Risoluzione del problema Weighted Feedback Vertex Set tramite un Hybrid Genetic Algorithm

Progetto di Artificial Intelligence

Daniele Materia
Prof. Vincenzo Cutello, Prof. Mario Francesco Pavone

10 Marzo 2025

Indice

1 Introduzione

Weighted Feedback Vertex Set (WFVS) è un noto problema di ottimizzazione combinatoriale su grafi per il quale non esistono algoritmi esatti capaci di risolverlo in tempi ragionevoli, specialmente per grafi molto complessi. Di conseguenza, l'utilizzo di metodi alternativi agli algoritmi deterministici, come algoritmi iterativi e approssimati, risulta essere un'ottima strategia per trovare soluzioni che siano sufficientemente buone in un tempo accettabile.

In questo progetto, per risolvere *WFVS*, è stato utilizzato un *Hybrid Genetic Algorithm* (HGA). Questa tipologia di algoritmo combina le caratteristiche di un algoritmo genetico classico ad una strategia di ricerca locale, che ha lo scopo di migliorare la qualità delle soluzioni ottenute.

Nella sezione 2, viene introdotto formalmente il problema *Feedback Vertex Set* (FVS), insieme alla sua versione pesata *Weighted Feedback Vertex Set* (WFVS). Inoltre, vengono brevemente descritte alcune delle applicazioni pratiche di questo problema.

La sezione 3 descrive in dettaglio l'algoritmo sviluppato per affrontare il problema, concentrandosi sulle caratteristiche specifiche dell'*Hybrid Genetic Algorithm* e sul suo funzionamento. Vengono illustrate le parti principali dell'algoritmo, come l'inizializzazione della popolazione, la selezione, il crossover, la mutazione e l'uso della ricerca locale per migliorare le soluzioni. Inoltre, vengono giustificate le scelte implementative prese durante lo svolgimento del progetto. A seguire sono illustrati gli pseudocodici dell'algoritmo.

Nelle sezioni 4 e 5 vengono discussi il protocollo sperimentale seguito, la combinazione di parametri dell'algoritmo che è stata scelta e i risultati ottenuti. Sono presenti anche alcuni esempi di grafi aciclici ottenuti dalle soluzioni fornite dall'algoritmo, insieme ai relativi plot di convergenza.

Infine seguono le conclusioni, in cui si discutono i punti di forza del progetto ed eventuali sviluppi e miglioramenti futuri.

2 Il problema Feedback Vertex Set

2.1 Definizione formale

Feedback Vertex Set (FVS) è un problema di ottimizzazione combinatoriale che consiste nel trovare un sottoinsieme di vertici di un grafo la cui rimozione porta il grafo stesso ad essere aciclico.
Formalmente, dato un grafo non diretto $G = (V, E)$, un FVS di G è un sottoinsieme $S \subseteq V$ di vertici la cui rimozione, insieme a quella degli archi incidenti ai vertici in questione, fa sì che il grafo risultante $G' = (V \setminus S, E \setminus S)$ non presenti alcun ciclo.

2.1.1 Weighted Feedback Vertex Set

Sia $w : V \to \mathbb{R}^+$ una funzione peso che associa un valore positivo ad ogni vertice $v \in V$ del grafo G, segue che il peso del FVS $S \subseteq V$ sarà la somma dei pesi vertici al suo interno: $\sum_{i=1}^{|S|} w(v_i)$.
Nel caso di grafi con vertici pesati, quindi, si parla di **Weighted Feedback Vertex Set** (WFVS), e il problema richiede di trovare un WFVS S la cui somma dei pesi $\sum_{i=1}^{|S|} w(v_i)$ è minima.

2.1.2 *NP-hardness* di WFVS

Si può dimostrare che Weighted Feedback Vertex Set appartiene alla classe *NP-hard* [1], il che rende poco praticabile la ricerca di una soluzione esatta, soprattutto per grafi complessi. Di conseguenza, è spesso conveniente ricorrere all'uso di **algoritmi approssimati**, come le **metaeuristiche**, in modo da trovare soluzioni non necessariamente esatte, ma pur sempre di buona qualità, in tempi ragionevoli.

2.2 Applicazioni reali di FVS

Un'interessante applicazione di questo problema la troviamo nei **sistemi operativi**, precisamente nel recupero di situazioni di stallo tra processi (deadlock): nel grafo delle attese di un sistema operativo, ogni ciclo diretto corrisponde a una situazione di deadlock. In questo contesto, un FVS minimo rappresenta il numero minimo di processi da terminare per eliminare il deadlock.

Quella appena brevemente descritta non è la sola applicazione pratica di questo problema, in quanto il Feedback Vertex Set trova impiego in altri ambiti, per esempio:

- **Microelettronica**: FVS ha applicazioni nel design di chip VLSI [2].
- **Teoria della complessità**: alcuni problemi *NP-hard* in generale possono essere risolti in tempo polinomiale per grafi con numero FVS limitato.

3 Algoritmo sviluppato

Per risolvere il problema, si è scelto di sviluppare un *algoritmo genetico ibrido* (**Hybrid Genetic Algorithm**, HGA). Come dice il nome stesso, un HGA è un particolare tipo di algoritmo genetico che integra una strategia di ricerca locale per affinare le soluzioni trovate dalla parte genetica dell'algoritmo.

Gli algoritmi genetici (o evolutivi) sono metaeuristiche che imitano i processi di evoluzione naturale per risolvere problemi di ricerca globale e si caratterizzano per la capacità di risolvere problemi anche con scarsa conoscenza del dominio [3].

Questi algoritmi riescono a esplorare efficacemente lo spazio delle soluzioni tramite operatori di *selezione*, *crossover* e *mutazione* applicati ad una popolazione di individui (soluzioni) in continua evoluzione.

La scelta di implementare un HGA deriva dal voler combinare i punti di forza di algoritmi genetici e ricerca locale. Gli algoritmi genetici sono capaci di esplorare ampie aree dello spazio di ricerca grazie all'uso di una popolazione di individui e degli operatori genetici. Tuttavia, possono richiedere un numero elevato di iterazioni per convergere a soluzioni ottimali. Inoltre, la sola evoluzione genetica potrebbe non essere sufficiente per affinare le soluzioni. Integrando la ricerca locale è possibile migliorare ulteriormente le soluzioni individuate, aumentandone la qualità. Quindi, questa combinazione sfrutta sia l'ampia esplorazione globale dello spazio di ricerca degli algoritmi genetici che la focalizzazione su soluzioni locali promettenti della ricerca locale.

3.1 Caratteristiche dell'algoritmo

3.1.1 Inizializzazione della popolazione

La popolazione iniziale viene generata attraverso un processo iterativo nel quale, per ogni individuo della popolazione, si rimuovono tanti vertici dal grafo di partenza $G = (V, E)$ quanto basta affinché G risulti essere *privo di cicli*. I vertici rimossi costituiranno l'individuo in questione.

In particolare, vengono identificati tutti i cicli presenti in G, e per ognuno di essi viene scelto randomicamente un vertice da rimuovere, spezzando il ciclo.

Usando questa strategia ogni individuo della popolazione iniziale è già una soluzione valida. Inoltre, la stocasticità della scelta dei vertici da rimuovere introduce diversità nella popolazione iniziale, specialmente nei casi in cui il grafo di partenza sia fortemente connesso e abbia un alto numero di vertici.

3.1.2 Funzione di fitness

La funzione obiettivo è definita come:

$$\textbf{\textit{fitness}}(S) = \sum_{v \in S} w(v) + \alpha \cdot num\_cycles \tag{1}$$

dove $\alpha = 10^4$ e num_cycles indica il numero di cicli presenti nel grafo indotto dal grafo di partenza G, ottenuto a partire dall'individuo S di cui si valuta la fitness. La penalità $\alpha \cdot num\_cycles$ varia in base al numero di cicli rimanenti nel grafo indotto, assicurando che l'algoritmo privilegi soluzioni prive di cicli[1].

3.1.3 Selezione

La selezione dei genitori avviene tramite torneo di $k = 5$ individui scelti casualmente, in cui quello col miglior valore di fitness viene a sua volta scelto con probabilità $p = 0.7$. Altrimenti, con probabilità $(1 - p)$, viene scelto un altro individuo casuale tra i 4 rimanenti . Questo approccio bilancia selezione elitista ed esplorazione dello spazio delle soluzioni, permettendo anche a soluzioni subottimali di contribuire all'evoluzione della popolazione.

3.1.4 Crossover e Mutazione

- L'operatore di **crossover** mantiene i vertici comuni tra i due genitori e aggiunge un sottoinsieme di vertici dalla loro differenza simmetrica. Precisamente, vengono aggiunti $\lfloor \frac{|\textit{diff}|}{2} \rfloor$ vertici all'individuo in questione, dove $|\textit{diff}|$ è la cardinalità della differenza simmetrica dei due genitori. Questa strategia assicura che il figlio mantenga elementi strutturali di entrambi i genitori, preservando informazioni utili.
- La **mutazione** avviene con probabilità $P_{mut} = 0.005$, rimuovendo o aggiungendo un vertice scelto in maniera casuale (*flip* di un gene). Introducendo una certa variabilità nella popolazione si argina il problema dell'intrappolamento prematuro in eventuali ottimi locali.

3.1.5 Ricerca locale

Dopo crossover e mutazione, ogni nuovo individuo viene affinato con una *ricerca locale* basata su **Tabu Search**: un metodo di ricerca locale che, a differenza di altre tecniche, cerca di evitare che la ricerca si fermi in ottimi locali. Questa caratteristica viene ottenuta facendo in modo che l'algoritmo di ricerca non torni sui suoi passi, proibendo le mosse recentemente eseguite grazie ad una memoria a breve termine detta *tabu list*, accettando anche eventuali mosse peggiorative con l'obiettivo di sfuggire da eventuali ottimi locali.

La Tabu Search implementata in questo progetto, a partire da un individuo S, ne genera il vicinato scegliendo casualmente un campione corrispondente

[1] Il valore di α è stato scelto empiricamente per garantire che le penalità sui cicli siano abbastanza alte da essere sempre prioritarie rispetto alla minimizzazione del peso complessivo.

al 5% dei vertici del grafo di partenza, e per ognuno di esso viene chiamata la funzione *flip* su S per generare un insieme di suoi vicini. A partire dai vicini, viene valutato il migliore di essi, il quale poi viene confrontato con la migliore soluzione trovata finora (a patto che il candidato non sia presente nella tabu_list).

3.2 Pseudocodici

Di seguito vengono riportati gli pseudocodici dell'HGA e dei suoi principali componenti (funzione di fitness, operatori genetici, ricerca locale).

Algorithm 1 Hybrid Genetic Algorithm (HGA)

Input: Grafo G, funzione peso w, dimensione popolazione pop_size, numero generazioni GEN_NUM, tasso di mutazione P_{mut}
Inizializzazione: Genera la *popolazione* iniziale costituita da pop_size soluzioni valide
Calcola la fitness iniziale per tutti gli individui della popolazione
for $gen = 1$ **to** GEN_NUM **do**
    $nuova\_popolazione \leftarrow \emptyset$
    for $i = 1$ **to** pop_size **do**
        $p_1 \leftarrow$ **Selection**$(popolazione)$
        $p_2 \leftarrow$ **Selection**$(popolazione)$
        $c \leftarrow$ **Crossover**(p_1, p_2)
        $c \leftarrow$ **Mutation**(c, G, P_{mut})
        $c \leftarrow$ **Tabu Search**(G, c, w)
        Aggiungi c a $nuova\_popolazione$
    end for
    $popolazione = popolazione \cup nuova\_popolazione$
    Ordina $popolazione$ in base alla fitness e conserva i migliori pop_size individui
end for
return Miglior individuo trovato e la sua fitness

Algorithm 2 Funzione di fitness

Input: Grafo G, individuo S, funzione peso w
$H \leftarrow$ sottografo di G indotto dai vertici $G - S$
$num\_cycles \leftarrow$ numero di cicli in H
$fitness\_value \leftarrow (10^4) \cdot num\_cycles + \sum_{v \in S} w[v]$
return $fitness\_value$

Algorithm 3 Selection - k-Tournament

Input: Popolazione P, numero candidati k, probabilità di selezione del migliore p
Seleziona casualmente un sottoinsieme S di k individui da P
Ordina S in base alla fitness (migliore per primo)
if random() $< p$ **then**
    return primo individuo di S (migliore)
else
    Seleziona casualmente uno tra gli altri $k-1$ individui in S
    return individuo selezionato
end if

Algorithm 4 Crossover

Input: Genitori p_1, p_2
$common \leftarrow p_1 \cap p_2$ ▷ Intersezione dei vertici dei genitori
$diff \leftarrow p_1 \triangle p_2$ ▷ Differenza simmetrica dei vertici dei genitori
$sample\_diff \leftarrow$ scegli $\lfloor \frac{|diff|}{2} \rfloor$ vertici da $diff$
return $common \cup sample\_diff$

Algorithm 5 Mutation - Modifica casuale di un individuo

Input: Individuo c, grafo G, tasso di mutazione P_{mut}
if random() $< P_{mut}$ **then**
    Seleziona un vertice casuale v da G
    Flip(v, c)
end if
return c ▷ Restituisco l'individuo mutato

Algorithm 6 Ricerca locale - Tabu Search

Input: Grafo G, individuo iniziale S, *max_iterations*
$tabu\_list\_size = \max(5, \lceil \sqrt[5]{|V|^2} \rceil)$
$current\_individual \leftarrow S$
$tabu\_list \leftarrow \emptyset$
for $iter = 1$ **to** *max_iterations* **do**
    $best\_candidate \leftarrow \emptyset$
    $best\_candidate\_fitness \leftarrow \infty$
    $neighborhood \leftarrow$ Scegli casualmente $\lceil |V| \cdot 0.05 \rceil$ vertici del grafo G
    for *node_to_flip* **in** *neighborhood* **do**
        $new\_candidate \leftarrow$ **Flip**$(node\_to\_flip, current\_individual)$
        if $new\_candidate \in tabu\_list$ **then continue**
        end if
        $new\_fitness \leftarrow$ **Fitness**$(G, new\_candidate, w)$
        if $new\_fitness < best\_candidate\_fitness$ **then**
            $best\_candidate \leftarrow new\_candidate$
            $best\_candidate\_fitness \leftarrow new\_fitness$
        end if
    end for
    if $best\_candidate = \emptyset$ **then break**
    end if
    $current\_individual \leftarrow best\_candidate$
    if $best\_candidate\_fitness < best\_fitness$ **then**
        $best\_individual \leftarrow best\_candidate$
        $best\_fitness \leftarrow best\_candidate\_fitness$
    end if
    Aggiungi *best_candidate* a *tabu_list*
    if $|tabu\_list| > tabu\_list\_size$ **then**
        Rimuovi l'elemento meno recente da *tabu_list*
    end if
end for
return *best_solution*

Algorithm 7 Funzione Flip

Input: vertice v**, individuo** c
if $v \in c$ **then**
    Rimuovi v da c
else
    Aggiungi v a c
end if

4 Protocollo sperimentale

4.1 Ambiente sperimentale

Per lo sviluppo dell'algoritmo è stato scelto Python, linguaggio di programmazione ad alto livello ed orientato agli oggetti, tra i più utilizzati per la sua versatilità e la ricchezza di librerie.

4.1.1 Librerie utilizzate

Sono state utilizzate le seguenti librerie Python:

- **NetworkX**: questa libreria fornisce strumenti per la creazione, manipolazione e analisi di grafi.
- **Matplotlib**: è la libreria più utilizzata per la creazione e visualizzazione di grafici.
- **Pandas**: libreria per l'analisi dei dati, in questo progetto viene usata per l'organizzazione e il salvataggio dei risultati sperimentali.
- **Random**: come dice il nome stesso, è una libreria per generare numeri pseudo-casuali. È essenziale, data la presenza preminente di comportamenti stocastici nell'algoritmo sviluppato.

4.2 Istanze di test

L'algoritmo è stato testato su un insieme di 6 diverse istanze di benchmark. Ognuna consiste in un *grafo non diretto* (con vertici pesati) di differente topologia, dimensione ($|V|$), densità e funzione peso.

In particolare, le istanze includono 3 grafi con topologia a griglia e 3 con topologia casuale. Per ognuno, sono presenti 5 istanze con caratteristiche identiche ma differente funzione peso w. In totale, le istanze sono 30.

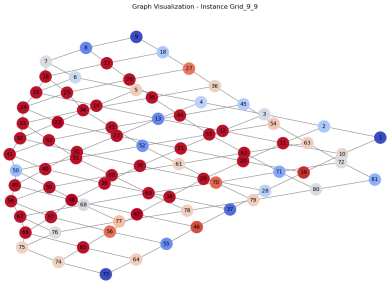


Figura 1: Esempio di istanza con topologia a griglia

Parametro	Valore
Numero di generazioni	250
Dimensione della popolazione	25
k-Tournament selection	5
Probabilità di selezione del miglior individuo	70%
Probabilità di mutazione	0,5%
Dimensione della tabu list	$\max(5, \lceil \sqrt[5]{\lvert V \rvert^2} \rceil)$
Iterazioni massime Tabu Search	10

Tabella 1: Scelta dei parametri dell'Hybrid Genetic Algorithm

4.3 Parametri dell'algoritmo

Questa particolare configurazione è stata determinata attraverso una serie di test preliminari, con l'obiettivo di ottimizzare le prestazioni dell'algoritmo in termini di **velocità di convergenza** e **qualità delle soluzioni** ottenute.

La popolazione è stata fissata a 25 individui e il numero di generazioni a 250 per garantire diversità genetica ed esplorare efficacemente lo spazio delle soluzioni.

La k-Tournament selection con k=5 e una probabilità del 70% di selezionare il miglior individuo aiuta a guidare l'evoluzione verso soluzioni ottimali, cercando di evitare al contempo una convergenza prematura in un ottimo locale.

La probabilità di mutazione dello 0,5% introduce variazioni genetiche che aiutano a mantenere la diversità nella popolazione, riducendo il rischio di stagnazione in ottimi locali.

Infine, l'utilizzo della ricerca locale tramite Tabu Search, con una tabu list di dimensione variabile e un massimo di 10 iterazioni, consente di affinare le soluzioni candidate, migliorando l'efficacia dell'algoritmo nel trovare soluzioni di alta qualità. La funzione scelta per far variare la *tabu_list_size* è stata scelta in modo da farla crescere all'aumentare del numero di vertici del grafo di partenza. In questo modo, essendo lo spazio di ricerca più grande, si cerca di evitare di tornare a soluzioni già esplorate poco prima, migliorando la diversificazione della ricerca.

Le scelte appena descritte sono volte a garantire un buon equilibrio tra **esplorazione** e **sfruttamento** nello spazio delle soluzioni, assicurando una convergenza verso soluzioni di qualità.

4.4 Caching dei valori di fitness

Per ridurre il tempo medio di convergenza dell'algoritmo si è cercato di identificare i suoi punti computazionalmente onerosi. È emerso che il principale collo di bottiglia dell'algoritmo consiste nel conteggio del numero di cicli nel grafo indotto da un individuo, facente parte della funzione di fitness.

Per ovviare al problema, nell'implementazione finale è stato introdotta una cache che memorizza i valori di fitness calcolati, evitando di ricalcolarla più volte

per la stessa soluzione. Questa semplice modifica ha portato a una riduzione significativa dei tempi di convergenza nei casi in cui l'algoritmo ritorna spesso su individui già valutati (soprattutto nel caso di istanze di tipo Grid).

5 Risultati sperimentali

Sono stati condotti 10 esperimenti indipendenti per ognuna delle istanze del problema a disposizione. La seguente tabella contiene il riepilogo dei risultati ottenuti insieme al tempo di esecuzione medio[2] ottenuto.

Tipologia	Media	Dev. std	Media valutazioni fitness	Tempo (s)
Grid_5_5	199.8	21.79	36178	2.38
Grid_7_7	255.4	7.8	70427	8.7
Grid_9_9	1168.8	142.28	125028	24.83
Rand_100_3069	1136.2	21.22	128209	24.09
Rand_100_841	1739.0	120.28	131117	24.0
Rand_200_3184	5255.4	843.48	272852	86.21

Tabella 2: Riepilogo dei risultati sperimentali

Dai valori medi di fitness ottenuti si osserva che le soluzioni trovate sono sempre valide.

Sebbene non si conoscano gli ottimi globali di queste specifiche istanze, i risultati ottenuti sono promettenti ed i tempi di esecuzione sono ottimi. Inoltre, l'algoritmo ha dimostrato una buona capacità di adattamento alle diverse tipologie di grafo, evidenziando una certa robustezza e versatilità.

5.1 Convergenza dell'algoritmo e qualità delle soluzioni

Osservando l'andamento della fitness lungo le generazioni, possiamo notare che le soluzioni trovate subiscono un progressivo miglioramento.

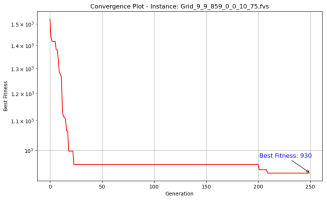
Talvolta, specialmente per istanze semplici, l'algoritmo converge dopo poche iterazioni, mentre nel caso di istanze complesse possono volerci molte più generazioni per arrivare a convergenza.

5.1.1 Esempi di convergenza e relative soluzioni

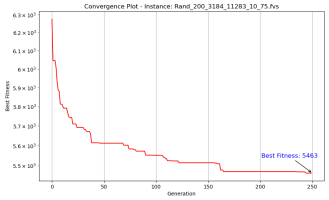
Di seguito sono riportati due esempi che illustrano il comportamento dell'algoritmo su istanze di diversa complessità. Nel primo caso (tipologia *Grid_9_9*), il grafo presenta una struttura a griglia e l'algoritmo converge dopo circa 210 generazioni. Nel secondo caso (tipologia *Rand_200_3184*), la casualità della topologia del grafo, il numero elevato di cicli e il maggior numero di vertici rendono la convergenza più lenta, fino ad arrivare, in questa circostanza, a circa 240 generazioni.

[2]Hardware su cui sono stati condotti gli esperimenti: chip Apple M3 Pro, CPU 12 core

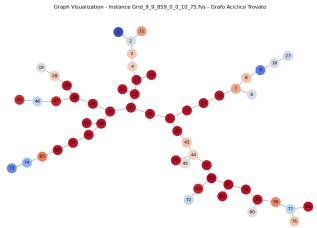
I grafici mostrano l'andamento del valore di best_fitness, evidenziandone il miglioramento progressivo. Le immagini dei grafi finali rappresentano i grafi aciclici indotti dalle soluzioni ottenute.



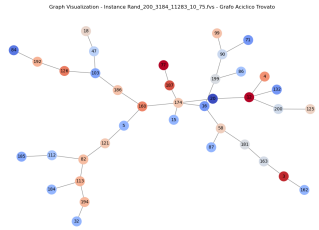
(a) Grafico 1 - Grid_9_9



(b) Grafico 2 - Rand_200_3184



(c) Grafo 1 - Grid_9_9



(d) Grafo 2 - Rand_200_3184

Figura 2: Andamento del valore di best_fitness e corrispettivi grafi aciclici finali

6 Conclusioni

Possiamo affermare che l'Hybrid Genetic Algorithm (HGA) sviluppato risolve efficacemente il problema *Weighted Feedback Vertex Set*. L'algoritmo ha mostrato un'ottima capacità di adattamento tra le diverse istanze, ottenendo soluzioni di qualità in poco tempo.

I principali punti di forza dell'approccio sono:

- **Equilibrio tra esplorazione e sfruttamento:** l'algoritmo esplora ampie aree dello spazio delle soluzioni e affina le soluzioni promettenti, garantendo una progressiva convergenza verso soluzioni di qualità.
- **Robustezza e versatilità:** l'algoritmo è in grado di adattarsi a diverse tipologie di grafi, dimostrando di poter gestire anche istanze complesse.
- **Efficienza computazionale:** l'implementazione del meccanismo di caching dei valori di fitness ha permesso una forte riduzione del tempo di esecuzione medio.

Tra i possibili sviluppi futuri si potrebbero considerare:

- Miglioraremenți della strategia di ricerca locale per avere una migliore esplorazione dello spazio delle soluzioni.
- Studiare nuove possibili implementazioni degli operatori genetici, possibilmente con parametri adattati dinamicamente.

In sintesi, l'algoritmo sviluppato si è dimostrato efficace e rappresenta una soluzione praticabile per problemi come il *Weighted Feedback Vertex Set*.

7 Bibliografia

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[2] P. Festa and P. Pardalos, "Feedback set problems," *Encyclopedia of Optimization*, vol. 2, 06 1999.

[3] G. Folino, "Algoritmi evolutivi e programmazione genetica: strategie di progettazione e parallelizzazione," Tech. Rep. RT-ICAR-CS-03-17, Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche, 2003.