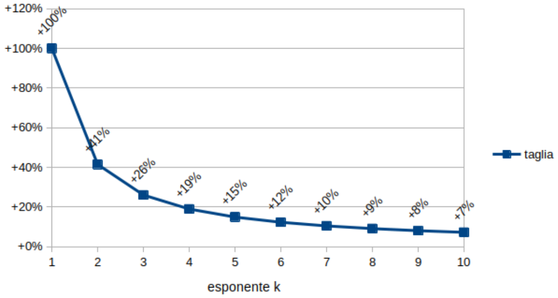
- $n < 80$: < 11 μsec
- $n = 120$: $\sim$ 12 sec
- $n = 140$: $\sim$ 3h 20'
- $n = 150$: $\sim$ 4gg 10h
- $n = 160$: $\sim$ 140gg
- $n = 170$: $>$ 12 anni e 3 mesi

(restiamo comunque lontani dagli ordini di grandezza 10^4 dell'esempio dei container e dei task più grandi di apprendimento del machine learning)

Applicazioni di Grover a problemi NP-completi (III)

In generale, l'incremento della taglia (a parità di tempo) dipende dalla complessità del problema

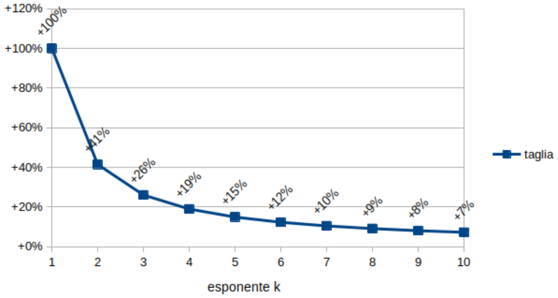
- Se l'algoritmo naive impiega tempo $O(2^{n^k})$, allora l'incremento da aspettarsi è $O(\sqrt[k]{2})$



Applicazioni di Grover a problemi NP-completi (III)

In generale, l'incremento della taglia (a parità di tempo) dipende dalla complessità del problema

- Se l'algoritmo naive impiega tempo $O(2^{n^k})$, allora l'incremento da aspettarsi è $O(\sqrt[k]{2})$



Applicazioni di Grover a problemi in NP

Domande da porsi per ogni problema specifico (rapporto costo-prestazioni):

- È facile o difficile? (in **P** oppure si sa/congettura che non è in **P**)
  - per quelli facili il calcolo tradizionale è molto competitivo, costa decisamente di meno

  Se è difficile:
- La dimensione del problema è gestibile da algoritmi tradizionali?
  - se si, possono essere convenienti (dipende dallo speedup quantistico)
- Altrimenti, è gestibile con un algoritmo quantistico?
  - se si, allora conviene usare QC
- Altrimenti bisogna adottare una soluzione approssimata
  - di solito richiedono tempo polinomiale → gli algoritmi tradizionali ritornano competitivi
  - a meno di risultati su approssimazione quantistica al momento inesistenti

Applicazioni di Grover a problemi in NP

Domande da porsi per ogni problema specifico (rapporto costo-prestazioni):

- È facile o difficile? (in **P** oppure si sa/congettura che non è in **P**)
  - per quelli facili il calcolo tradizionale è molto competitivo, costa decisamente di meno

Se è difficile:

- La dimensione del problema è gestibile da algoritmi tradizionali?
  - se si, possono essere convenienti (dipende dallo speedup quantistico)
- Altrimenti, è gestibile con un algoritmo quantistico?
  - se si, allora conviene usare QC
- Altrimenti bisogna adottare una soluzione approssimata
  - di solito richiedono tempo polinomiale → gli algoritmi tradizionali ritornano competitivi
  - a meno di risultati su approssimazione quantistica al momento inesistenti

Applicazioni di Grover a problemi in NP

Domande da porsi per ogni problema specifico (rapporto costo-prestazioni):

- È facile o difficile? (in **P** oppure si sa/congettura che non è in **P**)
  - per quelli facili il calcolo tradizionale è molto competitivo, costa decisamente di meno

Se è difficile:

- La dimensione del problema è gestibile da algoritmi tradizionali?
  - se si, possono essere convenienti (dipende dallo speedup quantistico)
- Altrimenti, è gestibile con un algoritmo quantistico?
  - se si, allora conviene usare QC
- Altrimenti bisogna adottare una soluzione approssimata
  - di solito richiedono tempo polinomiale → gli algoritmi tradizionali ritornano competitivi
  - a meno di risultati su approssimazione quantistica al momento inesistenti

Applicazioni di Grover a problemi in NP

Domande da porsi per ogni problema specifico (rapporto costo-prestazioni):

- È facile o difficile? (in **P** oppure si sa/congettura che non è in **P**)
    - per quelli facili il calcolo tradizionale è molto competitivo, costa decisamente di meno

Se è difficile:

- La dimensione del problema è gestibile da algoritmi tradizionali?
    - se si, possono essere convenienti (dipende dallo speedup quantistico)
- Altrimenti, è gestibile con un algoritmo quantistico?
    - se si, allora conviene usare QC
- Altrimenti bisogna adottare una soluzione approssimata
    - di solito richiedono tempo polinomiale → gli algoritmi tradizionali ritornano competitivi
    - a meno di risultati su approssimazione quantistica al momento inesistenti

Applicazioni di Grover a problemi in NP

Domande da porsi per ogni problema specifico (rapporto costo-prestazioni):

- È facile o difficile? (in **P** oppure si sa/congettura che non è in **P**)
  - per quelli facili il calcolo tradizionale è molto competitivo, costa decisamente di meno

Se è difficile:

- La dimensione del problema è gestibile da algoritmi tradizionali?
  - se si, possono essere convenienti (dipende dallo speedup quantistico)
- Altrimenti, è gestibile con un algoritmo quantistico?
  - se si, allora conviene usare QC
- Altrimenti bisogna adottare una soluzione approssimata
  - di solito richiedono tempo polinomiale $\rightarrow$ gli algoritmi tradizionali ritornano competitivi
  - a meno di risultati su approssimazione quantistica al momento inesistenti

Applicazioni dell'algoritmo quantistico per la fattorizzazione veloce

–

Impatto sulla sicurezza informatica

Premessa: Crittografia simmetrica e asimmetrica

- Nella crittografia simmetrica, si usa la stessa chiave per cifrare e decifrare
  - nelle comunicazioni a due, bisogna prima condividere la chiave, operazione *rischiosa*
- Nella crittografia asimmetrica, si usano due chiavi: una per cifrare e una per decifrare
  - nelle comunicazioni, una chiave (pubblica) si può dare a tutti per cifrare; l'altra (privata) si usa per decifrare
  - da cui il nome alternativo *cifratura a chiave pubblica*
  - per la firma digitale si cifra con la chiave privata
- La sicurezza dei cifrari asimmetrici è *computazionale* e fondata sulle congetture della teoria della complessità classica
  - non sono noti algoritmi polinomiali per decifrare il messaggio
  - si dimostra che se cifratura e decifratura sono "pratiche" (in **P**) allora gli attacchi sono al massimo in **NP**

Premessa: Crittografia simmetrica e asimmetrica

- Nella crittografia simmetrica, si usa la stessa chiave per cifrare e decifrare
  - nelle comunicazioni a due, bisogna prima condividere la chiave, operazione *rischiosa*
- Nella crittografia asimmetrica, si usano due chiavi: una per cifrare e una per decifrare
  - nelle comunicazioni, una chiave (pubblica) si può dare a tutti per cifrare; l'altra (privata) si usa per decifrare
  - da cui il nome alternativo *cifratura a chiave pubblica*
  - per la firma digitale si cifra con la chiave privata
- La sicurezza dei cifrari asimmetrici è *computazionale* e fondata sulle congetture della teoria della complessità classica
  - non sono noti algoritmi polinomiali per decifrare il messaggio
  - si dimostra che se cifratura e decifratura sono "pratiche" (in **P**) allora gli attacchi sono al massimo in **NP**

Premessa: Crittografia simmetrica e asimmetrica

- Nella crittografia simmetrica, si usa la stessa chiave per cifrare e decifrare
  - nelle comunicazioni a due, bisogna prima condividere la chiave, operazione *rischiosa*
- Nella crittografia asimmetrica, si usano due chiavi: una per cifrare e una per decifrare
  - nelle comunicazioni, una chiave (pubblica) si può dare a tutti per cifrare; l'altra (privata) si usa per decifrare
  - da cui il nome alternativo *cifratura a chiave pubblica*
  - per la firma digitale si cifra con la chiave privata
- La sicurezza dei cifrari asimmetrici è *computazionale* e fondata sulle congetture della teoria della complessità classica
  - non sono noti algoritmi polinomiali per decifrare il messaggio
  - si dimostra che se cifratura e decifratura sono "pratiche" (in **P**) allora gli attacchi sono al massimo in **NP**

Premessa: Crittografia simmetrica e asimmetrica

- Nella crittografia simmetrica, si usa la stessa chiave per cifrare e decifrare
  - nelle comunicazioni a due, bisogna prima condividere la chiave, operazione *rischiosa*
- Nella crittografia asimmetrica, si usano due chiavi: una per cifrare e una per decifrare
  - nelle comunicazioni, una chiave (pubblica) si può dare a tutti per cifrare; l'altra (privata) si usa per decifrare
  - da cui il nome alternativo *cifratura a chiave pubblica*
  - per la firma digitale si cifra con la chiave privata
- La sicurezza dei cifrari asimmetrici è *computazionale* e fondata sulle congetture della teoria della complessità classica
  - non sono noti algoritmi polinomiali per decifrare il messaggio
  - si dimostra che se cifratura e decifratura sono "pratiche" (in **P**) allora gli attacchi sono al massimo in **NP**

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) =$ $= e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) = $ $= e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) =$ $= e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) =$ $= e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) = = e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) =$ $= e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) =$ $= e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

Esempio: RSA (diffusissimo crittosistema a chiave pubblica)

1. Generazione chiavi pubblica e privata
   - Scegliere a caso due primi p e q (molto grandi) e porre $n = pq$
   - Calcolare la funzione toziente di Eulero $\Phi(n) = (p-1)(q-1)$
   - Scegliere a caso un generatore $e \in \mathbb{Z}_n^*$
   - Calcolare il suo inverso moltiplicativo $d = e^{-1} \mod \Phi(n) =$ $= e^{\Phi(n)-1} \mod \Phi(n)$
   - Chiave pubblica: (e, n). Chiave privata: (d, n)
2. Cifratura messaggio m (codificato come un intero)
   - $c = m^e \mod n$
3. Decifratura messaggio cifrato c
   - $m = c^d \mod n$

Nota: tutte operazioni eseguibili in tempo polinomiale a partire da p e q

Premessa: Crittografia simmetrica e asimmetrica

In che senso RSA è computazionalmente sicuro?

Non sono noti algoritmi polinomiali classici per:

- il calcolo del logaritmo discreto
  - dati $c^d \mod n$, c ed n, calcolare d
  - ovvero ricavare la chiave privata da una firma o dalla decifratura di un messaggio
- la fattorizzazione di interi
  - dato n, ricavare p e q
  - dalla chiave pubblica (e, n) si potrebbe calcolare $d = e^{(p-1)(q-1)-1} \mod n$ in tempo polinomiale
- il calcolo di $\Phi(n)$ a partire da n
  - permetterebbe di calcolare $d = e^{\Phi(n)-1} \mod n$ in tempo polinomiale
  - si dimostra che ha la stessa complessità della fattorizzazione

Premessa: Crittografia simmetrica e asimmetrica

In che senso RSA è computazionalmente sicuro?

Non sono noti algoritmi polinomiali classici per:

- il calcolo del logaritmo discreto
  - dati $c^d \mod n$, c ed n, calcolare d
  - ovvero ricavare la chiave privata da una firma o dalla decifratura di un messaggio
- la fattorizzazione di interi
  - dato n, ricavare p e q
  - dalla chiave pubblica (e, n) si potrebbe calcolare $d = e^{(p-1)(q-1)-1} \mod n$ in tempo polinomiale
- il calcolo di $\Phi(n)$ a partire da n
  - permetterebbe di calcolare $d = e^{\Phi(n)-1} \mod n$ in tempo polinomiale
  - si dimostra che ha la stessa complessità della fattorizzazione

Premessa: Crittografia simmetrica e asimmetrica

In che senso RSA è computazionalmente sicuro?

Non sono noti algoritmi polinomiali classici per:

- il calcolo del logaritmo discreto
  - dati $c^d \mod n$, c ed n, calcolare d
  - ovvero ricavare la chiave privata da una firma o dalla decifratura di un messaggio
- la fattorizzazione di interi
  - dato n, ricavare p e q
  - dalla chiave pubblica (e, n) si potrebbe calcolare $d = e^{(p-1)(q-1)-1} \mod n$ in tempo polinomiale
- il calcolo di $\Phi(n)$ a partire da n
  - permetterebbe di calcolare $d = e^{\Phi(n)-1} \mod n$ in tempo polinomiale
  - si dimostra che ha la stessa complessità della fattorizzazione

Premessa: Crittografia simmetrica e asimmetrica

In che senso RSA è computazionalmente sicuro?

Non sono noti algoritmi polinomiali classici per:

- il calcolo del logaritmo discreto
  - dati $c^d \mod n$, c ed n, calcolare d
  - ovvero ricavare la chiave privata da una firma o dalla decifratura di un messaggio
- la fattorizzazione di interi
  - dato n, ricavare p e q
  - dalla chiave pubblica (e, n) si potrebbe calcolare $d = e^{(p-1)(q-1)-1} \mod n$ in tempo polinomiale
- il calcolo di $\Phi(n)$ a partire da n
  - permetterebbe di calcolare $d = e^{\Phi(n)-1} \mod n$ in tempo polinomiale
  - si dimostra che ha la stessa complessità della fattorizzazione

Premessa: Crittografia simmetrica e asimmetrica

In che senso RSA è computazionalmente sicuro?

Non sono noti algoritmi polinomiali classici per:

- il calcolo del logaritmo discreto
  - dati $c^d \mod n$, c ed n, calcolare d
  - ovvero ricavare la chiave privata da una firma o dalla decifratura di un messaggio
- la fattorizzazione di interi   Shor permette(rà) di farlo
  - dato n, ricavare p e q
  - dalla chiave pubblica (e, n) si potrebbe calcolare $d = e^{(p-1)(q-1)-1} \mod n$ in tempo polinomiale
- il calcolo di $\Phi(n)$ a partire da n
  - permetterebbe di calcolare $d = e^{\Phi(n)-1} \mod n$ in tempo polinomiale
  - si dimostra che ha la stessa complessità della fattorizzazione

Impatto dell'algoritmo di Schor sulla sicurezza

- Riguarderà i principali crittosistemi a chiave pubblica (RSA, Diffie-Helmann,...) perchè non basati su problemi NP-completi
  - di conseguenza i protocolli SSL, HTTPS,... e le firme digitali
  - alternative: problemi NP-completi o crittografia quantistica
- Invece i crittosistemi simmetrici sembrano "immuni"[2]
  - l'algoritmo di Grover accelera gli attacchi a forza bruta (ricerca nello spazio delle chiavi) di un fattore $2^{n/2}$ (n=lunghezza chiave)
  - ma il costo resta $O(2^{n/2})$
  - raddoppiando la lunghezza della chiave si ripristina il livello di sicurezza (costo aggiuntivo (de)cifratura solo polinomiale)
- Nota bene: in tutti i protocolli la cifratura a chiave pubblica viene utilizzata solo nell'handshake per condividere una chiave simmetrica di sessione; poi si passa a cifratura simmetrica

[2] https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf, Sez. "Quantum computing threats"

Impatto dell'algoritmo di Schor sulla sicurezza

- Riguarderà i principali crittosistemi a chiave pubblica (RSA, Diffie-Helmann,...) perchè non basati su problemi NP-completi
  - di conseguenza i protocolli SSL, HTTPS,... e le firme digitali
  - alternative: problemi NP-completi o crittografia quantistica
- Invece i crittosistemi simmetrici sembrano "immuni"[2]
  - l'algoritmo di Grover accelera gli attacchi a forza bruta (ricerca nello spazio delle chiavi) di un fattore $2^{n/2}$ (n=lunghezza chiave)
  - ma il costo resta $O(2^{n/2})$
  - raddoppiando la lunghezza della chiave si ripristina il livello di sicurezza (costo aggiuntivo (de)cifratura solo polinomiale)
- Nota bene: in tutti i protocolli la cifratura a chiave pubblica viene utilizzata solo nell'handshake per condividere una chiave simmetrica di sessione; poi si passa a cifratura simmetrica

[2] https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf, Sez. "Quantum computing threats"

Impatto dell'algoritmo di Schor sulla sicurezza

- Riguarderà i principali crittosistemi a chiave pubblica (RSA, Diffie-Helmann,...) perchè non basati su problemi NP-completi
  - di conseguenza i protocolli SSL, HTTPS,... e le firme digitali
  - alternative: problemi NP-completi o crittografia quantistica
- Invece i crittosistemi simmetrici sembrano "immuni"[2]
  - l'algoritmo di Grover accelera gli attacchi a forza bruta (ricerca nello spazio delle chiavi) di un fattore $2^{n/2}$ (n=lunghezza chiave)
  - ma il costo resta $O(2^{n/2})$
  - raddoppiando la lunghezza della chiave si ripristina il livello di sicurezza (costo aggiuntivo (de)cifratura solo polinomiale)
- Nota bene: in tutti i protocolli la cifratura a chiave pubblica viene utilizzata solo nell'handshake per condividere una chiave simmetrica di sessione; poi si passa a cifratura simmetrica

[2] https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf, Sez. "Quantum computing threats"

Impatto dell'algoritmo di Schor sulla sicurezza

- Riguarderà i principali crittosistemi a chiave pubblica (RSA, Diffie-Helmann,...) perchè non basati su problemi NP-completi
  - di conseguenza i protocolli SSL, HTTPS,... e le firme digitali
  - alternative: problemi NP-completi o crittografia quantistica
- Invece i crittosistemi simmetrici sembrano "immuni"[2]
  - l'algoritmo di Grover accelera gli attacchi a forza bruta (ricerca nello spazio delle chiavi) di un fattore $2^{n/2}$ (n=lunghezza chiave)
  - ma il costo resta $O(2^{n/2})$
  - raddoppiando la lunghezza della chiave si ripristina il livello di sicurezza (costo aggiuntivo (de)cifratura solo polinomiale)
- Nota bene: in tutti i protocolli la cifratura a chiave pubblica viene utilizzata solo nell'handshake per condividere una chiave simmetrica di sessione; poi si passa a cifratura simmetrica

[2] https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf, Sez. "Quantum computing threats"

Impatto dell'algoritmo di Schor sulla sicurezza

- Riguarderà i principali crittosistemi a chiave pubblica (RSA, Diffie-Helmann,...) perchè non basati su problemi NP-completi
  - di conseguenza i protocolli SSL, HTTPS,... e le firme digitali
  - alternative: problemi NP-completi o crittografia quantistica
- Invece i crittosistemi simmetrici sembrano "immuni"[2]
  - l'algoritmo di Grover accelera gli attacchi a forza bruta (ricerca nello spazio delle chiavi) di un fattore $2^{n/2}$ (n=lunghezza chiave)
  - ma il costo resta $O(2^{n/2})$
  - raddoppiando la lunghezza della chiave si ripristina il livello di sicurezza (costo aggiuntivo (de)cifratura solo polinomiale)
- Nota bene: in tutti i protocolli la cifratura a chiave pubblica viene utilizzata solo nell'handshake per condividere una chiave simmetrica di sessione; poi si passa a cifratura simmetrica

[2] https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf, Sez. "Quantum computing threats"

Conclusioni

Take-away messages:

- *No silver bullet*: i problemi difficili (da **NP**-completi in su) resteranno plausibilmente difficili anche per il calcolo quantistico
  - quelli da 2**EXP** in su (ad es. ontology-based reasoning) sicuramente resteranno difficili
- *Si avranno comunque benefici*: aumento della taglia dei problemi NP-completi risolubili (usando Grover)
  - e qualche problema (ritenuto) nè **NP**-completo nè in **P** potrebbe diventare più facile (come la fattorizzazione)
- *Impatto futuro sulla sicurezza informatica*: Gli attuali crittosistemi asimmetrici (a chiave pubblica) potranno essere velocemente violati (quando i problemi tecnologici del QC saranno stati risolti).
  - bisognerà ripensare firme digitali e scambio iniziale di chiavi simmetriche nei protocolli per ripristinare la sicurezza di Internet

Conclusioni

Take-away messages:

- *No silver bullet*: i problemi difficili (da **NP**-completi in su) resteranno plausibilmente difficili anche per il calcolo quantistico
  - quelli da 2**EXP** in su (ad es. ontology-based reasoning) sicuramente resteranno difficili
- *Si avranno comunque benefici*: aumento della taglia dei problemi NP-completi risolubili (usando Grover)
  - e qualche problema (ritenuto) nè **NP**-completo nè in **P** potrebbe diventare più facile (come la fattorizzazione)
- *Impatto futuro sulla sicurezza informatica*: Gli attuali crittosistemi asimmetrici (a chiave pubblica) potranno essere velocemente violati (quando i problemi tecnologici del QC saranno stati risolti).
  - bisognerà ripensare firme digitali e scambio iniziale di chiavi simmetriche nei protocolli per ripristinare la sicurezza di Internet

Conclusioni

Take-away messages:

- *No silver bullet*: i problemi difficili (da **NP**-completi in su) resteranno plausibilmente difficili anche per il calcolo quantistico
    - quelli da 2**EXP** in su (ad es. ontology-based reasoning) sicuramente resteranno difficili
- *Si avranno comunque benefici*: aumento della taglia dei problemi NP-completi risolubili (usando Grover)
    - e qualche problema (ritenuto) nè **NP**-completo nè in **P** potrebbe diventare più facile (come la fattorizzazione)
- *Impatto futuro sulla sicurezza informatica*: Gli attuali crittosistemi asimmetrici (a chiave pubblica) potranno essere velocemente violati (quando i problemi tecnologici del QC saranno stati risolti).
    - bisognerà ripensare firme digitali e scambio iniziale di chiavi simmetriche nei protocolli per ripristinare la sicurezza di Internet

Conclusioni

Take-away messages:

- *No silver bullet*: i problemi difficili (da **NP**-completi in su) resteranno plausibilmente difficili anche per il calcolo quantistico
  - quelli da 2**EXP** in su (ad es. ontology-based reasoning) sicuramente resteranno difficili
- *Si avranno comunque benefici*: aumento della taglia dei problemi NP-completi risolubili (usando Grover)
  - e qualche problema (ritenuto) nè **NP**-completo nè in **P** potrebbe diventare più facile (come la fattorizzazione)
- *Impatto futuro sulla sicurezza informatica*: Gli attuali crittosistemi asimmetrici (a chiave pubblica) potranno essere velocemente violati (quando i problemi tecnologici del QC saranno stati risolti).
  - bisognerà ripensare firme digitali e scambio iniziale di chiavi simmetriche nei protocolli per ripristinare la sicurezza di Internet

Esercizi

- Conviene risolvere SORT con un computer quantistico?
- Conviene risolvere KNAPSACK con un computer quantistico?
- Conviene sviluppare algoritmi quantistici di crittografia?

Esercizi

- Conviene risolvere SORT con un computer quantistico?
- Conviene risolvere KNAPSACK con un computer quantistico?
- Conviene sviluppare algoritmi quantistici di crittografia?

Esercizi

- Conviene risolvere SORT con un computer quantistico?
- Conviene risolvere KNAPSACK con un computer quantistico?
- Conviene sviluppare algoritmi quantistici di crittografia?

potete contattarmi all'indirizzo

piero.bonatti@unina.it