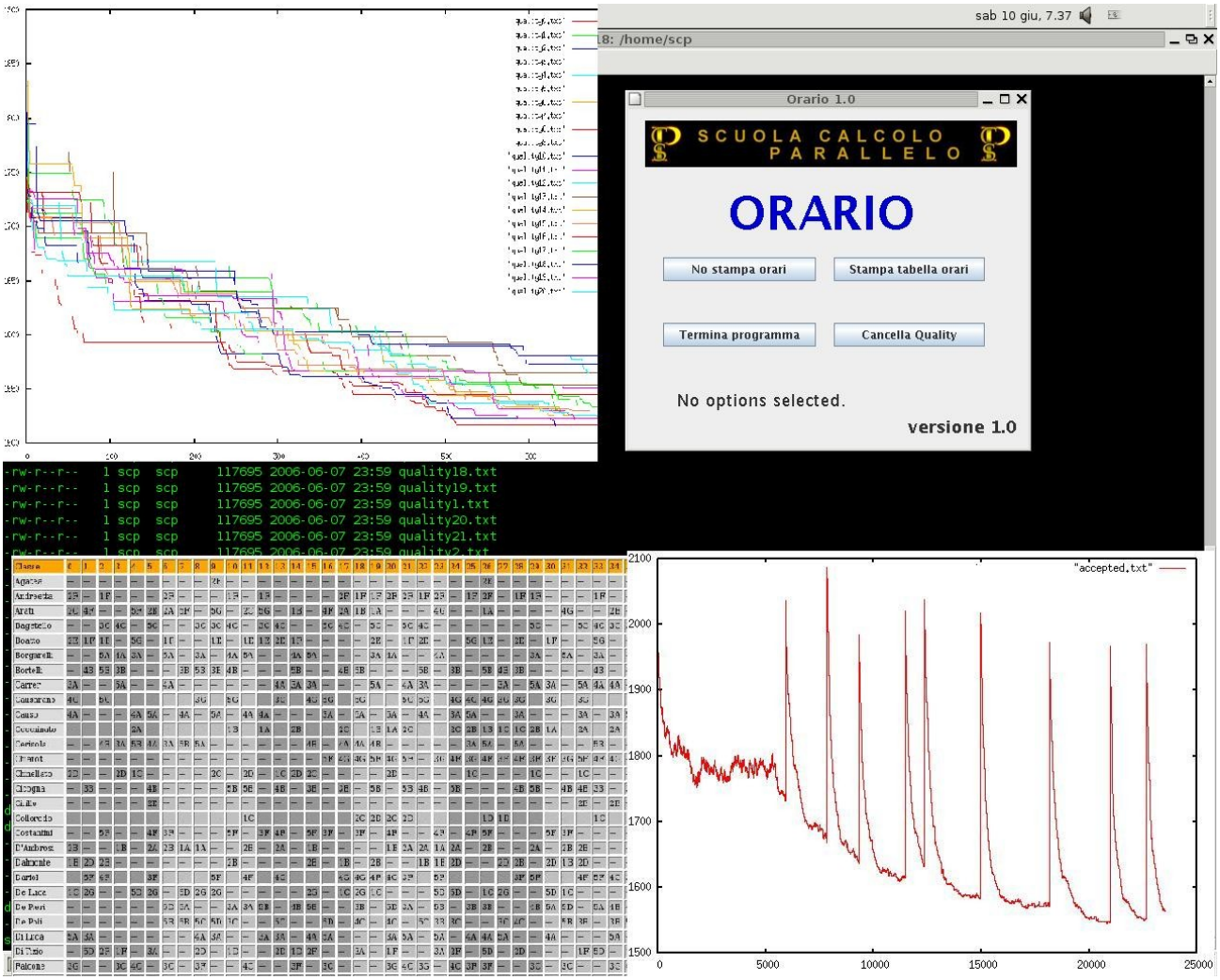
Baccega Andrea

Risoluzione di un problema NP-Completo attraverso una struttura parallela e algoritmi non deterministici.



Indice

1. Introduzione alla computazionalità

Ci sono vari modi per analizzare il problema:

- II costo computazionale di un dato algoritmo può essere individuato contando, per ogni possibile istanza dell'input dato dall'utente, le operazioni elementari (sui bit) che l'elaboratore impiega per rispondere correttamente con l'output richiesto.
- la complessità computazionale, invece, è il costo dell'algoritmo ad-hoc che risolve il problema dato.

E' nata così l'esigenza di analizzare i vari algoritmi secondo tre criteri:

1. Metodo del caso peggiore;
2. Metodo del caso migliore;
3. Metodo del caso medio.

Prima di entrare nel dettaglio dei tre metodi è bene osservare che predire il tempo di esecuzione senza conoscere nei dettagli l'input (o istanza) dato, non è un grave errore, sarebbe una buona idea, infatti, misurarlo in funzione della dimensione dell'istanza stessa.
Eppure, potrebbe accadere che due istanze di stessa dimensione implichino un tempo di esecuzione completamente diverso.

Supponiamo infatti di avere un elenco telefonico in formato digitale.Questo è posto in ordine lessicografico, e noi implementeremo due funzioni di ricerca:

- una che, data come istanza il nome, mi ritorni il numero telefonico ad esso associato;
- una che, data come istanza il numero telefonico, mi ritorni il nome ad esso associato.

Tuttavia si può notare come le due ricerche siano profondamente diverse: la prima infatti, opera su un insieme di dati già precedentemente ordinato, quindi è possibile in qualche modo sfruttare questa situazione per costruire un algoritmo efficiente, la seconda, invece, opera su un insieme non ordinato e per questo motivo si dovrà scandire tutto l'elenco prima di poter affermare che il numero richiesto non esiste.

Propongo qui di seguito le due differenti ricerche in pseudo codice:

```
algoritmo ricercanumero ( lista L , numero x) : nome {
        per ogni elemento y di L do
            if ( y = x) then return y
        return nontrovato;
}
```

```
algoritmo ricercanome ( lista L , nome x) : numero {
   sup=lunghezza di L;
   inf = 1;
   while (x ≠ L[(sup + inf)/2] ){
     m=(sup+inf)/2;
     if (x > L[m])
        inf=m+1;
     else
        sup=m-1;
     if (inf > sup) return non trovato;
   }
   return L[(sup-inf)/2];
```

Come si può ben notare l'algoritmo 'ricercanumero' è scritto in molte meno righe di pseudocodice rispetto all'algoritmo 'ricercanome'.

Per dare un'approssimazione di costo computazionale di questi due algoritmi contiamo il numero di confronti che vengono eseguiti. Ad esempio supponiamo che la nostra lista sia composta come segue:

	Nome	Numero
1	Andrea	055555
2	Antonio	055556
3	Daniela	055552
4	Daniele	004333
5	Massimo	056432
6	Matteo	056421
7	Veronica	056546

Come si può notare la lista è ordinata in ordine alfabetico, mentre i numeri telefonici non seguono alcun tipo di ordine in quanto fanno riferimento al nome.

Lanciamo ora l'algoritmo 'ricercanumero' con 060000 come numero da ricercare nell'elenco: partendo dall'inizio o anche dalla fine dell'elenco il nostro algoritmo dovrà scorrere tutte le righe presenti per poter affermare con sicurezza che il numero ricercato non esiste.
Facciamo la stessa identica cosa con l'algoritmo 'ricercanome' proponendo , anche in questo caso , all'algoritmo un nome inesistente alla lista, ad esempio 'Luca' :
Al passo 1, prende l'elemento (inf+sup)/2=(7+1)/2 = 4 quindi { "Daniele", 004333 } Confronta "Daniele" con "Luca" e si accorge immediatamente che Luca è maggiore di Daniele in ordine lessicografico, quindi l'estremo inferiore diventa 4= 3+1, è intuitivo infatti che è impossibile trovare Luca nelle righe precedenti in quanto sono ordinate.
Al passo 2, esamina l'elemento (7+4)/2=5 quindi { "Massimo", 056432 } Fa lo stesso procedimento di confronto e questa volta l'estremo superiore diventa 4=5-1
Al passo 3, esamina l'elemento (4+4)/2=4 quindi di nuovo {"Daniele", 004333} fa il confronto uguale al primo passo e incrementa l'estremo inferiore diventando cosi 5 permettendo all'algoritmo di terminare con l'istruzione if (inf > sup) return non trovato;

Riepilogando, l'algoritmo 'ricercanumero' ha finito l'esecuzione in 7 passi perché doveva esaminare tutte le righe della tabella, mentre l'algoritmo 'ricercanome' in 3 passi è terminato correttamente.
Tuttavia se provassimo a ricercare la prima riga ovvero { "Andrea", 055555 }, l'algoritmo ricercanumero terminerebbe la sua esecuzione dopo un solo passo, mentre 'ricercanumero' impiegherebbe 3 passi.

Ancora { "Daniele", 004333 } ,

- 'ricercanumero' : 4 passi;
- 'ricercanome' : 1 passo;

Come fare per determinare quindi l'algoritmo più veloce?
Come detto sopra esistono tre metodologie per analizzare la velocità di un algoritmo:

1. Analisi del caso peggiore (worst case) : consiste nel contare quante operazioni esegue l'algoritmo nel suo caso peggiore, ovvero per l'istanza o l'input che in ingresso genera più lavoro per l'algoritmo;
2. Analisi del caso medio (average case): consiste nel contare il tempo impiegato su istanze tipiche, assumendo che ogni input abbia una probabilità di verificarsi, calcoliamo una media pesata;
3. Analisi del caso migliore (best case): consiste nel contare le operazioni sull'istanza che genera meno lavoro per l'algoritmo.

Il metodo più utilizzato nell'analisi di un algoritmo è il worst case, gli altri metodi sono stati scartati per i seguenti motivi :

- Best case: Non dice assolutamente nulla sull'efficienza di un algoritmo ed è poco utilizzata per il semplice fatto che nella vita reale raramente gli algoritmi vengono interrogati solo con best

case;

- Caso medio: E' un buon modo per descrivere un algoritmo, tuttavia è difficile se non impossibile calcolare la probabilità del presentarsi di un'istanza .

Ogni algoritmo quindi viene analizzato con la 'worst case' e per indicare il suo costo computazionale si utilizza

O (f(n))

Dove n è la grandezza dell'input ed f è la funzione con cui il tempo d'esecuzione cambia al variare della grandezza dell'input.

Ricercanome	Ricercanumero
$O(\log_2 n)$	**O(n)**

Ricarcanome infatti, ad ogni iterazione del ciclo while elimina metà del sottoinsieme su cui poi proseguirà la ricerca.Ricercanumero invece, nel caso peggiore, deve scorrere comunque tutta la lista.

Sono state create, quindi, delle classi che dividono gli algoritmi in base alla loro complessità:

- **P**: la classe P è formata da tutti quei problemi che sono risolvibili con costo computazionale uguale ad $O(n^k)$ dove n è la grandezza dell'input e k una qualsiasi costante;
- **NP**: questa classe è formata da quei problemi che, data una possibile soluzione essa è verificabile in tempo polinomiale;
- **NPC**: la classe NP-Completo contiene tutti quei problemi che sono contenuti nella classe NP ma non possono essere dimostrati in tempo polinomiale. Definiamo questi problemi difficili da risolvere; tuttavia non è dimostrato che ogni problema NPC non sia risolvibile in modo efficiente. Qualora qualcuno riuscisse a risolverne uno in tempo polinomiale, allora tutti i problemi NPC hanno un possibile algoritmo polinomiale. Molti informatici ritengono i problemi NPC intrattabili perchè, considerando l'ampia gamma di quelli esaminati, nessuno è riuscito a darne una soluzione polinomiale. Sarebbe quindi davvero sbalorditivo poterli risolvere tutti in tempo polinomiale.

La classi che noi tratteremo in questo documento sono la classe NP e la classe NPC.

Un problema famoso che rientra nella NPC è il TSP (Travelling Salesmen Problem), o meglio conosciuto come problema del commesso viaggiatore, dove il commesso deve servire i clienti appartenenti a differenti città toccandole tutte (se possibile) spendendo la minor quantità di benzina possibile.

2. Introduzione al problema

Ci si è chiesti come si potesse creare un programma in grado di poter elaborare e restituire come output un orario scolastico che, oltre a rispettare dei vincoli obbligatori, ottimizzasse, ad esempio, quelle che potrebbero essere delle preferenze espresse dai professori.

In sé il problema esclude la possibilità di poter adottare una tecnica del tipo bruteforce, ossia tentare tutte le possibili combinazioni di orario fino ad ottenere così il migliore.
E' infatti facilmente intuibile che l'elaborazione di così grandi quantità di dati non sono nemmeno avvicinabili alle possibilità che la tecnologia ci offre oggigiorno.

Ci siamo quindi orientati verso un approccio di tipo non deterministico, optando per l' utilizzo dei metodi di Montecarlo.
Grazie all'aiuto di un generatore di numeri pseudocasuali, cerchiamo di esplorare nella maniera più uniforme possibile l'universo delle soluzioni; tuttavia questo non garantisce nè di trovare la soluzione ottimale, nè di esplorare l'intero universo.

Lo spazio delle soluzioni in questo caso è enorme… Se lo volessimo esplorare in modo deterministico dovremmo applicare un algoritmo che, provando tutte le combinazioni, in esecuzione utilizzi

- $\text{Oresettimanali}^{\text{Nprofessori}}$ cicli di clock.

Proviamo subito a vedere quanto efficiente sarebbe utilizzare un algoritmo di questo tipo.
Oresettimanali = 2*36
Professori = 5
Cicli utilizzati = 1296^5 = 3656158440062976
Anche potendo tagliare molte soluzioni il numero resterebbe alto per i processori attuali. Supponendo ottimisticamente che un processore a 2000Mhz riesca a elaborare un' istruzione ogni ciclo di clock e 1296^5 sia il numero di istruzioni allora:

3566158440062976/2000000000 = 1828079 secondi

Servirebbero quindi all'incirca 3 ore per conoscere la configurazione ottimale piazzando 5 professori in 2 classi con 36 ore settimanali ciascuna.

3. Cos'è la programmazione parallela

La programmazione parallela è una delle metodologie oggi più utilizzate per ottenere sistemi di calcolo estremamente efficienti.
Il motivo per il quale si sono sviluppate tecniche di calcolo parallelo è la possibilità di ottenere un notevole incremento della velocità elaborativa di un problema semplicemente scomponendo il dominio delle soluzioni tra varie unità di calcolo per poi ricomporlo una volta risolti tutti i "sottoproblemi".

All'inizio si creavano supercalcolatori con numerosi processori già integrati, ora invece si opta per un'altra soluzione altrettanto prestante che è creare un gruppo di normalissimi pc, anche con processori mediocri, collegati tramite una rete veloce .
Quest'ultima configurazione di cluster è molto più vantaggiosa della prima in quanto si ha:

a. **Economicità**: Essi sono da 5 a 10 volte piu economici rispetto ai supercomputer Commerciali.
b. **Scalabilità**: In caso di bisogno, integrare un nuovo personal computer nella rete è una cosa estremamente facile, in quanto si lavora con un' architettura estremamente flessibile.
c. **Piattaforma parallela standard**: E' possibile utilizzare gli stessi programmi che vengono usati sia sui supercomputer che sulle reti di computer, senza alcuna modifica in quanto vengono utilizzate le stesse librerie.
d. **Software open Source**: La maggior parte del software utilizzato nei cluster odierni è liberamente disponibile ed estremamente testato e affidabile oltre ad essere aggiornato continuamente.

Nella programmazione sequenziale si risolve il problema tramite un algoritmo le cui istruzioni saranno appunto eseguite in sequenza, mentre nella programmazione parallela le istruzioni vengono eseguite in parallelo, tuttavia l'algoritmo deve sfruttare in maniera efficiente il parallelismo esplicitatabile del problema.

Esiste una metodologia generale per esplicitare il parallelismo:

a- Si suddivide il problema tra i vari processori (domain decomposition)
b- I vari processori lavorano sul proprio sottoproblema e comunicano tra loro solo nel caso di cooperazione.
c- Si bilancia il carico del lavoro e si riducono le comunicazioni allo stretto necessario.

Negli ultimi anni abbiamo visto un incremento abbastanza massiccio degli Hz di clock per ogni singola cpu passando da poche decine di Hz a milioni.
Questo, insieme ad altre migliorie avvenute questi anni, hanno dato vita a dei colli di bottiglia riguardanti le comunicazioni tra processori e la velocità della memoria volatile (RAM) .
Infatti queste due tecnologie non sono migliorate in modo proporzionale dal punto di vista delle prestazioni.

Grazie all'incremento di prestazione delle cpu, è stato possibile utilizzare internet come una grande rete di calcolo parallelo per alcune applicazioni che non richiedevano poi così tante sincronizzazioni.

Alcuni esempi sono:

- Seti@tHome : Il quale analizzava i segnali elettromagnetici provenienti dallo spazio con la speranza di trovare forme intelligenti.

- La fattorizzazioni di interi molto grandi, questo ha applicazioni notevoli nella crittografia.
- Grid Computing, ovvero la realizzazione di un software per la gestione delle risorse distribuite.

Un programma sequenziale può essere suddiviso in due tipi di sezioni :

a- Parti di codice non parallelizzabili , ovvero sezioni di codice che sono interamente sequenziali
b- Sezioni potenzialmente parallelizzabili.

Il rapporto tra queste due sezioni è molto importante per quanto riguarda le prestazioni raggiungibili . Supponiamo di avere :

a- Parti non parallelizzabili 50%
b- Parti potenzialmente parallelizzabili 50%

Il tempo di esecuzione del nostro programma dipenderebbe per metà da ognuna di queste due sezioni, a questo punto, anche se noi riuscissimo a rendere 0 il tempo di esecuzione delle parti potenzialmente parallelizzabili , avremmo ottenuto un tempo di esecuzione dimezzato e quindi due volte più veloce del programma originale sequenziale.

E' evidente quindi che molto dipende anche da questo rapporto.
E' molto importante , inoltre , che i metodi di suddivisione del problema vengano scelti con assoluta cura perche il risultato dipende strettamente da questa decisione.

Per ottenere una buona parallelizzazione bisogna:

a- Mantenere tutti i processori equamente occupati (load balancing).
b- Minimizzare le comunicazioni inter-processors.
c- Minimizzare i tempi nei quali ogni processore rimane in attesa di un Input o di una sincronizzazione (idle time). Per ovviare a questi problemi si potrebbe operare
    a. Eliminando le sincronizzazioni.
    b. Destinando ad ogni processore più problemi; così facendo, ogni qualvolta un processore deve aspettare, continua nel frattempo l'esecuzione dell'altro processo.

Si può notare come queste migliorie siano strettamente collegate fra loro, ad esempio un load balancing adeguato elimina moltissimo gli idle time.

Attualmente la programmazione parallela ha moltissime applicazioni nella realtà e soprattutto nel mondo scientifico , infatti essa viene utilizzata per effettuare studi su:

- Chimica
- Biologia
- Fisica delle alte energie
- Meteorologia
- Astrofisica
- Fluidodinamica
- Scienza dei materiali
- Economia.

4. Spiegazione del problema

Il problema richiede di trovare l'orario ottimo a partire da un input che fornirà informazioni riguardo:

- Il Numero di professori;
- Il Numero di classi;
- Il Numero di ore per ogni professore i in ogni classe j;
- Le compresenze per la gestione di laboratori;
- Preferenze dei professori quali ad esempio:
  - Primo e secondo giorno libero preferito;
  - Eventuali ore impossibili (per la gestione di professori che lavorino in altri istituti);
  - Eventuali ore preferite;
  - Eventuali ore non preferite;

Per la costruzione dell'orario ottimale abbiamo bisogno di seguire le informazioni preferenziali ma anche di dati obbligatori che se non vengono rispettati rendono l'orario inaccettabile, suddividiamo quindi I vincoli in due gruppi :

1- **Obbligatori** : Sono tutti quei vincoli per i quali la configurazione assume uno stato inaccettabile nella realtà, ad esempio non è ammissibile avere lo stesso professore in due posti diversi nello stesso momento ;

2- **Facoltativi** : Sono tutti quei vincoli che, anche se non rispettati, non fanno perdere significato alla configurazione, ma ne resta danneggiato solamente il suo valore.

Il problema può essere classificato come uno dei tanti di ottimizzazione. Data una configurazione dei possibili output e una funzione di peso, dobbiamo ricercare la corretta combinazione dei dati che minimizzi o massimizzi il valore della funzione.

In questo caso la tabella dei dati è l'intero orario, dove saranno suddivisi opportunamente X professori in K*N unita orarie, dove K è il numero di ore settimanali e N è il numero di classi totali.

Come ho dimostrato nell'introduzione al problema, esso ha un dominio di possibili configurazioni estremamente grande e praticamente impossibile da "digerire" per un computer, quindi pur tenendo conto dei possibili vincoli obbligatori e applicando degli opportuni tagli euristici non andando nemmeno a tentare di esaminare una soluzione che viola un vincolo obbligatorio, la complessità del problema resta comunque enorme.

Ci siamo quindi orientati verso la programmazione parallela , che permette di scomporre il problema
In L parti dove L è il numero di calcolatori dedicati alla risoluzione concorrente.

Tuttavia ,ci siamo accorti che lo spazio delle soluzioni rimaneva enorme, tale da renderci impossibile la risoluzione del problema anche tentando la via della parallelizzazione.

Ci siamo orientati, quindi, verso un metodo di approssimazione non deterministico, precisamente abbiamo optato per uno dei metodi di Montecarlo, il Simulated Annealing.

5. L'euristica e il Simulated Annealing

Messi di fronte ad un problema, esistono tre tipi di approccio per la sua risoluzione attraverso un algoritmo, ovvero:

- **La programmazione dinamica**, secondo la quale alcuni problemi di grandezza N sono risolvibili in modo efficiente avendo la soluzione dello stesso problema ma di grandezza inferiore. Una specie di ridefinizione del problema in base allo stesso piu piccolo, ad esempio : $F(N) := \sum_{i}^{N} F(N-1) * F(N-i)$

- **La programmazione greedy (euristica)**, secondo questa tecnica algoritmica possiamo risolvere moltissimi problemi e in costo computazionale anche molto basso , tuttavia questo tipo di approccio di risoluzione non garantisce in molti problemi di trovare la soluzione ottimale , ma di darne un'approssimazione vicina.

- **La programmazione esaustiva**, secondo quest'ultima per risolvere il problema tentiamo tutte le possibili strade fino a trovare l'ottimale.

E' utile precisare che la programmazione dinamica e l' euristica sono molto simili, tuttavia esse si differenziano per la metodologia; la prima infatti opera in modalità bottom-up, ovvero per risolvere il problema si parte dalla fine per arrivare all'inizio, mentre nella seconda si opera in top-down, ovvero dall'inizio alla fine.

Non potendo definire il nostro problema in funzione dello stesso con grandezza inferiore, siamo caduti nell'euristica, accontentandoci così di un'approssimazione della soluzione ottimale.

Il simulated annealing è un euristica ampiamente discussa.Alcuni autori come Johnson A, forniscono delle descrizioni eccellenti di questo algoritmo paragonandolo al fenomeno fisico formazione del reticolo cristallino . Il simulated annealing è nato proprio come metodo di simulazione di questo fenomeno.

L’annealing è il processo con il quale un materiale solido , portato allo stato fluido mediante il riscaldamento di alte temperature, viene riportato poi di nuovo allo stato solido, controllando e riducendo gradualmente la temperatura.

Ad alte temperatura la struttura cristallina del solido non esiste, infatti gli atomi nel sistema si trovano in uno stato altamente disordinato e quindi l’energia totale è piuttosto elevata.

Per portare gli atomi in una configurazione cristallina ottimale, deve essere abbassata la temperatura, tuttavia riduzioni troppo brusche di questa possono causare problemi nel reticolo cristallino con conseguente instabilita del sistema chiamato stress termico.
L’annealing evita lo stress termico procedendo ad un graduale raffreddamento del sistema portandolo ad una struttura globalmente ottimale.

Il sistema si dice essere in equilibrio termico alla temperatura T se la probabilita P(Ei) di uno stato con energia Ei è governata dalla distribuzione di Boltzmann. Questo tipo di distribuzione scoperta appunto da Boltzmann gestisce l’energia di un sistema in equilibrio termico.

Ad alte temperatura tutti gli stati di energia sono probabilmente possibili, mentre a basse temperature il sistema si trova sicuramente in stati di energia piccolissima.
Nel 1953 venne sviluppato un algoritmo per simulare il comportamento di un insieme di atomi in equilibrio termico ad una particolare temperatura, questo algoritmo ha un ruolo realmente fondamentale per l’applicazione del metodo a problemi di ottimizzazione.

Questo algoritmo genera un insieme di configurazioni ad ogni temperatura T con la proprietà che la Energia delle diverse configurazioni generate possono essere rappresentate dalla distribuzione di Boltzmann.

Il metodo comincia da una assegnata configurazione iniziale degli atomi in un sistema con Energia E(0), vengono generate quindi molte configurazioni con piccole perturbazioni della configurazione corrente. Viene poi deciso se accettare o meno la configurazione in base alla differenza tra energia della configurazione corrente e quella della nuova configurazione.Questa decisione dipende dal fatto che le energie delle configurazioni accettate devono seguire la distribuzione di Boltzmann.

Quindi per decidere se accettare o meno una configurazione candidata si opera la seguente decisione:

- Se Energia(configurazione candidata) < Energia(configurazione corrente)
  Accetta la configurazione candidata;

  $$Oppure\ se:\ e^{\frac{-\Delta E}{(kb * T)}} < Random(0;1)$$

  $$Accetta\ comunque\ la\ configurazione\ candidata;$$

Praticamente se l'energia della configurazione candidata è minore di quella attuale allora accettiamo la nuova configurazione candidata, altrimenti la accettiamo con probabilità:

$$e^{\frac{-\Delta E}{(kb * T)}}$$

Dove :

- T= Temperatura attuale
- ΔE= Energia della configurazione candidata – Energia delle configurazione corrente
- Kb= Costante di Boltzmann

All'inizio l'algoritmo ha una maggiore probabilità di accettare anche pessime configurazioni in quanto la temperatura iniziale è alta, successivamente man mano che la temperatura cala l'algoritmo viene confinato in regioni sempre più ristrette accettando molto più selettivamente configurazioni pessime.
Da quanto detto si riesce ad intuire che ci sono un certo numero di parametri che l'analista deve decidere, consentendo al metodo di adattarsi a molti problemi essendo elastico, tuttavia la taratura di questi parametri implica un grande lavoro iniziale per consentire al metodo di poter convergere.

Un altro grande vantaggio che ci offre il simulated annealing è che lo si può applicare anche a problemi di ottimizzazioni per i quali non si ha una buona conoscenza.

L'algoritmo SA può essere riassunto nel seguente generico pseudo codice:

a. Sia data una configurazione iniziale o soluzione x0 con valore della funzione di qualita Q, e selezionare un valore iniziale T0 per la temperatura.
b. Per ogni stadio della temperatura effettuare i seguenti step:
   1. Generare una configurazione candidata **ammissibile** tramite una piccola perturbazione casuale della configurazione corrente e valutare ΔE = Q(soluzione candidata) – Q(configurazione corrente);
   2. Se ΔE < 0, allora la configurazione candidata ha un valore di qualita migliore della precedente , quindi la accettiamo come nuova configurazione corrente.

      $$Altrimenti\ la\ accettiamo\ con\ probabilità:\ e^{\frac{-\Delta E}{(kb * T)}}$$

3. Se non si è raggiunto l’equilibrio termico torna a b altrimenti vai a c.

c. Se il processo di annealing è incompleto torniamo allo step b.

Uno dei parametri più importanti per la convergenza del **sa** è la scelta della temperatura e dello schema che deve seguire per diminuire.
La temperatura deve essere diminuita solo quando nel ciclo lo step 3 va a c, ovvero quando le configurazioni hanno raggiunto l’equilibrio termico, tuttavia generare troppe soluzioni candidate senza trovare l’equilibro termico porta ad un aumento del tempo di calcolo.A tal fine è necessario tarare un altro parametro detto di *transizione*.

Il numero di transizioni deve essere tale da assicurare che vengano generate un numero di configurazioni candidate tale da garantire di passare da uno stadio termico all’altro.
Un altro problema riguardante lo schema di variazione della temperatura è quanto questa debba essere modificata.

Ridurre troppo velocemente la temperatura, infatti, come già detto, potrebbe dare vita ad un sistema instabile, inoltre questo potrebbe rimanere confinato in un minimo locale.
Ridurre troppo lentamente, invece, aumenterà di molto il tempo di calcolo.

E’ compito quindi dell’analista trovare un compromesso a questi due estremi in modo da rendere il metodo convergente per il problema dato.
Esistono inoltre due metodologie di riduzione della temperatura :

a. **Geometrico** : Consiste nel moltiplicare la temperatura per una costante compresa tra 0 e 1 chiamata cooling ratio che spesso è quella piu utilizzata.
b. **Logaritmico** : Questo schema di raffreddamento è piu complesso e usato solo in particolari casi.

6. Heap.h

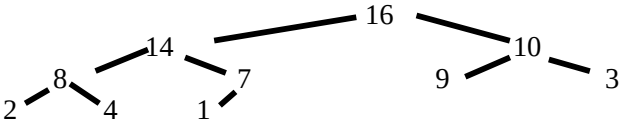
Heap.h è un header file richiamato poi da orario.h.
Esso al suo interno implementa attraverso l'heap una particolare struttura dati chiamata "Priority Queue".
Essa permette di inserire al suo interno elementi con una certa priorità, e quando richiediamo alla struttura dati un elemento, quest'ultima ci ritornerà quello con priorità maggiore (o minore).

Perchè Heap: Concettualmente può essere visto come un albero binario quasi completo, dove ogni nodo contiene un elemento nel quale è presente la sua priorità .
Inoltre ogni nodo deve soddisfare la seguente proprietà:

Priorita(figlio) <= Priorita(Padre)

Ovviamente nella radice dell'albero avremo l'elemento con priorità maggiore.
Tuttavia è possibile realizzare questa struttura dati in modo molto più efficiente e semplice dal punto di vista informatico pensando l'albero come un array di lunghezza "Nnodi".

Ecco un esempio di heap rappresentato come un albero :



Può essere espresso in un array nel modo seguente.

Indice	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Heap	16	14	10	8	7	9	3	2	4	1

Se notiamo la radice dell'albero è stata messa nella posizione 0 dell'array.
Con l'array ci si potrebbe porre il problema di come trovare il figlio sinistro e il figlio destro .

Dx (i) = 2* i +1;
Sx (i) = 2* (i+1);
Padre (i) = (i-1)/2;

Quindi la stessa struttura dati riscritta in un array , ci permette notevoli vantaggi:

- Meno spazio usato in memoria (evito i puntatori a padre , figlio sinistro e figlio destro);
- Più velocita per quanto riguarda lo scorrere degli elementi nell'heap;
- Molto più facile, e veloce, da scrivere sotto forma di codice.

L'heap inoltre deve implementare le seguenti operazioni:

- Push : Serve per inserire un elemento all'interno dell'Heap , questa operazione potrebbe modificare la posizione di alcuni elementi , ma al massimo potrebbe modificare tanti elementi quanta è l'altezza dell'albero. Ovvero log2n;
- Pop : Questa funzione serve per la restituzione dell'elemento con priorità maggiore nella coda, come per la push il tempo di esecuzione è O(lg2n).

7. Stack.h

Stack è un header file richiamato poi da orario.c.

Al suo interno è codificata la struttura dati a pila semplice di tipo Last In First Out (l'ultimo ad entrare è il primo ad uscire).
Lo stack ci serve per poter implementare una procedura di backtrack sulle permutazioni dell'orario: quando generiamo un orario nuovo e per qualche motivo questo non è accettato dall'euristica SA, ci siamo posti il problema di come poter tornare all'orario precedente.

Inizialmente abbiamo optato per la creazione di volta in volta di due tabelle differenti facendo delle copie delle stesse qualora ne avessimo avuto bisogno; tuttavia ci siamo accorti che questo tipo di soluzione non era affatto efficiente.
Lo stack ci permette di ripercorrere nel senso inverso gli scambi che effettuiamo, evitandoci così della ridondanza nella memoria.

Questa struttura dati implementa le seguenti operazioni:

- Pop: La quale restituisce l'ultimo elemento entrato. Nel caso in cui la struttura dati non contenga alcun dato, allora questa funzione restituisce l'elemento speciale NULL.
- Push: La quale inserisce nella struttura dati un elemento.

Queste due operazioni hanno una complessità algoritmica di O(1), ovvero tempo pressoché costante.
La nostra implementazione dello stack avviene nel seguente modo:

- Creiamo un array di *n* elementi, il numero n viene deciso dall'utente.
- Ogni qualvolta si inserisce un elemento aumentiamo l'indice del numero di elementi inseriti(ninseriti), finchè *ninseriti* non sara uguale al numero n deciso all'inizio dall'utente.
- Quando viene chiamata la pop, decrementiamo di uno *ninseriti* e restituiamo l'elemento dell'array puntato da *ninseriti*.

0	***1***	***2***	***3***	***4***	**Ninseriti=0**
NULL	NULL	NULL	NULL	NULL	**n=5**

Ecco una rappresentazione di uno stack di 5 elementi appena creato.

Applichiamo ora una serie di operazioni per chiarire come funziona uno stack:

1- push(“Ciao”);

0	***1***	***2***	***3***	***4***	**Ninseriti=1**
Ciao	NULL	NULL	NULL	NULL	**n=5**

2- push(“Mamma”);

0	***1***	***2***	***3***	***4***	**Ninseriti=2**
Ciao	Mamma	NULL	NULL	NULL	**n=5**

3- pop();

0	***1***	***2***	***3***	***4***	**Ninseriti=1**
Ciao	NULL	NULL	NULL	NULL	**N=5**

8. Orario.h

Orario.h è una libreria che si occupa della costruzione di tutte le strutture dati necessarie al programma vero e proprio, inoltre questo genera una possibile configurazione dei dati rispettando i vincoli obbligatori.

Questo file è composto principalmente da 4 parti :
1. La struttura dati;
2. La lettura dei dati;
3. La creazione del primo orario;
4. Stampa della struttura dati.

La struttura dati:

Abbiamo già discusso di strutture dati quando abbiamo parlato dell'heap e dello stack, tuttavia non sono le uniche che abbiamo implementato nel nostro programma.
Qui di seguito riporto tutte le strutture che vengono utilizzate da orario.h per la creazione del primo orario:

```
struct _classe_prof {
    uint8_t id_prof;
    uint8_t ore_rim;
    int laboratorio;
    uint8_t id;
    uint8_t id_materia;
};
```

Questa struttura serve per avere informazioni, data una classe e un professore, riguardo alle ore rimanenti che il professore deve svolgere nella classe , all'id della materia che deve insegnare, e se per caso è in laboratorio.

```
struct _prof {
    uint32_t ore_rim;
    uint32_t num_materie;
    uint32_t teorico;
    uint32_t num_classi;
    uint32_t num_classi_rim;
    struct _classe_prof *classi;
};
```

Quest'altra serve per avere informazioni solamente su i professori, fornendo dati riguardo alle ore totali rimanenti da svolgere nelle varie classi, il numero di classi dove lui insegna e indica inoltre se il docente in questione è solamente teorico.

```
struct _info_classe {
    uint32_t ore;
    uint32_t prof;
    int laboratorio;
};
```

Questa ha il compito di dare informazioni riguardo ai prof che insegnano in quella classe e quante ore hanno ancora da insegnare.

```
struct _classe{
    uint32_t orario[GIORNI*OREGIORNO];
};
```

Questa contiene in formato tabulare l'orario vero e proprio.

```
struct global_frequency {
    uint32_t classe;
    int ore_rim;
    int lab;
    int prof;
};
struct local_frequency {
    uint32_t prof;
    int ore_rim;
    int lab;
    uint32_t materia;
};
```

Global_frequency e local frequency , contengono tutte le informazioni che servono all'heap per determinare il docente con più priorità.

La lettura dei dati:

Quest'azione viene eseguita da una funzione chiamata leggiorario.
Questa funzione , carica in Ram l'intero file di configurazione ORARIO.bsb, questo contiene al suo interno le seguenti informazioni così formattate:

- Numero Materie
  - 'Numero Materie' volte il nome della iesima materia;
- Numero Professori
  - 'Numero professori' volte il nome dell'iesimo professore;
- Numero Classi
  - 'Numero classi' volte il nome dell'iesima classe;
- Numero Laboratori
  - 'Numero laboratori' volte il nome dell'iesimo laboratorio;
- Di seguito ci sono le informazioni riguardo ai professori e le ore da fare formattate da un carattere di tabulazione nel seguente modo:
  - Numero Ore;
  - Nome della materia;
  - Nome del professore;
  - Nome della classe;
  - Eventualmente il Nome del laboratorio.

Durante la lettura vengono riempite tutte le strutture dati che serviranno poi all'euristica SA , per generare gli orari.

La creazione del primo orario:

Per renderci la vita più facile e per non investire l'utilizzatore di questo problema , ci siamo chiesti come si potesse creare un orario che rispettasse tutti i vincoli obbligatori.
Per risolvere questo dilemma ci siamo rivolti verso un'euristica da me inventata.
Quest'algoritmo lavora nel seguente modo:

```
for i da 1 a Numero ore
        for j da 1 a Numero classi
                Orario[i][j]= pop(Heap[i][j]);
                if ( Professore[Orario[i][j]] == OCCUPATO AND Non ci sono altri professori disponibili)
                        first_backtrack(Professore, i, j);
```

Tuttavia quest'euristica senza first_backtrack non garantisce che un professore non si possa trovare in due classi contemporaneamente, per risolvere questo problema abbiamo creato una funzione che applica un backtrack qual'ora si stia tentando di inserire due docenti nella stessa ora in più aule.
La funzione first_backtrack opera nel seguente modo:

1. Cerca nelle classi precedentemente inserite, nella stessa ora il professore.
2. Quando lo trova, la funzione cerca di rimpiazzare il professore in questione con un altro libero.

Ecco qui un esempio di quando la funzione first_backtrack entra in atto.

Nore -1
A
C

Prof disponibili	
B B	
A	D

In questo caso quando l'euristica tenta di assegnare 'A' nella terza classe si accorge che il professore in questione è gia occupato in un altra ora e perciò applica la seguente modifica:

Nore -1
B
C
A

Prof disponibili	
A	
B	
D	

Quindi il problema viene corretto, tuttavia esiste una complicazione.Ne riporto di seguito un esempio:

Nore -1
A
B

Prof disponibili	
B	
C	
A	D

Qui si ripresenta la stessa situazione precedente con una differenza nella seconda ora infatti se applichiamo la stessa correzzione di prima ci accorgiamo che la tabella risultante sarebbe :

Nore -1
B
B
A

Prof disponibili	
A	
C D	

E' evidente che esiste un’inconsistenza dei dati,in quanto viene violata la condizione obbligatoria per la quale lo stesso professore non può coesistere in due classi nella medesima ora.
La funzione first_backtrack è intelligente, e accorgendosi di questo problema, lo risolve ricorsivamente generando così la seguente configurazione:

Nore -1
B
C
A

Prof disponibili	
A	
B	
D	

La funzione first_backtrack è identificata in orario.h con la funzione correggi.

A ulteriore conferma del funzionamento di ques'euristica, bisogna ricordarsi che l'Heap funziona in base ad una priorità che ora noi andremo a definire.

La priorità di un professore generalmente aumenta in base a:

A- Numero di ore ancora da assegnare in quella classe;
B- Numero di ore ancora da assegnare in altre classi;

Le funzioni che dati due professori restituiscono chi fra i due ha maggire priorità di essere assegnato sono:

- maxgf;
- maxlf;

Utilizziamo due funzioni perchè la prima opera in modo globale, ovvero ci permette di scegliere la classe che ha più priorità di essere scelta.
Successivamente, quando abbiamo la classe, interroghiamo l'heap con *maxlf* così avremo il professore che ha maggiore priorità di essere assegnato.

Utilizziamo quindi due Heap, uno per le classi e uno per i professori, dove la priorità per l'heap globale aumenta con le stesse modalità con le quali aumenta la priorità dei professori.

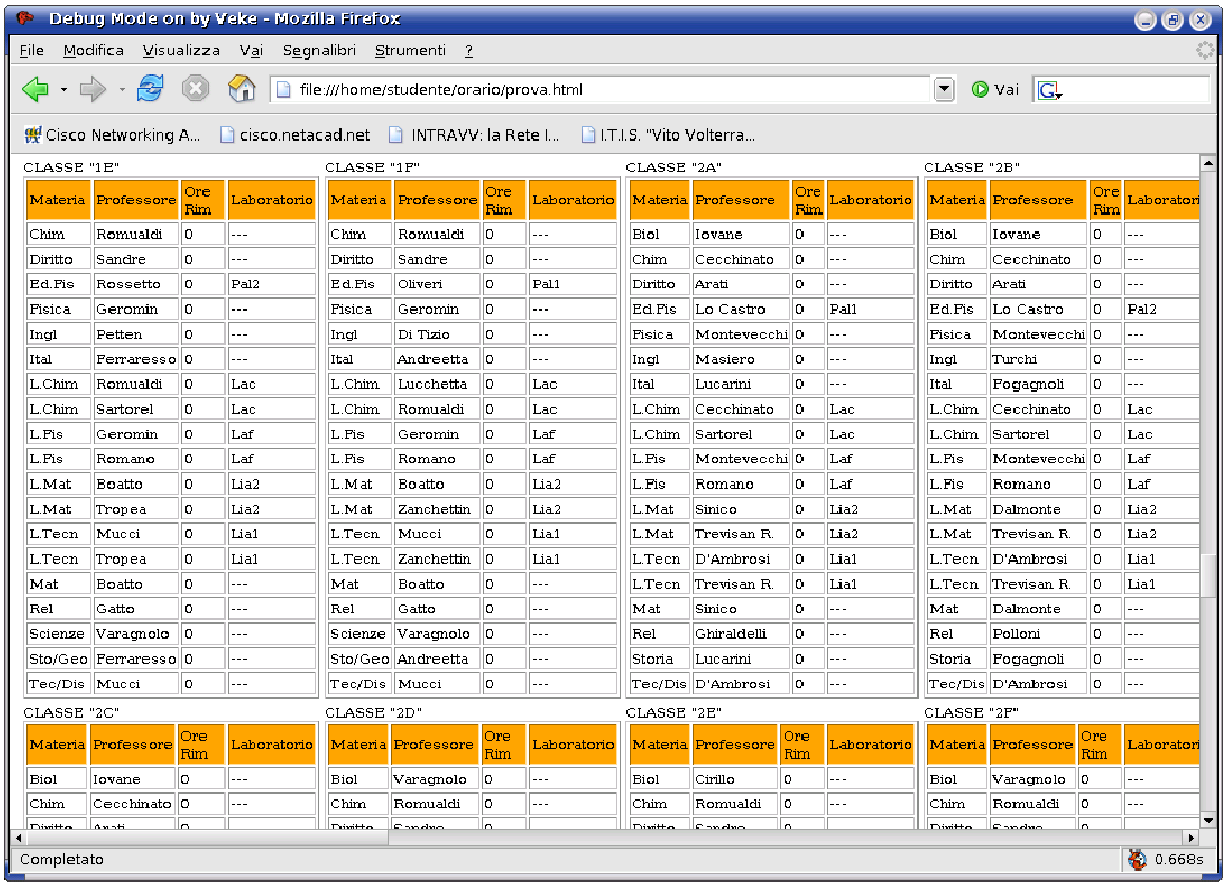
Quindi per primi verranno inseriti i professori con più ore in assoluto così facendo i docenti avanzeranno omogeneamente nella loro assegnazione, evitando che qualche professore, quando siamo all'inserimento della penultima ora, ad esempio abbia più di 2 ore disponibili.

Tuttavia l'euristica funziona in qualunque caso in cui sia è possibile generare un orario valido, altrimenti la funzione first_backtrack cade in una ricorsione infinita tentando di correggere problemi che creano altre inconsistenze.

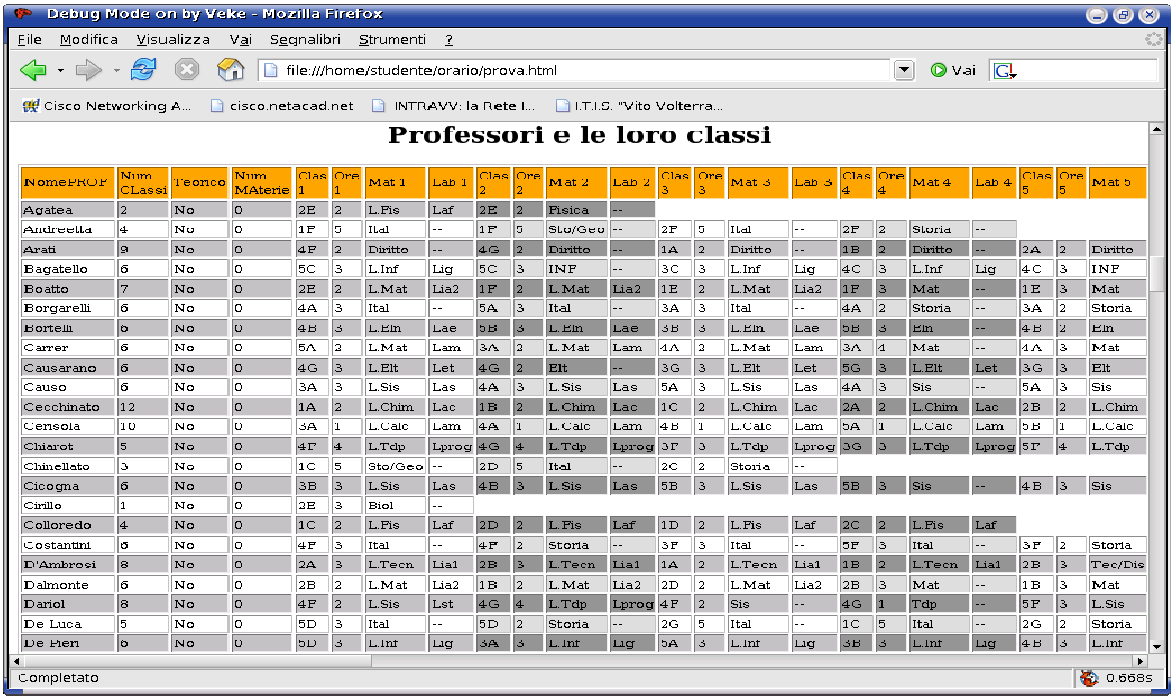
La stampa della struttura dati:
Per verificare che la struttura dati fosse stata creata correttamente, e per verificare che l'orario generato dalla nostra euristica non creasse orari inconsistenti, abbiamo creato una funzione che facesse uno snapshot dell'intera struttura dati su un file html.

Questa 'stampa' è risultata molto utile per correggere piccoli problemi dell'algoritmo.
Ecco un esempio di stampa della struttura dati:

Questa è uno screenshot di un pezzo della struttura dati , la quale riporta in riga i professori , e nelle colonne le informazioni riguardo a dove e quando deve insegnare.



Questo screenshot crea una tabella per ogni classe i professori che ci insegnano , le materie e gli eventuali laboratori.



9. Porting.h

Porting.h è un header file che serve per passare dalle strutture dati utilizzate in orario.h a quelle che poi utilizzerà orario.c .

Al suo interno sono implementate delle routine che servono solamente a trasformare le strutture dati.

Questo è risultato necessario in quanto orario.c e orario.h sono stati sviluppati separatamente e sopratutto sono stati sviluppati in modo da ottenere la massima prestazione con la struttura dati piu adatta.

Le due funzioni implementate al suo interno vengono richiamate appena dopo la creazione del primo orarioo.

10. Orario.c

Orario.c al suo interno implementa il vero e proprio programma che genera i vari orari e che attraverso l'utilizzo del simulated annealing, cerca di far convergere la struttura dati alla configurazione perfetta.

Descriverò brevemente qui di seguito le funzioni che intervengono direttamente nella convergenza della struttura dati:

La funzione loops:

La funzione loops è responsabile dei cicli principali per gestire gli scambi.
Nel nostro programma abbiamo deciso di creare due differenti cicli loops, uno che intervenisse sugli scambi di una singola classe, e uno che invece intervenisse sugli scambi di piu classi.Questi due cicli poi sono racchiusi da un ciclo esterno più grande.
Il primo effettua scambi di livello uno e non effettua modifiche molto grandi sulla struttura dati in quanto, scelta casualmente una classe, gli scambi avvengono tra due ore di due professori diversi nella stessa classe.

Il secondo, invece, necessita di un approfondimento, in quanto questo opera nelle modifiche relativamente più profonde del primo ciclo; abbiamo deciso di scrivere questo ciclo perché il primo, a nostro modo di vedere, operava solo pochi scambi, e non modificava la struttura dati in modo significativo. Per chiarire meglio questo concetto si può pensare ad una situazione di questo tipo:

Classe 1	Nane	Bepi	Mario	Matteo
Classe 2	Toni	Nane	Bepi	Mario

Supponiamo, nella precendente situazione, di operare nella Classe 2 il seguente scambio ('Nane', 'Bepi') . Questo non ci è reso possibile in quanto il professor 'Bepi' è gia occupato in quell'ora nella Classe 1.

La funzione loops che opera scambi in profondità provvede a non fermarsi a queste difficoltà ma rende possibile uno scambio nella Classe 1 per effettuare l'operazione desiderata, ottenendo così una possibile configurazione come quella riportata di seguito:

Classe 1	Nane	Mario	Bepi	Matteo
Classe 2	Toni	Bepi	Nane	Mario

A nostro modo di vedere, così, riusciamo ad esplorare la struttura dati in modo più omogeneo e completo.

Questa distinzione tra i due tipi di cicli ci ha portato a creare due costanti con le quali gestiamo il numero massimo di scambi per ogni tipo di ciclo.

1. SINGLE_VALORE_STALLO che gestisce gli scambi di profondità 1. Questo valore è relativamente grande poiché esso crea piccole perturbazioni alla struttura dati ; mantenendo questa costante alta si ha una maggiore probabilità che la tabella, in quella configurazione, venga esplorata in modo 'completo' ma superficiale.

2. MULTIPLE_VALORE_STALLO che gestisce, invece, gli scambi di profondità multipla. Questo valore viene assegnato in modo direttamente proporzionale al valore della profondità massima. Da essa dipendono modifiche molto radicali alla struttura dati e conseguentemente all'esplorazione omogenea e totale di questa.

Le funzioni di scambio e di backtrack:

Queste sono:

1. scambia(): richiamata direttamente dalla funzione loops, crea una variabile casuale che definisce la classe dove si svolgerà successivamente lo scambio effettivo.
2. tenta_scambio(): richiamata dalla funzione scambia.Come suggerisce il nome stesso, essa tenta uno scambio all'interno della classe scelta.Se essa viene richiamata precedentemente dal ciclo di profondita 1, allora, in caso di non riuscità dello scambio, essa termina, altrimenti continua ricorsivamente tentando di sanare eventuali problemi, come quello visto precedentemente.
3. effettua_scambio(): richiamata dalla funzione tenta_scambio, essa si occupa di effettuare fisicamente lo scambio nella memoria, definendo anche le variabili utili per il backtrack.
4. backtrack(): richiamata dalla funzione loops.Nel caso in cui la nuova configurazione dell'orario non venga accettata dall'euristica SA, essa effettua gli scambi appena eseguiti in senso inverso, in modo da poter riavere in modo efficiente la configurazione della tabella precedente.

La funzione di qualità:

La funzione di qualità è una tra le più importanti funzioni del programma.
Essa, data in ingresso l'intera struttura dati, restituisce un intero che rappresenta la qualità del sistema.
Parlando in termini di Simulated Annealing, la qualità nel nostro orario è **equiparabile** all'energia del sistema nell'annealing.

Per poter restituire una qualità coerente è stato necessario definire una serie di vincoli non obbligatori e ad essi assegnare un peso, privilegiando quelli con priorità maggiore. E' utile dire che se la funzione di qualità non è coerente e i pesi sono sbagliati, l'intera euristica SA non convergerà al risultato sperato.

Qui di seguito riporto i vincoli più importanti con i rispettivi pesi da noi utilizzati:

- Se un professore un giorno fa piu di 4 ore lavorative sommo, nella qualità, 3*ogni ora aggiuntiva di lavoro;
- Se un professore alla settimana fa meno di 5 giorni lavorativi sommo, nella qualità, 8*ogni giorno non fatto;
- Se una materia che ha poche ore, al giorno fa più di un'ora sommo, nella qualità, 5*ogni ora aggiuntiva;
- Se una materia che ha poche ore ha lezione anche il giorno successivo sommo, nella qualità, 2* ogni ora eseguita il giorno successivo;
- Se una materia non arriva al numero massimo di ore consecutive sommo, nella qualità, 2*ogni ora mancante.
- Se una materia ha gia avuto il numero massimo di ore consecutive e un altro giorno tento di assegnare più di un'ora consecutiva, allora sommo, nella qualità, 2*ogni ora in più;
- Se una materia non raggiunge nella settimana il numero massimo di ore consecutive allora sommo 20 alla qualità.
- Per ogni prima o ultima ora "di troppo" alla settimana sommo 5 alla qualità.

In pratica la qualità tiene conto :

- Della distribuzione delle ore nella settimana , ad esempio non tollera che le ore di una materia si concentrino nella prima parte settimanale ;
- Della distribuzione delle ore nei singoli giorni, ad esempio non tollera :
    - Buchi tra due ore di spiegazione;
    - Che la materia in una singola classe venga insegnata sempre alla prima o all'ultima ora;
    - Che un professore venga a scuola per insegnare la prima e l'ultima ora.
- Delle ore di troppo per una materia, ad esempio non tollera che matematica abbia 4 ore nella stessa classe consecutive.

Tuttavia i pesi che attualmente i vincoli hanno, sono stati assegnati in modo abbastanza superficiale e non a

seguito di una dovuta analisi.

La funzione di qualità inoltre è nata anche per gestire gli eventuali desiderata che un professore può richiedere (un giorno libero piuttosto che un altro, delle ore buche in alcuni giorni …) , tuttavia non è stato possibile completare questo aspetto che comunque sarebbe integrabile in poco tempo.

La funzione salvatutto e rebuild:

La funzione salvatutto accetta in ingresso la struttura dati e in uscita restituisce una stringa di byte che identificano la tabella corrente.
Questa funzione ci serve soprattutto nei processi paralleli, infatti quando stiamo lavorando con più macchine e vogliamo conoscere l'orario migliore trovato fino ad allora da tutte le macchine, dobbiamo, necessariamente, farci mandare in qualche modo tutta la struttura dati dal processo che detiene l'orario migliore. Per eseguire questa operazione abbiamo preferito mandare un vettore di byte che la struttura dati intera per due motivi:

1. La soluzione del vettore è piu veloce in esecuzione e rende il codice sorgente più leggibile;
2. La soluzione del vettore permette di utilizzare una sola chiamata a sistema per mandare l'intera struttura dati.

Una volta che la macchina dove opera direttamente l'utente ha ricevuto questa sequenza binaria, per ricostruire l'intera struttura dati utilizziamo la funzione rebuild, la quale in ingresso accetta la sequenza binaria e in uscita fornisce la struttura dati.

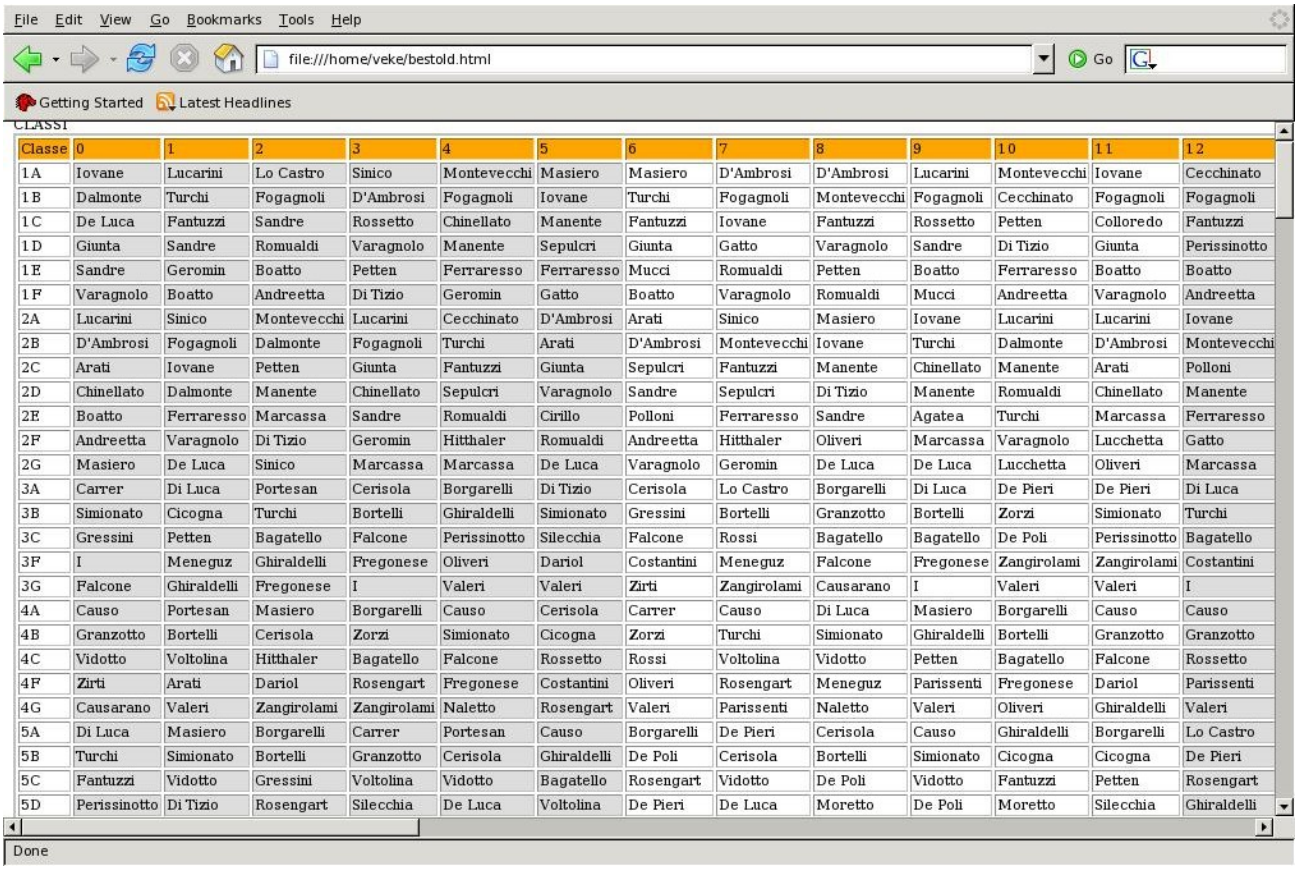
Queste due funzioni ci sono servite anche prima dell'implementazione del backtrack.Infatti prima di eseguire gli scambi in senso inverso, facevamo una copia dell'intera struttura dati prima di ogni scambio con *salvatutto(),* e nel caso avessimo dovuto tornare indietro, eliminavamo dalla memoria la struttura dati corrente e la ricostruivamo con *rebuild().* Questo processo, ovviamente, era molto dispendioso in termini di risorse e perciò abbiamo optato per il backtrack.

La funzione stampastrutturadati:

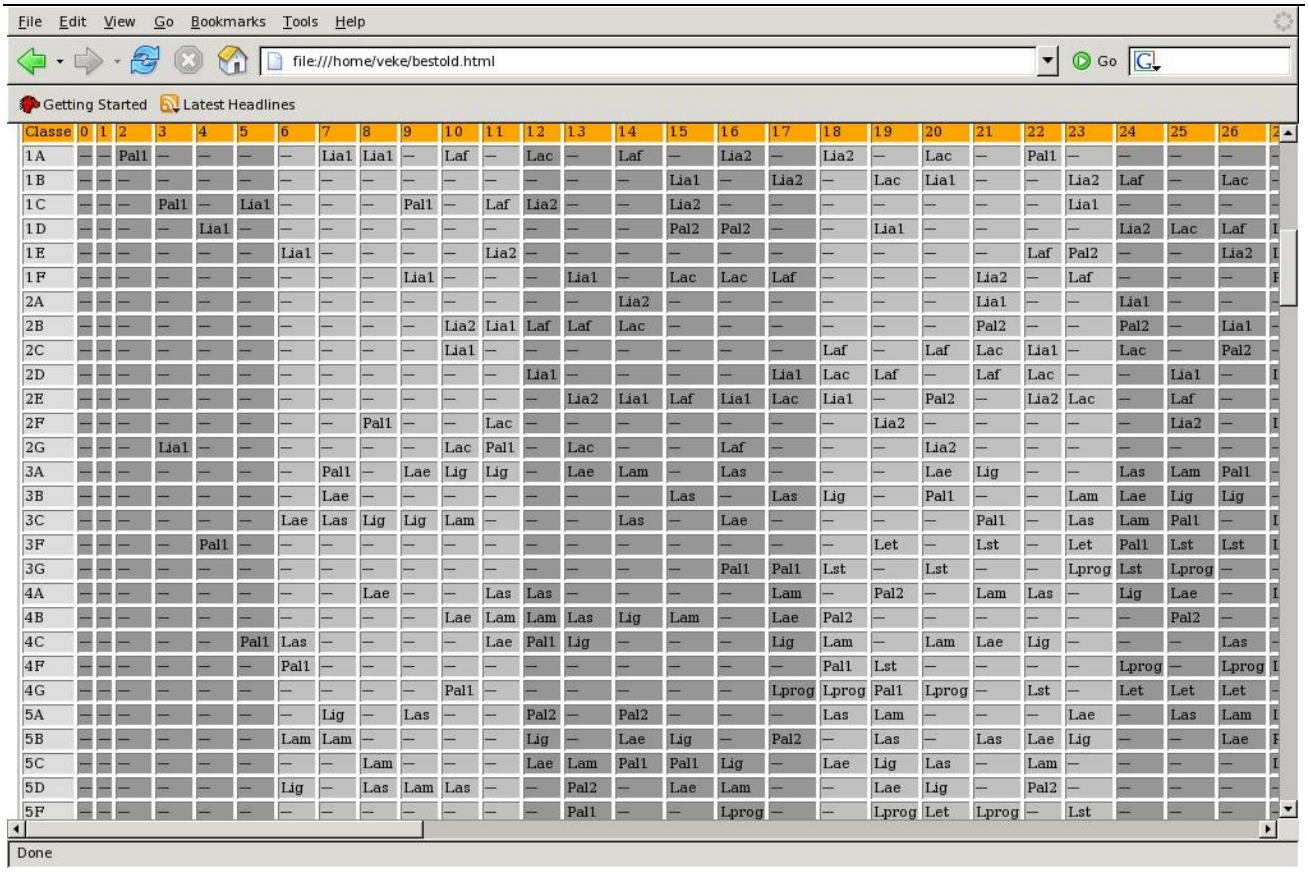
Come visto per orario.h anche in questo sorgente è stata scritta una funzione che è servita moltissimo in fase di sviluppo come verifica di correttezza del programma, ma soprattutto serve per visualizzare il risultato ottenuto. Questa funzione accetta in ingresso la struttura dati e il nome del file e come output scrive su disco il file con estensione .html contenente le seguenti tabelle:

- Classi / Ore : Per ogni classe e ogni ora inserisco l'insegnate assegnato.
- Classi/Ore : Per ogni classe e ogni ora inserisco l'eventuale laboratorio.
- Professori/Ore: Per ogni professore e ogni ora inserisco l'eventuale classe assegnata.
- Professori/Ore: Per ogni professore e ogni ora inserisco l'eventuale laboratorio.

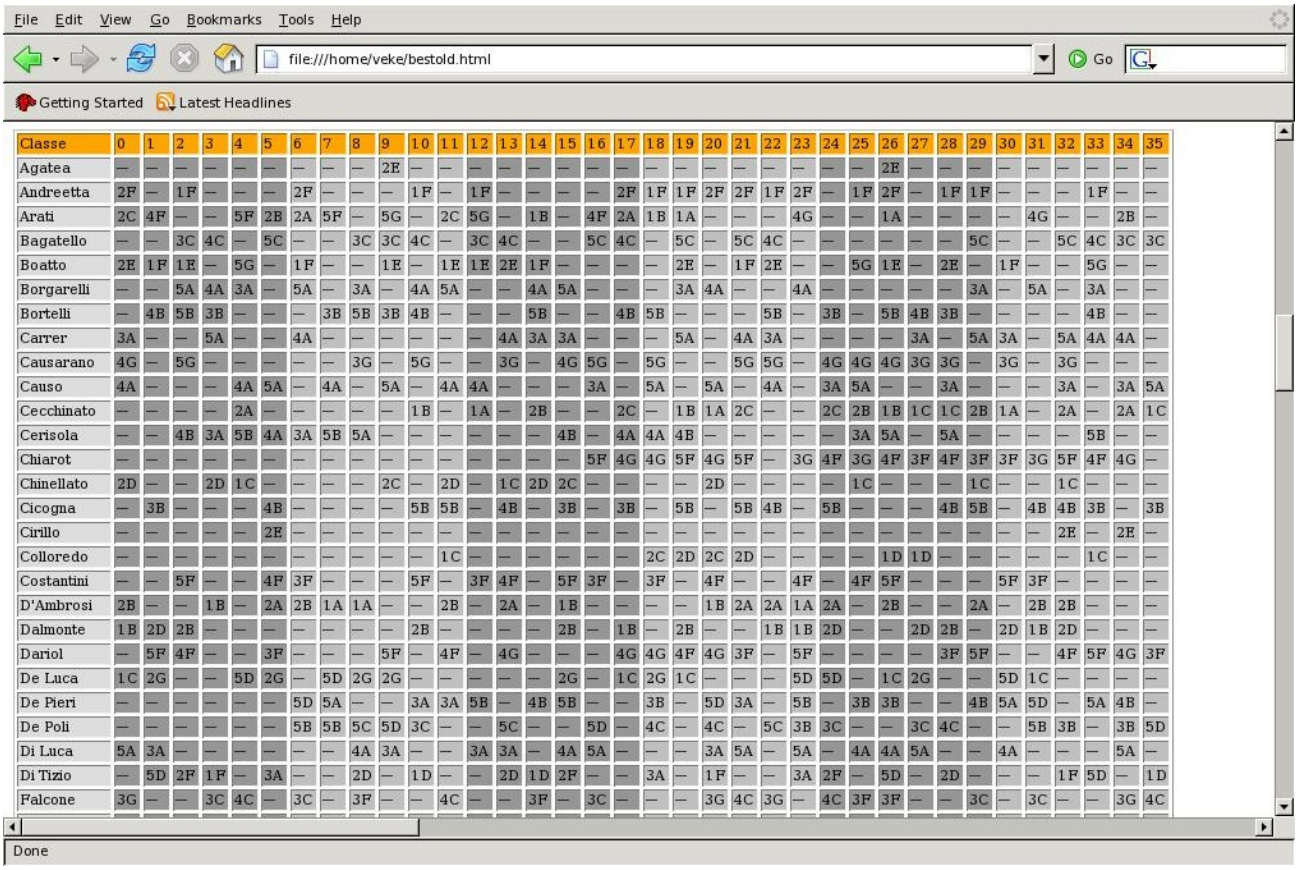
Queste quattro tabelle sono proprio come noi ce le aspettiamo. Un vero e proprio orario.
Qui di seguito riporto lo screenshot di un orario casuale generato:



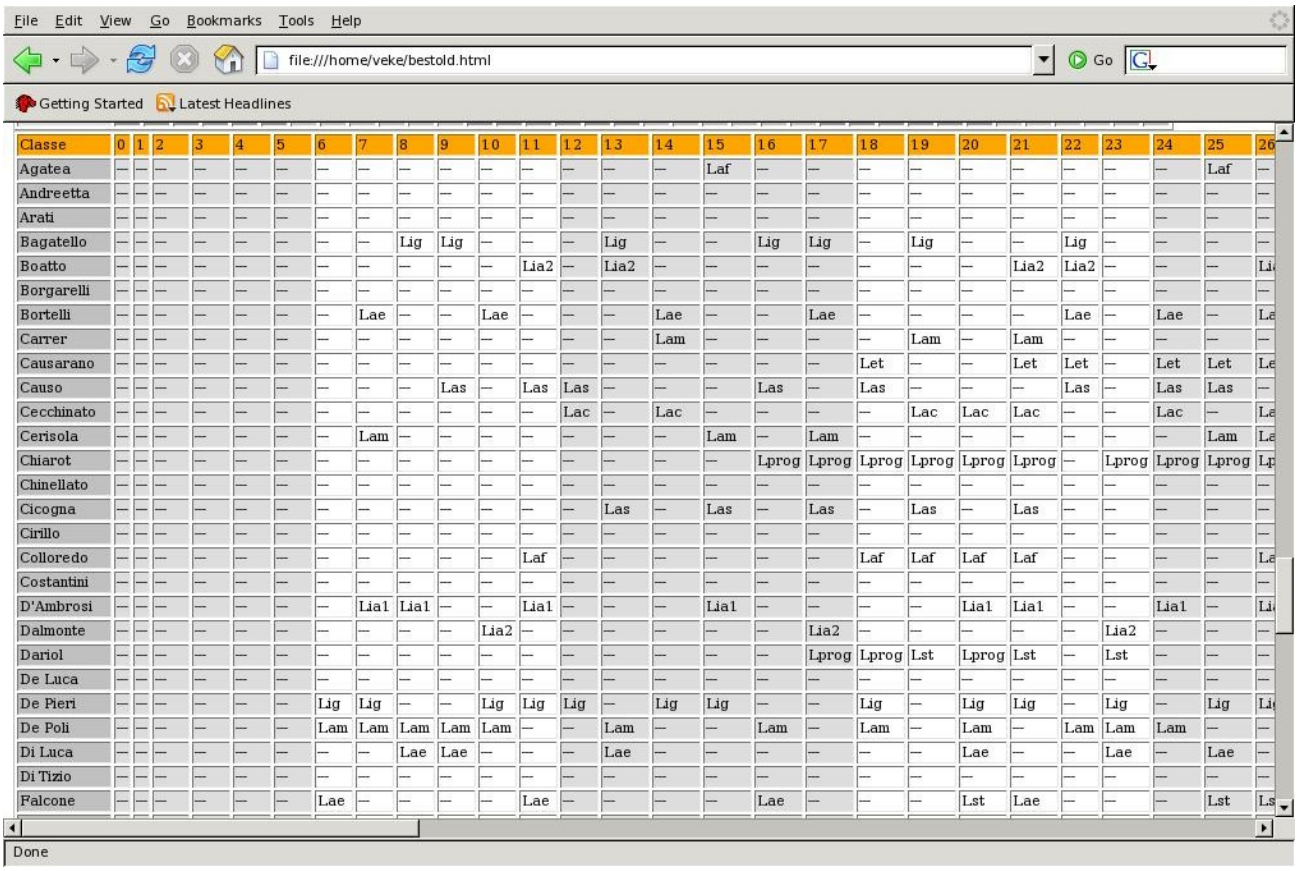
Classe	0	1	2	3	4	5	6	7	8	9	10	11	12
1A	Iovane	Lucarini	Lo Castro	Sinico	Montevecchi	Masiero	Masiero	D'Ambrosi	D'Ambrosi	Lucarini	Montevecchi	Iovane	Cecchinato
1B	Dalmonte	Turchi	Fogagnoli	D'Ambrosi	Fogagnoli	Iovane	Turchi	Fogagnoli	Montevecchi	Fogagnoli	Cecchinato	Fogagnoli	Fogagnoli
1C	De Luca	Fantuzzi	Sandre	Rossetto	Chinellato	Manente	Fantuzzi	Iovane	Fantuzzi	Rossetto	Petten	Colloredo	Fantuzzi
1D	Giunta	Sandre	Romualdi	Varagnolo	Manente	Sepulcri	Giunta	Gatto	Varagnolo	Sandre	Di Tizio	Giunta	Perissinotto
1E	Sandre	Geromin	Boatto	Petten	Ferraresso	Ferraresso	Mucci	Romualdi	Petten	Boatto	Ferraresso	Boatto	Boatto
1F	Varagnolo	Boatto	Andreetta	Di Tizio	Geromin	Gatto	Boatto	Varagnolo	Romualdi	Mucci	Andreetta	Varagnolo	Andreetta
2A	Lucarini	Sinico	Montevecchi	Lucarini	Cecchinato	D'Ambrosi	Arati	Sinico	Masiero	Iovane	Lucarini	Lucarini	Iovane
2B	D'Ambrosi	Fogagnoli	Dalmonte	Fogagnoli	Turchi	Arati	D'Ambrosi	Montevecchi	Iovane	Turchi	Dalmonte	D'Ambrosi	Montevecchi
2C	Arati	Iovane	Petten	Giunta	Fantuzzi	Giunta	Sepulcri	Fantuzzi	Manente	Chinellato	Manente	Arati	Polloni
2D	Chinellato	Dalmonte	Manente	Chinellato	Sepulcri	Varagnolo	Sandre	Sepulcri	Di Tizio	Manente	Romualdi	Chinellato	Manente
2E	Boatto	Ferraresso	Marcassa	Sandre	Romualdi	Cirillo	Polloni	Ferraresso	Sandre	Agatea	Turchi	Marcassa	Ferraresso
2F	Andreetta	Varagnolo	Di Tizio	Geromin	Hitthaler	Romualdi	Andreetta	Hitthaler	Oliveri	Marcassa	Varagnolo	Lucchetta	Gatto
2G	Masiero	De Luca	Sinico	Marcassa	Marcassa	De Luca	Varagnolo	Geromin	De Luca	De Luca	Lucchetta	Oliveri	Marcassa
3A	Carrer	Di Luca	Portesan	Cerisola	Borgarelli	Di Tizio	Cerisola	Lo Castro	Borgarelli	Di Luca	De Pieri	De Pieri	Di Luca
3B	Simionato	Cicogna	Turchi	Bortelli	Ghiraldelli	Simionato	Gressini	Bortelli	Granzotto	Bortelli	Zorzi	Simionato	Turchi
3C	Gressini	Petten	Bagatello	Falcone	Perissinotto	Silecchia	Falcone	Rossi	Bagatello	Bagatello	De Poli	Perissinotto	Bagatello
3F	I	Meneguz	Ghiraldelli	Fregonese	Oliveri	Dariol	Costantini	Meneguz	Falcone	Fregonese	Zangirolami	Zangirolami	Costantini
3G	Falcone	Ghiraldelli	Fregonese	I	Valeri	Valeri	Zirti	Zangirolami	Causarano	I	Valeri	Valeri	I
4A	Causo	Portesan	Masiero	Borgarelli	Causo	Cerisola	Carrer	Causo	Di Luca	Masiero	Borgarelli	Causo	Causo
4B	Granzotto	Bortelli	Cerisola	Zorzi	Simionato	Cicogna	Zorzi	Turchi	Simionato	Ghiraldelli	Bortelli	Granzotto	Granzotto
4C	Vidotto	Voltolina	Hitthaler	Bagatello	Falcone	Rossetto	Rossi	Voltolina	Vidotto	Petten	Bagatello	Falcone	Rossetto
4F	Zirti	Arati	Dariol	Rosengart	Fregonese	Costantini	Oliveri	Rosengart	Meneguz	Parissenti	Fregonese	Dariol	Parissenti
4G	Causarano	Valeri	Zangirolami	Zangirolami	Naletto	Rosengart	Valeri	Parissenti	Naletto	Valeri	Oliveri	Ghiraldelli	Valeri
5A	Di Luca	Masiero	Borgarelli	Carrer	Portesan	Causo	Borgarelli	De Pieri	Cerisola	Causo	Ghiraldelli	Borgarelli	Lo Castro
5B	Turchi	Simionato	Bortelli	Granzotto	Cerisola	Ghiraldelli	De Poli	Cerisola	Bortelli	Simionato	Cicogna	Cicogna	De Pieri
5C	Fantuzzi	Vidotto	Gressini	Voltolina	Vidotto	Bagatello	Rosengart	Vidotto	De Poli	Vidotto	Fantuzzi	Petten	Rosengart
5D	Perissinotto	Di Tizio	Rosengart	Silecchia	De Luca	Voltolina	De Pieri	De Luca	Moretto	De Poli	Moretto	Silecchia	Ghiraldelli



Classe	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1A	–	–	Pal1	–	–	–	–	Lia1	Lia1	–	Laf	–	Lac	–	Laf	–	Lia2	–	Lia2	–	Lac	–	Pal1	–	–	–	–
1B	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Lia1	–	Lia2	–	Lac	Lia1	–	–	Lia2	Laf	–	Lac
1C	–	–	–	Pal1	–	Lia1	–	–	–	Pal1	–	Laf	Lia2	–	–	Lia2	–	–	–	–	–	–	–	Lia1	–	–	–
1D	–	–	–	–	Lia1	–	–	–	–	–	–	–	–	–	–	Pal2	Pal2	–	–	Lia1	–	–	–	–	Lia2	Lac	Laf
1E	–	–	–	–	–	–	Lia1	–	–	–	–	Lia2	–	–	–	–	–	–	–	–	–	–	Laf	Pal2	–	–	Lia2
1F	–	–	–	–	–	–	–	–	–	Lia1	–	–	–	Lia1	–	Lac	Lac	Laf	–	–	–	Lia2	–	Laf	–	–	–
2A	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Lia2	–	–	–	–	–	–	Lia1	–	–	Lia1	–	–
2B	–	–	–	–	–	–	–	–	–	–	Lia2	Lia1	Laf	Laf	Lac	–	–	–	–	–	–	Pal2	–	–	Pal2	–	Lia1
2C	–	–	–	–	–	–	–	–	–	–	Lia1	–	–	–	–	–	–	–	Laf	–	Laf	Lac	Lia1	–	Lac	–	Pal2
2D	–	–	–	–	–	–	–	–	–	–	–	–	Lia1	–	–	–	–	Lia1	Lac	Laf	–	Laf	Lac	–	–	Lia1	–
2E	–	–	–	–	–	–	–	–	–	–	–	–	–	Lia2	Lia1	Laf	Lia1	Lac	Lia1	–	Pal2	–	Lia2	Lac	–	Laf	–
2F	–	–	–	–	–	–	–	–	Pal1	–	–	Lac	–	–	–	–	–	–	–	Lia2	–	–	–	–	–	Lia2	–
2G	–	–	–	Lia1	–	–	–	–	–	–	Lac	Pal1	–	Lac	–	–	Laf	–	–	–	Lia2	–	–	–	–	–	–
3A	–	–	–	–	–	–	–	Pal1	–	Lae	Lig	Lig	–	Lae	Lam	–	Las	–	–	–	Lae	Lig	–	–	Las	Lam	Pal1
3B	–	–	–	–	–	–	–	Lae	–	–	–	–	–	–	–	Las	–	Las	Lig	–	Pal1	–	–	Lam	Lae	Lig	Lig
3C	–	–	–	–	–	–	Lae	Las	Lig	Lig	Lam	–	–	–	Las	–	Lae	–	–	–	–	Pal1	–	Las	Lam	Pal1	–
3F	–	–	–	–	Pal1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Let	–	Lst	–	Let	Pal1	Lst	Lst
3G	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Pal1	Pal1	Lst	–	Lst	–	–	Lprog	Lst	Lprog	–
4A	–	–	–	–	–	–	–	–	Lae	–	–	Las	Las	–	–	–	–	Lam	–	Pal2	–	Lam	Las	–	Lig	Lae	–
4B	–	–	–	–	–	–	–	–	–	–	Lae	Lam	Lam	Las	Lig	Lam	–	Lae	Pal2	–	–	–	–	–	–	Pal2	–
4C	–	–	–	–	–	Pal1	Las	–	–	–	–	Lae	Pal1	Lig	–	–	–	Lig	Lam	–	Lam	Lae	Lig	–	–	–	Las
4F	–	–	–	–	–	–	Pal1	–	–	–	–	–	–	–	–	–	–	–	Pal1	Lst	–	–	–	–	Lprog	–	Lprog
4G	–	–	–	–	–	–	–	–	–	–	Pal1	–	–	–	–	–	–	Lprog	Lprog	Pal1	Lprog	–	Lst	–	Let	Let	Let
5A	–	–	–	–	–	–	–	Lig	–	Las	–	–	Pal2	–	Pal2	–	–	–	Las	Lam	–	–	–	Lae	–	Las	Lam
5B	–	–	–	–	–	–	Lam	Lam	–	–	–	–	Lig	–	Lae	Lig	–	Pal2	–	Las	–	Las	Lae	Lig	–	–	Lae
5C	–	–	–	–	–	–	–	–	Lam	–	–	–	Lae	Lam	Pal1	Pal1	Lig	–	Lae	Lig	Las	–	Lam	–	–	–	–
5D	–	–	–	–	–	–	Lig	–	Las	Lam	Las	–	–	Pal2	–	Lae	Lam	–	–	Lae	Lig	–	Pal2	–	–	–	–
5F	–	–	–	–	–	–	–	–	–	–	–	–	–	Pal1	–	–	Lprog	–	–	Lprog	Let	Lprog	–	Lst	–	–	–



Classe	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
Agatea	–	–	–	–	–	–	–	–	–	2E	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2E	–	–	–	–	–	–	–	–	–
Andreetta	2F	–	1F	–	–	–	2F	–	–	–	1F	–	1F	–	–	–	–	2F	1F	1F	2F	2F	1F	2F	–	1F	2F	–	1F	1F	–	–	–	1F	–	–
Arati	2C	4F	–	–	5F	2B	2A	5F	–	5G	–	2C	5G	–	1B	–	4F	2A	1B	1A	–	–	–	4G	–	–	1A	–	–	–	–	4G	–	–	2B	–
Bagatello	–	–	3C	4C	–	5C	–	–	3C	3C	4C	–	3C	4C	–	–	5C	4C	–	5C	–	5C	4C	–	–	–	–	–	–	5C	–	–	5C	4C	3C	3C
Boatto	2E	1F	1E	–	5G	–	1F	–	–	1E	–	1E	1E	2E	1F	–	–	–	–	2E	–	1F	2E	–	–	5G	1E	–	2E	–	1F	–	–	5G	–	–
Borgarelli	–	–	5A	4A	3A	–	5A	–	3A	–	4A	5A	–	–	4A	5A	–	–	–	3A	4A	–	–	4A	–	–	–	–	–	3A	–	5A	–	3A	–	–
Bortelli	–	4B	5B	3B	–	–	–	3B	5B	3B	4B	–	–	–	5B	–	–	4B	5B	–	–	–	5B	–	3B	–	5B	4B	3B	–	–	–	–	4B	–	–
Carrer	3A	–	–	5A	–	–	4A	–	–	–	–	–	–	4A	3A	3A	–	–	–	5A	–	4A	3A	–	–	–	–	3A	–	5A	3A	–	5A	4A	4A	–
Causarano	4G	–	5G	–	–	–	–	–	3G	–	5G	–	–	3G	–	4G	5G	–	5G	–	–	5G	5G	–	4G	4G	4G	3G	3G	–	3G	–	3G	–	–	–
Causo	4A	–	–	–	4A	5A	–	4A	–	5A	–	4A	4A	–	–	–	3A	–	5A	–	5A	–	4A	–	3A	5A	–	–	3A	–	–	–	3A	–	3A	5A
Cecchinato	–	–	–	–	2A	–	–	–	–	–	1B	–	1A	–	2B	–	–	2C	–	1B	1A	2C	–	–	2C	2B	1B	1C	1C	2B	1A	–	2A	–	2A	1C
Cerisola	–	–	4B	3A	5B	4A	3A	5B	5A	–	–	–	–	–	–	4B	–	4A	4A	4B	–	–	–	–	–	3A	5A	–	5A	–	–	–	–	5B	–	–
Chiarot	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	5F	4G	4G	5F	4G	5F	–	3G	4F	3G	4F	3F	4F	3F	3F	3G	5F	4F	4G	–
Chinellato	2D	–	–	2D	1C	–	–	–	–	2C	–	2D	–	1C	2D	2C	–	–	–	–	2D	–	–	–	–	1C	–	–	–	1C	–	–	1C	–	–	–
Cicogna	–	3B	–	–	–	4B	–	–	–	–	5B	5B	–	4B	–	3B	–	3B	–	5B	–	5B	4B	–	5B	–	–	–	4B	5B	–	4B	4B	3B	–	3B
Cirillo	–	–	–	–	–	2E	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2E	–	2E	–
Colloredo	–	–	–	–	–	–	–	–	–	–	1C	–	–	–	–	–	2C	2D	2C	2D	–	–	–	–	1D	1D	–	–	–	–	–	–	1C	–	–	–
Costantini	–	–	5F	–	–	4F	3F	–	–	–	5F	–	3F	4F	–	5F	3F	–	3F	–	4F	–	–	4F	–	4F	5F	–	–	–	5F	3F	–	–	–	–
D'Ambrosi	2B	–	–	1B	–	2A	2B	1A	1A	–	–	2B	–	2A	–	1B	–	–	–	–	1B	2A	2A	1A	2A	–	2B	–	–	2A	–	2B	2B	–	–	–
Dalmonte	1B	2D	2B	–	–	–	–	–	–	2B	–	–	–	–	2B	–	1B	–	2B	–	–	1B	1B	2D	–	–	2D	2B	–	2D	1B	2D	–	–	–	–
Dariol	–	5F	4F	–	–	3F	–	–	–	5F	–	4F	–	4G	–	–	–	4G	4G	4F	4G	3F	–	5F	–	–	–	–	3F	5F	–	–	4F	5F	4G	3F
De Luca	1C	2G	–	–	5D	2G	–	5D	2G	2G	–	–	–	–	–	2G	–	1C	2G	1C	–	–	–	5D	5D	–	1C	2G	–	–	5D	1C	–	–	–	–
De Pieri	–	–	–	–	–	–	5D	5A	–	–	3A	3A	5B	–	4B	5B	–	–	3B	–	5D	3A	–	5B	–	3B	3B	–	–	4B	5A	5D	–	5A	4B	–
De Poli	–	–	–	–	–	–	5B	5B	5C	5D	3C	–	–	5C	–	–	5D	–	4C	–	4C	–	5C	3B	3C	–	–	3C	4C	–	–	5B	3B	–	3B	5D
Di Luca	5A	3A	–	–	–	–	–	–	4A	3A	–	–	3A	3A	–	4A	5A	–	–	–	3A	5A	–	5A	–	4A	4A	5A	–	–	4A	–	–	–	5A	–
Di Tizio	–	5D	2F	1F	–	3A	–	–	2D	–	1D	–	–	2D	1D	2F	–	–	3A	–	1F	–	–	3A	2F	–	5D	–	2D	–	–	–	1F	5D	–	1D
Falcone	3G	–	–	3C	4C	–	3C	–	3F	–	–	4C	–	–	3F	–	3C	–	–	–	3G	4C	3G	–	4C	3F	3F	–	–	3C	–	3C	–	–	3G	4C



Classe	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Agatea	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Laf	–	–	–	–	–	–	–	–	–	Laf	–
Andreetta	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Arati	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Bagatello	–	–	–	–	–	–	–	–	Lig	Lig	–	–	–	Lig	–	–	Lig	Lig	–	Lig	–	–	Lig	–	–	–	–
Boatto	–	–	–	–	–	–	–	–	–	–	–	Lia2	–	Lia2	–	–	–	–	–	–	–	–	Lia2	Lia2	–	–	–	Li
Borgarelli	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Bortelli	–	–	–	–	–	–	–	Lae	–	–	Lae	–	–	–	Lae	–	–	Lae	–	–	–	–	Lae	–	Lae	–	La
Carrer	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Lam	–	–	–	–	Lam	–	Lam	–	–	–	–	–
Causarano	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Let	–	–	Let	Let	–	Let	Let	Le
Causo	–	–	–	–	–	–	–	–	–	Las	–	Las	Las	–	–	–	Las	–	Las	–	–	–	Las	–	Las	Las	–
Cecchinato	–	–	–	–	–	–	–	–	–	–	–	–	Lac	–	Lac	–	–	–	–	Lac	Lac	Lac	–	–	Lac	–	La
Cerisola	–	–	–	–	–	–	–	Lam	–	–	–	–	–	–	–	Lam	–	Lam	–	–	–	–	–	–	–	Lam	La
Chiarot	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Lprog	Lprog	Lprog	Lprog	Lprog	Lprog	–	Lprog	Lprog	Lprog	Lp
Chinellato	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Cicogna	–	–	–	–	–	–	–	–	–	–	–	–	–	Las	–	Las	–	Las	–	Las	–	Las	–	–	–	–	–
Cirillo	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Colloredo	–	–	–	–	–	–	–	–	–	–	–	Laf	–	–	–	–	–	–	Laf	Laf	Laf	Laf	–	–	–	–	La
Costantini	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
D'Ambrosi	–	–	–	–	–	–	–	Lia1	Lia1	–	–	Lia1	–	–	–	Lia1	–	–	–	–	Lia1	Lia1	–	–	Lia1	–	Li
Dalmonte	–	–	–	–	–	–	–	–	–	–	Lia2	–	–	–	–	–	–	Lia2	–	–	–	–	–	Lia2	–	–	–
Dariol	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	Lprog	Lprog	Lst	Lprog	Lst	–	Lst	–	–	–
De Luca	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
De Pieri	–	–	–	–	–	Lig	Lig	–	–	Lig	Lig	Lig	–	Lig	Lig	–	–	–	Lig	–	Lig	Lig	–	Lig	–	Lig	Li
De Poli	–	–	–	–	–	Lam	Lam	Lam	Lam	Lam	–	–	–	Lam	–	–	Lam	–	Lam	–	Lam	–	Lam	Lam	Lam	–	–
Di Luca	–	–	–	–	–	–	–	Lae	Lae	–	–	–	–	Lae	–	–	–	–	–	–	Lae	–	–	Lae	–	Lae	–
Di Tizio	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Falcone	–	–	–	–	–	–	Lae	–	–	–	–	Lae	–	–	–	–	Lae	–	–	–	Lst	Lae	–	–	–	Lst	Ls

L’interfacciamento con gnuplot:

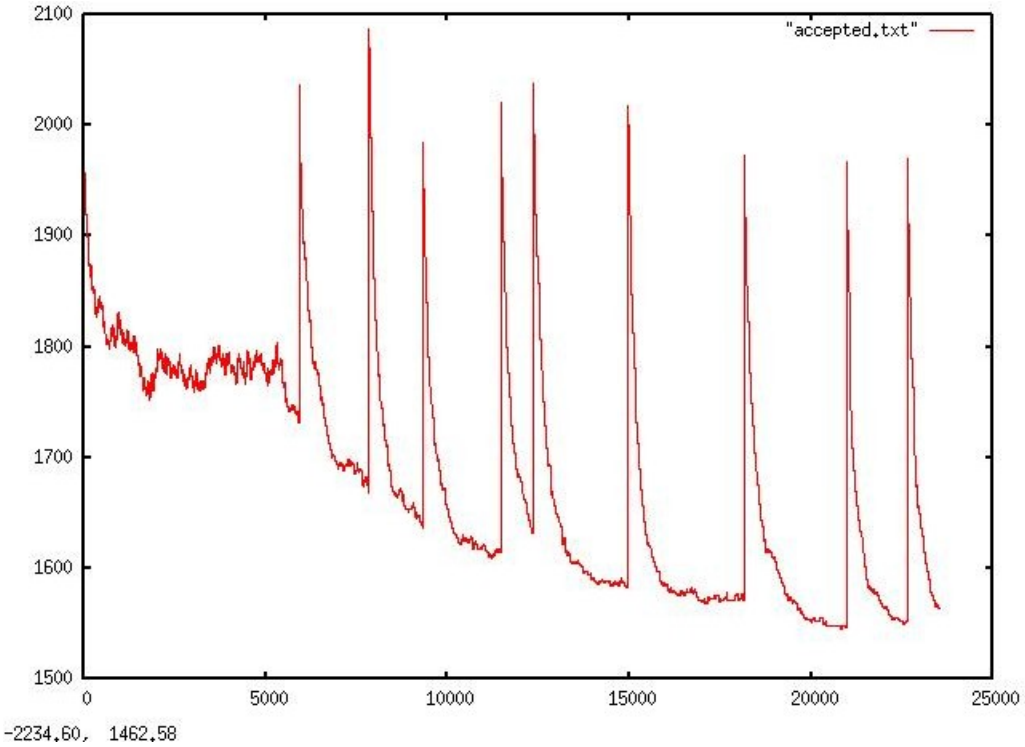
Per verificare l’andamento della funzione SA, e verificare di conseguenza la correttezza dei parametri, abbiamo deciso di tracciare il grafico delle qualità degli orari accettati dall’euristica. Per effettuare questo, ad ogni scambio accettato, scriviamo in un file la nuova qualità della tabella appena creata, successivamente questo viene letto periodicamente da un programma che richiama gnuplot.
Il file che utilizza gnuplot per leggere le qualità è il seguente:

```
Plot “accepted.txt” with lines
Pause 3
Reread
```

Questo piccolo file richiamato come segue:

veke@veke$ gnuplot gnuplot.gn

Dove gnuplot.gn è il piccolo file di comandi che genera il seguente grafico aggiornato ogni 3 secondi dinamicamente:



Nell’asse delle x abbiamo il numero di cicli, e nelle ordinate la qualità.

Come si può notare nella parte iniziale [0;5000] del grafico abbiamo degli sfasamenti molto ampi di qualità accettate. Nella parte successiva del grafico la qualità tende a scendere e inoltre si vedono dei picchi di qualità altissime accettate.
Questi picchi sono frutto di una funzione da noi creata che genera scambi di profondità 1 non regolati dall’euristica SA. Sarebbe probabilmente una contraddizione uscire dall’ SA , tuttavia ci siamo accorti dopo vari test che la qualità non scendeva. Perciò siamo stati indotti a credere che l’euristica ci isolasse in una configurazione della struttura dati.
Abbiamo creato, quindi, una funzione che facesse scambi assolutamente casuali senza alcun tipo di vincolo su essi in modo da perturbare di molto la struttura dati.

Tuttavia, quando ci siamo trovati nel gestire 24 grafici di gnuplot, di 24 singole macchine differenti, ci siamo dovuti un attimo ravvedere sull'entita dei dati da tracciare.
Infatti se avessimo voluto stampare per ogni processo le qualità accettate, si sarebbero create moltissime comunicazioni tra i vari host, così da generare un rallentamento significativo e inutile dell'elaborazione.
Ci siamo orientati, quindi, per la stampa della qualità migliore di ogni processo , e solamente ogni ciclo esterno.
Il risultato è stato un cambiamento esiguo del codice , e la creazione di un altro programmino che crea automaticamente dato il numero dei processi il file gnuplot.gn.

```
#!/bin/bash
echo –n "plot 'quality0.txt' with lines" > gnuplot.gn
for ((i=1; i<$1; i++)); do
  echo –n " , 'quality$i.txt' with lines" >> gnuplot.gn;
done
echo >> gnuplot.gn
echo "pause 3" >>gnuplot.gn
echo "reread" >>gnuplot.gn
```

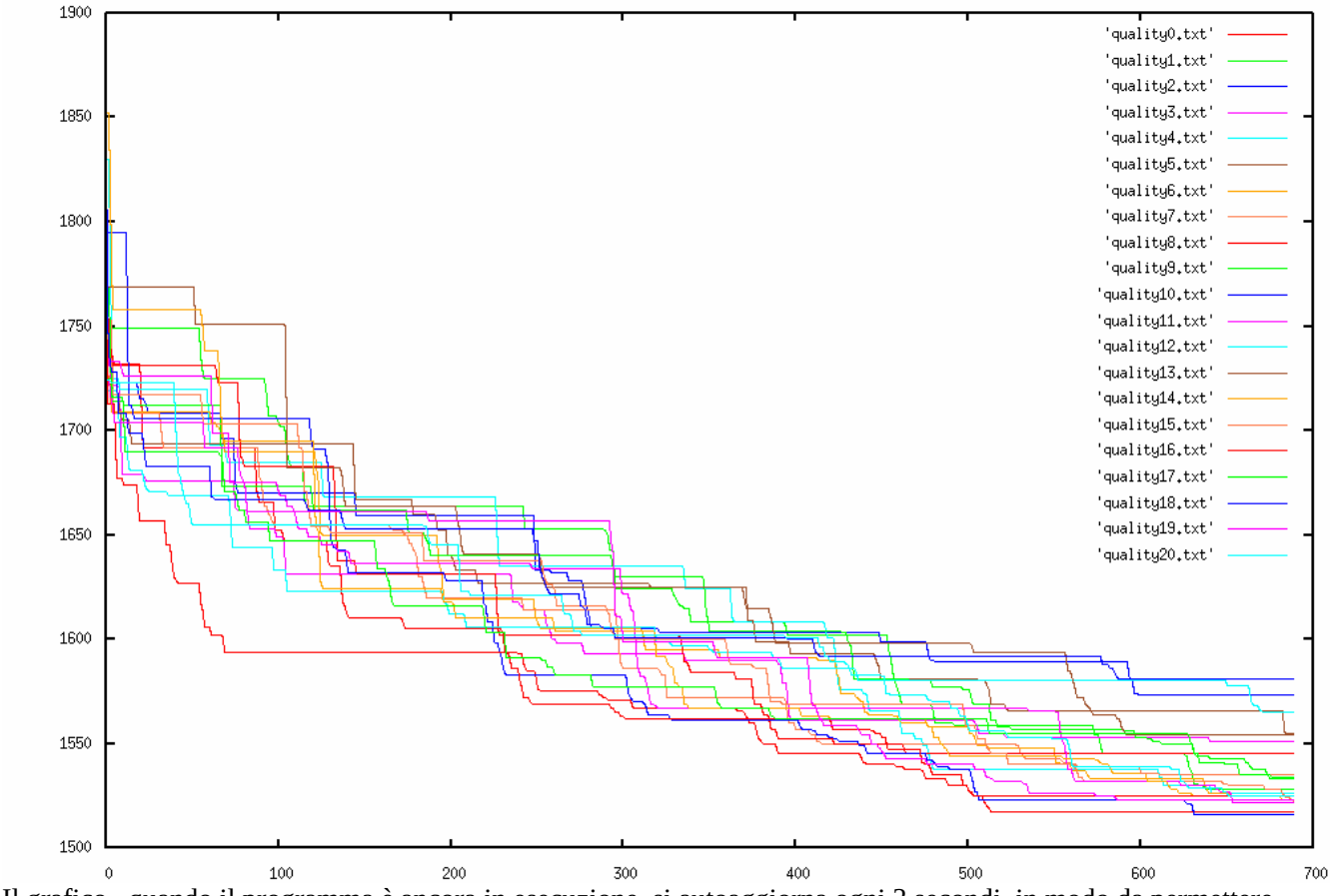
Questo programmino richiamato nel seguente modo:

veke@veke$ ./programma.sh NumeroComputer

genera automaticamente il file configurato per gnuplot in modo che gestisca N macchine.
Il programma orario.c si preoccupa di creare ed aggiornare dei file chiamati *qualityX.txt* . Esiste un quality per ogni processo. Quando lanciamo il programma orario in parallelo , e lo lasciamo andare grazie al gnuplot e al programmino appena visto, digitando il seguente comando nella console :

veke@veke$ gnuplot gnuplot.gn

Otteniamo il seguente risultato:



Il grafico , quando il programma è ancora in esecuzione, si autoaggiorna ogni 3 secondi, in modo da permettere

all’operatore di monitorare quale sia l’andamento dell’euristica SA, e in generale del programma.

Come nel grafico singolo, nell’asse x abbiamo il numero di cicli esterni, mentre nell’asse ‘y’ abbiamo la qualità . Tuttavia le differenze tra il grafico di un solo processo e questo sono date da:

1. Presenza di molte linee ognuna corrispondente ad un computer e non più una singola linea.
2. Ogni linea qui è monotona e decrescente ; questo perché prima lavoravamo con ogni qualità accettata mentre ora operiamo solamente sulla qualità migliore ottenuta da quel processo.E dato che questa è destinata a diminuire, la funzione tracciata nel grafico sarà monotona decrescente.

11. I risultati

Il risultato è stato un programma funzionante in tutte le sue parti, anche se ancora limitato e non elastico in alcune.

Riguardo l'euristica, come si può notare dal grafico multi-host, l'SA tende a far diminuire la best quality anche in un tempo molto esiguo; tuttavia non abbiamo potuto testare sufficientemente il programma cambiando i parametri che intervengono direttamente nella convergenza dell'orario.

Inoltre nei nostri pochissimi test abbiamo notato che con l'avanzare del programma, nel grafico multi-host, le best quality si 'schiacciavano' in una qualità che faticavano a superare.

Nella parallelizzazione temevamo che due o più elaboratori avessero in esame lo stesso orario o comunque orari molto simili. In questo caso, la parallelizzazione sarebbe stata inutile, in quanto due processi facevano esattamente le stesse operazioni senza guadagno alcuno per noi.Questo problema è stato risolto implementando il generatore di numeri pseudo-casuali lagged fibonacci, che ci garantisce una maggiore omogeneità dei numeri generati. Soprattutto nei grafici ottenuti dai vari processi, si è notato che, a parità di cicli, questi non s'assomigliavano, concludendo così che questi operavano su zone differenti.

12. Il codice sorgente

```
// Heap usati come priority queue Max HEAP
struct Priority_queue {
	void **pq;
	uint32_t _index;
};
typedef struct Priority_queue PQ;
PQ *creaHeap(uint32_t npq,uint32_t nel) {
	uint32_t i;
	PQ *heap=(PQ *) malloc(npq*sizeof(PQ));
	for (i=0;i<npq;i++) {
		(heap[i].pq)=(void **) malloc(nel*sizeof(void *));
		heap[i]._index=0;
	}
	return heap;
}
void _push(PQ *_Heap,void *a ,int (*maggiore)(void *a,void *b),uint32_t size) {
	uint32_t i;
	void *tmp;
	_Heap->pq[_Heap->_index]=(void *) malloc(size);
	memcpy(_Heap->pq[_Heap->_index],a,size);
	//*(_Heap->pq[_Heap->_index])=*a;
	i=_Heap->_index;
	(_Heap->_index)++;
	while (i>0){
		if (maggiore(_Heap->pq[(i-1)/2],_Heap->pq[i]))
			i=0;
		else {
			tmp=_Heap->pq[i];
			_Heap->pq[i]=_Heap->pq[(i-1)/2];
			_Heap->pq[(i-1)/2]=tmp;
			i=(i-1)/2;
		}
	}
}
void *_pop (PQ *_Heap,int (*maggiore)(void *a,void *b)) {
	void *tmp;
	uint32_t i,j;
	tmp=_Heap->pq[0];
	i=0;
	do {
		if (2*(i+1)>=_Heap->_index ){
			_Heap->pq[i]=_Heap->pq[2*i+1];
			i=2*i+1;
		}else if( maggiore(_Heap->pq[2*i +1],_Heap->pq[2*(i+1)])){
			_Heap->pq[i]=_Heap->pq[2*i+1];
			i=2*i+1;
		}else {
			_Heap->pq[i]=_Heap->pq[2*(i+1)];
			i=2*(i+1);
		}

	} while (2*i+1<_Heap->_index);
	if (i<_Heap->_index)
		for (j=i;j<_Heap->_index -1 ;j++)
			_Heap->pq[j]=_Heap->pq[j+1];
	(_Heap->_index)--;
	return tmp;
```

```
}
void *_top (PQ *_Heap) {
	return _Heap->pq[0];
}
int _size(PQ *_Heap) {
	return _Heap->_index;
}
PQ *unisci_code(PQ *a, PQ *b,int (*__max_id)(const void *,const void *),PQ *(* __operazione)(void *,void *, int ), uint32_t size) {
	int i;
	void *ta=(void *) malloc(size*a->_index);
	void *tb=(void *) malloc(size*a->_index);
	memcpy(ta, a->pq[0], size*a->_index);
	memcpy(ta, b->pq[0], size*a->_index);
	qsort(ta, a->_index, size, __max_id);
	qsort(tb, b->_index, size, __max_id);
	return __operazione( ta, tb, a->_index );
}
```

Fine Heap.h ; Inizio Stack.h

```
// Stack.h
struct _stack {
	void **el;
	int nel,nmax;
};
typedef struct _stack  stack;
 stack* creastack(int nmaxel) {
	stack *s=(stack *) malloc(sizeof(stack));
	s->el=(void **) malloc(nmaxel*sizeof(void *));
	s->nel=0;
	s->nmax=nmaxel;
	return s;
}
// funzione che restituisce false se lo stack ha qualche elemento
int _sempty(stack *s) {
	if (s->nel >0)
		return 0;
	else
		return 1;
}
// funzione che restituisce false se lo stack ha ancora qualche elemento
int _sfull(stack *s) {
	if (s->nel < s->nmax)
		return 0;
	else
		return 1;
}
// funzione che pusha nello stack l'elemento , se non puo farlo allora restituisce 0
int _spush (stack *s, void * elem, int size) {
	if ( ! _sfull(s)) {
		s->el[s->nel]=malloc(size);
		memcpy(s->el[s->nel],elem,size);
		(s->nel)++;
		return 1;
	} else{
		printf ("Errore Stack pieno\ n");
		getchar();
		return 0;
	}
}
```

```
// funzione che prende l'ultimo elemento inserito , se non puo farlo allora restituisce NULL
void * _spop(stack *s ){
	if (! _sempty(s)) {
		(s->nel)--;
		return s->el[s->nel] ;
	}else
		return (void *)NULL;
}
// funzione che "svuota" lo stack utile se si vuole cancellare completamente lo stack per riutilizzarlo
void _strash(stack *s) {
	int i;
	for (i=0; i<s->nel; i++)
		free(s->el[i]);
	s->nel=0;
}
```

Fine Stack.h ; Inizio Orario.h

```
#include <stdint.h>
//#include "Trye.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#ifdef CPP
	#include <queue>
	using namespace std;
#endif
#define GIORNI 6
#define PROF 1
#define MATERIE 0
#define CLASSI 2
#define LABORATORI 3
#define OREGIORNO 6
#define BUFSIZE 128
void rebuildfrequency ();
struct _classe_prof {
	uint8_t id_prof; // id dell'altro prof in caso sia una mat di laboratorio.....
	uint8_t ore_rim;
	int laboratorio;
	uint8_t id;
	uint8_t id_materia;
};
struct _prof {
//	uint32_t ore_tot;
	uint32_t ore_rim;
	uint32_t num_materie;
	uint32_t teorico;
	uint32_t num_classi;
	uint32_t num_classi_rim;	// seRVE ALLA STRUTTURA DAI
	struct _classe_prof *classi;
	//uint32_t giorno_libero;
};
struct _info_classe {
	uint32_t ore;
	uint32_t prof;
	int laboratorio;
};
struct _classe{
	uint32_t orario[GIORNI*OREGIORNO];
};
```

```
struct global_frequency {
	uint32_t classe;
	int ore_rim;
	int lab;
	int prof;
};
struct local_frequency {
	uint32_t prof;
	int ore_rim;
	int lab;
	//uint32_t prof2;
	uint32_t materia;
//	uint32_t ora;
};
typedef struct global_frequency gf;
typedef struct local_frequency lf;
typedef struct _prof prof;
typedef struct _classe classe;
typedef struct _info_classe infoclasse;
uint8_t **_Classi,*Disponibilita,*Displab;
uint32_t nmat,nprof,nclas,nlab,ncattedre;
char **_Corrispondenze[4]; //  [Class/Mat/Prof/Lab][id]="Stringa corrispondente all'id"
uint8_t *materie; // 0 se Teorica , 1 se laboratorio.

infoclasse ***Iclassi; // [classe][materia][0].ora -> Quanti prof insegnano quella materia in quella classe
                       // [classe][materia]([1]: [1+[classe][materia][0].ora]) -> info sui prof e le ore rimanenti
                       // insegnare
classe *classi;
prof *professori;
#ifdef CPP
	bool operator < (const gf &a, const gf &b) {
		return a.ore_rim < b.ore_rim;
	}
	bool operator < (const lf &a, const lf &b) {
		/*if (a.ora < b.ora )
			return false;
		else if ( a.ora > b.ora)
			return true;
		*/ return a.ore_rim < b.ore_rim;
	}

	priority_queue <lf> *Frequenze_locali;
	priority_queue <gf> Frequenze_globali;
#else
	#include "Heap.h"
	PQ *Frequenze_locali,*Frequenze_globali;
	int maxlf(void *a1 ,void *b1) {
		if (b1==NULL )
			return 1;
		if (a1==NULL )
			return 0;
		lf a=*(lf *) a1;
		lf b=*(lf *) b1;
		if (a.ore_rim >0 ){
			if (b.ore_rim >0){
				if (a.lab==1 && b.lab!=1)
					return 1;
				if (a.lab!=1 && b.lab==1)
					return 0;
				if (professori[a.prof].ore_rim == professori[b.prof].ore_rim)
					return a.ore_rim> b.ore_rim;
```

```
                                        else
                                                return professori[a.prof].ore_rim > professori[b.prof].ore_rim;
                                } else
                                        return 1;
                        }
                        return 0;

                }
                int maxgf (void *a1 , void *b1) {
                        if (b1==NULL )
                                return 1;
                        if (a1==NULL )
                                return 0;
                        gf a=*(gf *) a1;
                        gf b=*(gf *) b1;
                        if (a.ore_rim >0 ){
                                if (b.ore_rim >0){
                                        if (a.lab==1 && b.lab!=1)
                                                return 1;
                                        if (a.lab!=1 && b.lab==1)
                                                return 0;
                                        // ATTENZIONE QUI è sbagliato Il problema nasce nella scelta di due professori che
                                        //hanno la stesse ore rimanenti .. o meglio ancora che sono stessi prof... qui
                                        //sarebbe da agire non sull'heap ma sull'algoritmo inserendo in qualche modo la
                                        //possibilita di tornare indietro e scegliere l'ora di un altro prof quando da
                                        //errore...
                                        if (professori[a.prof].ore_rim == professori[b.prof].ore_rim)
                                                return a.ore_rim> b.ore_rim;
                                        else
                                                return professori[a.prof].ore_rim > professori[b.prof].ore_rim;
                                } else
                                        return 1;
                        }
                        return 0;
}
#endif

void rebuildfrequency () {
        uint32_t i;
        lf *tmp;
        tmp=(lf *) malloc(sizeof(lf));
        gf *temp=(gf *) malloc(sizeof(gf));
        for (i=0;i<nclas-2;i++) {

                temp->classe=i;
                #ifdef CPP
                        (*tmp)=Frequenze_locali[i].top();
                        temp->ore_rim=tmp->ore_rim;
                        Frequenze_globali.push(*temp);
                #else
                        tmp=(lf *)_top(&(Frequenze_locali[i]));
                        temp->ore_rim=tmp->ore_rim;
                        temp->lab=tmp->lab;
                        temp->prof=tmp->prof;
                        _push(Frequenze_globali,temp,maxgf,sizeof(*temp));
                #endif
        }

}
int correggi (uint32_t prof,uint32_t mat, uint32_t cl,uint8_t *cA ,uint8_t *mA,uint32_t stop,uint32_t ora) {
        uint32_t i,j;
```

```
lf *tmp;
uint32_t indice,corretto,trovato;
PQ *tmpplf=creaHeap(1,nprof);
indice=-1;
// Cerca del prof criminante nelle altre classi
for (i=0;i<stop && indice==-1;i++) {
        for (j=0;j<Iclassi[cA[i]][classi[cA[i]].orario[ora]][0].ore && indice==-1;j++)
                if (Iclassi[ cA[i] ][ classi[ cA[i] ].orario[ora]][j+1].prof == prof)
                        indice=i;
}

// Scorro tutta la pq della classe che detiene il prof e ne cerco uno libero.
// Se trovo il prof allora ritorno allla situazione in cui non lo ho messo qui
corretto=trovato=0;
while (_size(&(Frequenze_locali[cA[indice]]) ) >0 ) {
        tmp=(lf *)_pop(&(Frequenze_locali[cA[indice]]),maxlf);
        if (tmp->prof==prof) {
                tmp->ore_rim=tmp->ore_rim+1;
                (professori[tmp->prof].ore_rim)++;
                _push(tmpplf,tmp,maxlf,sizeof(*tmp));
                trovato=1;
        } else if (Disponibilita[tmp->prof]!=1) {
                _push(tmpplf,tmp,maxlf,sizeof(*tmp));
        } else if (!corretto){
                // se non è gia stato corretto ed è disponibile ed non è il prof in questione
                (professori[tmp->prof].ore_rim)--;
                classi[cA[indice]].orario[ora]=tmp->materia;;
                tmp->ore_rim=tmp->ore_rim-1;
                classi[cl].orario[ora]=mat;
                _push(&(Frequenze_locali[cA[indice]]),tmp,maxlf,sizeof(lf));
                corretto=1;
        } else
                _push(tmpplf,tmp,maxlf,sizeof(*tmp));
}
if (corretto==1) {
        // Aggiorno la pq sapendo che la _pop mi dara lf * che rompe i coglioni.
        tmp=(lf *)_pop(&(Frequenze_locali[cl]),maxlf);
        tmp->ore_rim=tmp->ore_rim-1;
        (professori[tmp->prof].ore_rim)--;
        _push(&(Frequenze_locali[cl]),tmp,maxlf,sizeof(*tmp));
        // Se non ha trovato il vecchio prof vuol dire che ha finito le ore e bisogna creare un lf*
        if (trovato==0) {
                tmp=(lf *) malloc(sizeof(lf));
                tmp->prof=prof;
                tmp->ore_rim=1;
                tmp->lab=materie[mA[indice]];
                tmp->materia=mA[indice];
                _push(tmpplf,tmp,maxlf,sizeof(*tmp));
        }
}
// Ripopolo Frequenze_locali della classe che detiene il professore bloccato
while(_size(tmpplf)>0)
        _push(&(Frequenze_locali[cA[indice]]),_pop(tmpplf,maxlf),maxlf,sizeof(lf));
// Se non è stato corretto provo a fare la stessa roba
if (corretto==0) {
        while (_size(&(Frequenze_locali[cA[indice]]) ) >0 ) {
                tmp=(lf *)_pop(&(Frequenze_locali[cA[indice]]),maxlf);
                if (Disponibilita[tmp->prof]!=1 && tmp->prof!=prof) {
                        correggi(tmp->prof,tmp->materia,cA[indice],cA,mA,stop,ora);
                        _push(tmpplf,tmp,maxlf,sizeof(*tmpplf));
                }
```

```
                }
                while(_size(tmpplf)>0)
                _push(&(Frequenze_locali[cA[indice]]),_pop(tmpplf,maxlf),maxlf,sizeof(lf));
        }

        if (corretto==1)
                return 1;
        else
                return 0;

}
void creaorario () {
        int j,max,x;
        #ifdef CPP
                priority_queue <lf> tmpplf;
        #else
                PQ *tmpplf=creaHeap(1,nprof);
        #endif
        uint8_t *classiAssegnate=(uint8_t *) malloc((nclas-2)*sizeof(uint8_t));
        uint8_t *matAss=(uint8_t *) malloc((nclas-2)*sizeof(uint8_t));
        gf *temp=(gf *) malloc(sizeof(gf));
        lf *templf=(lf *) malloc(sizeof(lf));
        uint32_t i,z,boolean=1,ok;
        for (i=0;i<OREGIORNO*GIORNI;i++) {
                memset (Disponibilita,1,nprof*sizeof(Disponibilita[0]));
                memset (Displab,1,nlab*sizeof(Displab[0]));
                memset (matAss,0,(nclas-2)*sizeof(uint8_t));
                memset (classiAssegnate,0,(nclas-2)*sizeof(uint8_t));
                        for (j=0;j<nclas-2;j++) {
                                #ifdef CPP
                                        *temp=Frequenze_globali.top();
                                        Frequenze_globali.pop();
                                #else
                                        temp=(gf *)_pop(Frequenze_globali,maxgf);
                                #endif
                                classiAssegnate[j]=temp->classe;
                                boolean=x=0;
                                do {
                                        #ifdef CPP
                                                *templf=Frequenze_locali[temp->classe].top();
                                                Frequenze_locali[temp->classe].pop();
                                        #else
                                                templf=(lf *)_pop(&(Frequenze_locali[temp->classe]),maxlf);
                                        #endif
                                        ok=0;
                                        // Se è una materia non teorica allora verifico che tutti i professori
                                        // Che insegnano in quell'ora siano liberi.
                                        if (materie[templf->materia]==1 ){
                                                if (Displab[Iclassi[temp->classe][templf->materia][1].laboratorio]==1){
                                                        ok=1;
                                                        if (Iclassi[temp->classe][templf->materia][0].ore>1 &&
Disponibilita[Iclassi[temp->classe][templf->materia][2].prof]!=1)
                                                                ok=2;

                                                } else ok=2;
                                        }
                                        // se tutto è ok allora assegno il prof all'ora ... e aggiorno la pq
                                        if (Disponibilita[templf->prof]==1 && (ok==0 || ok==1) ) {

                                                if (materie[templf->materia])
                                                        Displab[Iclassi[temp->classe][templf->materia][1].laboratorio]=0;
```

```
                                                max=1;
                                                for (z=0;z<Iclassi[temp->classe][templf->materia][0].ore;z++) {
                                                        (Iclassi[temp->classe][templf->materia][z+1].ore)--;
                                                        Disponibilita[Iclassi[temp->classe][templf->materia][z+1].prof]=0;
                                                        (professori[Iclassi[temp->classe][templf-
>materia][z+1].prof].ore_rim)--;
                                                        if ((professori[Iclassi[temp->classe][templf-
>materia][z+1].prof].ore_rim)>professori[max].ore_rim)
                                                                max=z+1;
                                                }
                                                classi[temp->classe].orario[i]=templf->materia;
                                                matAss[j]=templf->materia;
                                                templf->prof=Iclassi[temp->classe][templf->materia][max].prof;
                                                templf->ore_rim=Iclassi[temp->classe][templf->materia][max].ore;
                                                boolean=1;
                                                #ifdef CPP
//                                                      printf("2Ore rim = %d\ n",templf->ore_rim);
                                                        Frequenze_locali[temp->classe].push(*templf);
                                                #else
                                                        if (templf->ore_rim>0)
                                                                _push(&(Frequenze_locali[temp-
>classe]),templf,maxlf,sizeof(*templf));
                                                #endif
                                        } else{
                                                #ifdef CPP
                                                        tmpplf.push(*templf);
                                                #else
                                                        _push(tmpplf,templf,maxlf,sizeof(*templf));
                                                #endif
                                        }
                                        x++;
                                } while (boolean==0 &&          _size(&(Frequenze_locali[temp->classe]))>0);
                                // sE NN è andato bene allora reimpienishi la localfrequency con i prof non usati.
                                while (     _size(tmpplf)>0){
                                        #ifdef CPP
                                                Frequenze_locali[temp->classe].push(tmpplf.top());
                                                tmpplf.pop();
                                        #else
                                                _push(&(Frequenze_locali[temp-
>classe]),_pop(tmpplf,maxlf),maxlf,sizeof(lf));
                                        #endif
                                }
                                if (boolean==0){
                                        templf=(lf *)_top(&(Frequenze_locali[temp->classe]));
                                        if (!correggi(templf->prof,templf->materia,temp->classe,classiAssegnate,matAss,j,i))
                                                printf("Problema ora %d %d %d %d\ n",i,x,j,temp->classe);
                                }
                        }

                rebuildfrequency();
        }
}
void stampaorario() {
        uint32_t z,i,j;
        FILE *fw=fopen("orario.html","w");
        fprintf(fw,"<html><body><center>\ n");
        int appoggio[nprof][36];
        for (i=0;i<nprof;i++)
                memset(appoggio[i],-1,36*sizeof(int));
        for (i=0; i<nclas-2; i++)
                for (j=0; j<36; j++)
```

```
                        for (z=0; z<(Iclassi[i][classi[i].orario[j]][0].ore); z++){
                                if (appoggio[Iclassi[i][classi[i].orario[j]][z+1].prof][j]!=-1)
                                        printf ("porcodio\ n");
                                appoggio[Iclassi[i][classi[i].orario[j]][z+1].prof][j]=i;
                        }
        for (z=0;z<3;z++){
                fprintf(fw,"<table border=2>\ n");
                for (i=0;i<nclas-2;i++) {
                        fprintf(fw,"<tr><td> Classe %s </td>",_Corrispondenze[CLASSI][i]);
                        for (j=0;j<OREGIORNO*GIORNI;j++) switch (z) {
                                case 0:  fprintf(fw,"<td
bgcolor=#%s>%s</td>",i%2?(j/6)%2?"ffffff":"dddddd":(j/6)%2?"c0c0c0":"969696",_Corrispondenze[MATERIE][classi[i].orario[j]]);
                                        break;
                                case 1: fprintf(fw,"<td
bgcolor=#%s>%s</td>",i%2?(j/6)%2?"ffffff":"dddddd":(j/6)%2?"c0c0c0":"969696",_Corrispondenze[PROF][Iclassi[i][classi[i].orario
[j]][1].prof]);
                                        break;
                                case 2: fprintf(fw,"<td
bgcolor=#%s>%s</td>",i%2?(j/6)%2?"ffffff":"dddddd":(j/6)%2?"c0c0c0":"969696",materie[classi[i].orario[j]]==1?_Corrispondenze[
LABORATORI][Iclassi[i][classi[i].orario[j]][1].laboratorio]:"---");
                                        break;
                        }

                        fprintf(fw,"</tr>\ n");
        /*              for (j=0;j<nprof;j++)
                                printf("|%d",A[j]);
                        printf("|\ n");
        */      }
                fprintf(fw,"</table>");
        }
        fprintf(fw,"<table border=2><tr><td> PROF</td>");
        for (i=0;i<36;i++)
                fprintf(fw,"<td> %d </td>",i);
        fprintf(fw,"</tr>");
        for (i=0;i<nprof;i++) {
                fprintf(fw,"<tr><td>%s</td>",_Corrispondenze[PROF][i]);
                for (j=0;j<36;j++) {
                        fprintf (fw,"<td>%s</td>",appoggio[i][j]==-1?"---":_Corrispondenze[CLASSI][appoggio[i][j]]);
                }
                fprintf(fw,"</tr>");
        }
                fprintf(fw,"</table></body></html>");
}
void lleggi(char **fbuf,uint32_t n,char ***sol,uint32_t *fbufsize) {
        int i;
        *fbufsize=*fbufsize-(strlen(*fbuf)+1);
        (*fbuf)=&fbuf[0][strlen(*fbuf)+1];
        sol[0]=(char **) malloc(n*sizeof(char *));
        // LEGGO Le materie
        for (i=0;i<n;i++) {
                sol[0][i]=(char *) malloc((strlen(*fbuf)+1)*sizeof(char));
                memcpy(sol[0][i],*fbuf,(strlen(*fbuf)+1)*sizeof(char));
                *fbufsize=*fbufsize-(strlen(*fbuf)+1);
                *fbuf=&fbuf[0][strlen(*fbuf)+1];
        }

//      for (i=0;i<n;i++)
//              printf("[%d] = \ "%s\ "\ n",i,sol[i]);
}

int cerca (char **buf,uint32_t n,char *pattern) {
```

```
	int i;
	for (i=0;i<n;i++)
		if (strlen(buf[i])==strlen(pattern) && memcmp(buf[i],pattern,strlen(pattern))==0)
			return i;
	printf("Not Found %s \ n",pattern);
	return -1;
}


void leggiorario (char *file) {
	FILE *fr;
	int fbufsize;
	infoclasse *tinfo;
	uint32_t i,tmph,tmpm2,tmpc2,tmpp2,j,trovato;
	char tmpm[BUFSIZE],tmpp[BUFSIZE],tmpc[BUFSIZE],tmpl[BUFSIZE];
	struct _classe_prof *cla;
	struct stat buf;
	char *fbuf;
	if (stat(file,&buf))
		printf("Errore nella stat\ n");
	// Leggo tutto in fbuf cosi evito di fare tanti accessi al disco
	fbuf=(char *) malloc(buf.st_size*sizeof(char));
	fr=fopen(file,"r");
	fbufsize=buf.st_size;
	fread(fbuf,sizeof(char),buf.st_size,fr);
	fclose(fr);

	// Cambio i \ t e i \ n in \ 0 in modo da poter condiderare un array di stringhe da far esplodere.
	for (i=0;i<buf.st_size;i++)
		if (fbuf[i]=='\ t' || fbuf[i]=='\ n')
			fbuf[i]='\ 0';

	// Comincio la lettura dei dati dal file
	sscanf(fbuf,"%d",&nmat);
	lleggi(&fbuf,nmat,&_Corrispondenze[MATERIE],&fbufsize);
	materie=(uint8_t *) malloc (nmat *sizeof(uint8_t));
	// Marco le materie come teoriche o meno.
	for (i=0;i<nmat;i++)
		if (_Corrispondenze[MATERIE][i][0]=='L' || strcmp(_Corrispondenze[MATERIE][i],"Ed.Fis")==0)
			materie[i]=1;
		else
			materie[i]=0;

	sscanf(fbuf,"%d",&nprof);
	lleggi(&fbuf,nprof,&_Corrispondenze[PROF],&fbufsize);

	sscanf(fbuf,"%d",&nclas);
	lleggi(&fbuf,nclas,&_Corrispondenze[CLASSI],&fbufsize);

	sscanf(fbuf,"%d",&nlab);
	lleggi(&fbuf,nlab,&_Corrispondenze[LABORATORI],&fbufsize);

	professori=(prof *) malloc ( nprof *(sizeof(prof)));
	classi=(classe *) malloc(nclas * (sizeof(classe)));
	Iclassi=(infoclasse ***) malloc(nclas*sizeof(infoclasse **));
	Disponibilita=(uint8_t *) malloc (nprof*sizeof(uint8_t));
	Displab=(uint8_t *) malloc(nlab*sizeof(uint8_t));

	for (i=0;i<nprof;i++) {
		professori[i].ore_rim=0;
		professori[i].num_materie=0;
```

```
                professori[i].teorico=0;
                professori[i].num_classi=0;
                professori[i].num_classi_rim=0;
        }

        for (i=0;i<nclas;i++) {
                Iclassi[i]=(infoclasse **) malloc(nmat*sizeof(infoclasse *));
                for (j=0;j<nmat;j++) {
                        Iclassi[i][j]=(infoclasse *) malloc(sizeof(infoclasse));
                        Iclassi[i][j][0].ore=0; // numero materie
                }

        }
        // Lettura dei Dati veri e propri
        for (i=0;fbufsize>0;i++) {
                        // LEttura H

                sscanf(fbuf,"%d",&tmph);
                        // lettura materia
                fbuf=&fbuf[strlen(fbuf)+1];
                fbufsize-=(strlen(fbuf)+1);
                sscanf(fbuf,"%s",tmpm);
                        // lettura Professore
                fbuf=&fbuf[strlen(fbuf)+1];
                fbufsize-=(strlen(fbuf)+1);
                memcpy(tmpp,fbuf,strlen(fbuf)+1);
                // Lettura Classe
                strcpy(tmpc,"ciao");
                fbuf=&fbuf[strlen(fbuf)+1];
                fbufsize-=(strlen(fbuf)+1);
                sscanf(fbuf,"%s",tmpc);
                        // Se è una materia di laboratorio allora leggi il laboratorio dove metto sti disperati
                tmpm2=cerca(_Corrispondenze[MATERIE],nmat,tmpm);
                if ( materie[tmpm2]==1 || !(fbuf[strlen(fbuf)+1]>='0' && fbuf[strlen(fbuf)+1]<='9')) {
                        fbuf=&fbuf[strlen(fbuf)+1];
                        fbufsize-=(strlen(fbuf)+1);
                        sscanf(fbuf,"%s",tmpl);
                }
                tmpp2=cerca(_Corrispondenze[PROF],nprof,tmpp);
                tmpc2=cerca(_Corrispondenze[CLASSI],nclas,tmpc);
                trovato=0;
                // Se è gia stat ainserita la classse e la materia specificata allora aggiunge semplicemente ore
                if (tmpc2 <nclas-2) {
                        for (j=0;j<professori[tmpp2].num_classi && !trovato;j++)
                                if (professori[tmpp2].classi[j].id_materia== tmpm2 && professori[tmpp2].classi[j].id == tmpc2){
                                        professori[tmpp2].classi[j].ore_rim+=tmph;
                                        (professori[tmpp2].ore_rim)+=tmph;
                                        trovato=1;
                                }
                        if (!trovato) {
                                professori[tmpp2].num_classi++;
                                (professori[tmpp2].num_classi_rim)++;
                                cla=(struct _classe_prof *)malloc(professori[tmpp2].num_classi*sizeof(struct _classe_prof));
                                for (j=0;j<professori[tmpp2].num_classi-1;j++)
                                        memcpy(&cla[j],&(professori[tmpp2].classi[j]),sizeof(struct _classe_prof));
                                professori[tmpp2].classi=cla;
                                (professori[tmpp2].ore_rim)+=tmph;
                                professori[tmpp2].classi[professori[tmpp2].num_classi-1].ore_rim=tmph;
                                professori[tmpp2].classi[professori[tmpp2].num_classi-
1].id=cerca(_Corrispondenze[CLASSI],nclas,tmpc);
                                professori[tmpp2].classi[professori[tmpp2].num_classi-1].id_materia=tmpm2;
```

```
                                        if ( materie[tmpm2]==1 )
                                                professori[tmpp2].classi[professori[tmpp2].num_classi-
1].laboratorio=cerca(_Corrispondenze[LABORATORI],nlab,tmpl);
                                        else
                                                professori[tmpp2].classi[professori[tmpp2].num_classi-1].laboratorio=-1;
                                }
                                trovato=0;
                                for (j=0;j<Iclassi[tmpc2][tmpm2][0].ore && !trovato;j++)
                                        if (Iclassi[tmpc2][tmpm2][j+1].prof==tmpp2)
                                                trovato=j+1;
                                if (!trovato) {
                                        (Iclassi[tmpc2][tmpm2][0].ore)++;
                                        tinfo=(infoclasse *) malloc((Iclassi[tmpc2][tmpm2][0].ore+1)*sizeof(infoclasse));
                                        memcpy(tinfo,Iclassi[tmpc2][tmpm2],(Iclassi[tmpc2][tmpm2][0].ore)*sizeof(infoclasse));
                                        Iclassi[tmpc2][tmpm2]=tinfo;
                                        Iclassi[tmpc2][tmpm2][Iclassi[tmpc2][tmpm2][0].ore].prof=tmpp2;
                                        trovato=Iclassi[tmpc2][tmpm2][0].ore;
                                        Iclassi[tmpc2][tmpm2][trovato].laboratorio=-1;
                                }

                                Iclassi[tmpc2][tmpm2][trovato].ore +=tmph;
                                if (materie[tmpm2]==1)
                                        Iclassi[tmpc2][tmpm2][trovato].laboratorio=cerca(_Corrispondenze[LABORATORI],nlab,tmpl);
                        }
                                fbuf=&fbuf[strlen(fbuf)+1];
                                fbufsize-=(strlen(fbuf)+1);

        }
        ncattedre=0;
        for (i=0;i<nprof;i++)
                ncattedre+=professori[i].num_classi;
        //Comincio a costruire i priority queue per le classi e poi alla fine una priority queue globale
        Frequenze_locali=creaHeap(nclas,nprof+1);
        Frequenze_globali=creaHeap(1,nclas+1);
        lf *lftmp;
        for (i=0;i<nclas-2;i++)
                for (j=0;j<nmat;j++) {
                        if (Iclassi[i][j][0].ore >0) {
                                lftmp=(lf *) malloc(sizeof(lf));
                                lftmp->prof=Iclassi[i][j][1].prof;
                                lftmp->materia=j;
                                if (materie[j]==1){
                                        lftmp->lab=1;
                                        if (Iclassi[i][j][0].ore>1) {
                                                if (professori[Iclassi[i][j][2].prof].ore_rim >
professori[Iclassi[i][j][1].prof].ore_rim)
                                                        lftmp->prof=Iclassi[i][j][2].prof;
                                        }
                                } else
                                        lftmp->lab=0;
                                (lftmp->ore_rim)=Iclassi[i][j][1].ore;
                                _push(&(Frequenze_locali[i]),lftmp,maxlf,sizeof(*lftmp ));
                        }
                }
        gf *gftmp=(gf *) malloc(sizeof(gf));
        for (i=0;i<nclas-2;i++) {
                gftmp->classe=i;
                #ifdef CPP
                        (*tmp)=Frequenze_locali[i].top();
                        temp->ore_rim=tmp->ore_rim;
                        Frequenze_globali.push(*temp);
```

```
		#else
			lftmp=(lf *) _top(&(Frequenze_locali[i]));
			gftmp->ore_rim=lftmp->ore_rim;
			gftmp->lab=lftmp->lab;
			gftmp->prof=lftmp->prof;
			_push(Frequenze_globali,gftmp,maxgf,sizeof(*gftmp));
		#endif
	}


}

void stampastrutturadatiorario(char *nomefile ) {
	FILE *fw;
	int i,j,z,tmp;
	fw=fopen(nomefile,"w");
	// stampa classi:
	if (nmat>nprof)
		tmp=nmat;
	else
		tmp=nprof;
	if (tmp<nlab)
		tmp=nlab;
	if (tmp<nclas)
		tmp=nclas;

	fprintf (fw,"<html>\ n\ t<head><title> Debug Mode on by Veke </title></head><body>\ n ");
	fprintf (fw,"<center><h1> OCCORRENZE </h1> \ n");
	fprintf (fw,"<table BORDER=1><tr
bgcolor=orange><td>Id</td><td>Materie</td><td>Prof</td><td>CLassi</td><td>Laboratori</td></tr>\ n");
	for (i=0;i<tmp;i++){
		fprintf (fw,"<tr><td bgcolor=orange>%d</td>",i);
		fprintf (fw,"<td>%s</td>",i<nmat?_Corrispondenze[MATERIE][i]:" ");
		fprintf (fw,"<td>%s</td>",i<nprof?_Corrispondenze[PROF][i]:" ");
		fprintf (fw,"<td>%s</td>",i<nclas?_Corrispondenze[CLASSI][i]:" ");
		fprintf (fw,"<td>%s</td>",i<nlab?_Corrispondenze[LABORATORI][i]:" ");
		fprintf (fw,"</tr>");
	}

	fprintf (fw,"</table><h1> Professori e le loro classi </h1>");
	tmp=0;
	for (j=0;j<nprof;j++)
		tmp=tmp<professori[j].num_classi?professori[j].num_classi:tmp;

	fprintf (fw,"<table BORDER=1><tr bgcolor=orange><td>NomePROF</td><td>Num CLassi</td><td>Teorico</td><td>Num
MAterie</td>\ n");
	for (j=0;j<tmp;j++)
			fprintf(fw,"<td>Clas %d</td><td>Ore %d </td><td>Mat %d</td><td> Lab %d</td>\ n",j+1,j+1,j+1,j+1);
	fprintf (fw,"</tr>");

	for (j=0;j<nprof;j++) {
		fprintf(fw,"<tr
bgcolor=#%s><td>%s</td><td>%d</td><td>%s</td><td>%d</td>",j%2?"ffffff":"c0c0c0",_Corrispondenze[PROF][j],professori[j].num_cl
assi,professori[j].teorico==1?"Si":"No",professori[j].num_materie);


		for(z=0;z<professori[j].num_classi;z++)
			fprintf(fw,"<td bgcolor=#%s>%s</td><td bgcolor=#%s>%d</td><td bgcolor=#%s>%s</td><td
bgcolor=#%s>%s</td>\ n",j%2?z%2?"dddddd":"ffffff":z%2?"969696":"c0c0c0",_Corrispondenze[CLASSI][professori[j].classi[z].id],j
%2?z%2?"dddddd":"ffffff":z%2?"969696":"c0c0c0",professori[j].classi[z].ore_rim,j%2?z%2?"dddddd":"ffffff":z%2?"969696":"c0
c0c0",_Corrispondenze[MATERIE][professori[j].classi[z].id_materia],j%2?z%2?"dddddd":"ffffff":z%2?"969696":"c0c0c0",professo
```

```
ri[j].classi[z].laboratorio!=-1?_Corrispondenze[LABORATORI][professori[j].classi[z].laboratorio]:"--");
                fprintf (fw,"</tr>");
        }

        fprintf (fw,"</table><h1> CLASSI </h1><table>");
        for (i=0;i<nclas;i++) {
                if (i%4==0)
                        fprintf(fw,"<tr>");
                fprintf (fw,"<td>");
                fprintf(fw,"CLASSE \ "%s\ "",_Corrispondenze[CLASSI][i]);
                fprintf (fw,"<table BORDER=1><tr bgcolor=orange><td>Materia</td><td>Professore</td><td>Ore
Rim</td><td>Laboratorio</td></tr>\ n");
                for (j=0;j<nmat;j++)
                        for (z=0;z<Iclassi[i][j][0].ore;z++)
                                fprintf(fw,"<tr>
<td>%s</td><td>%s</td><td>%d</td><td>%s</td></tr>\ n",_Corrispondenze[MATERIE][j],_Corrispondenze[PROF][Iclassi[i][j][z+1].prof]
,Iclassi[i][j][z+1].ore,Iclassi[i][j][z+1].laboratorio==-1?"---":_Corrispondenze[LABORATORI][Iclassi[i][j][z+1].laboratorio]);

                fprintf(fw,"</table></td>");
                if (i-2%4==0)
                        fprintf(fw,"</tr>");


        }
        fprintf (fw,"</table></body></html>");
        fclose (fw);
}
```

Fine orario.h; Inizio porting.h

```
#include <limits.h>
#define LAB_GENERIC 1

void porting (tab_classi *clas, tab_prof *orprof, tab_dati_prof *dati_prof, tab_dati_cattedre *dati_cattedre) {
        int i,j,z;

        int contnumcattedra = 0;
        int num_catt_curr_prof;

        // Carica la dati_cattedre
        for (i=0; i<nprof; i++) {
                num_catt_curr_prof=professori[i].num_classi;
                // dovrebbe essere ok ma bisogna includerlo nel ciclo di controlli che parte da riga 26
                for (j=0; j<num_catt_curr_prof; j++) {
                        (*dati_cattedre)[contnumcattedra].prof = i;
                        (*dati_cattedre)[contnumcattedra].classe = professori[i].classi[j].id;
                        (*dati_cattedre)[contnumcattedra].materia = professori[i].classi[j].id_materia;
                        (*dati_cattedre)[contnumcattedra].max_ore_consec = INT_MAX;
                        contnumcattedra++;
                }

        }

        int ncattedra (int classe, int materia, int prof) {
                int x;
                for (x=0;x<ncattedre;x++)
                        if ((*dati_cattedre)[x].prof==prof && (*dati_cattedre)[x].classe==classe &&
(*dati_cattedre)[x].materia==materia)
                                return x;
                return -1;
```

```
	}
	// Carica tab prof
	for (i=0; i<nclas-2; i++)
		for (j=0; j<ORE; j++)
			for (z=0; z<(Iclassi[i][classi[i].orario[j]][0].ore); z++){
				// Se è un laboratorio alora in prof[ idprof ] .lab [ ora ] metto l'id del laboratorio +1 altrimenti
metto 0
				if (materie[classi[i].orario[j]] == LAB_GENERIC)
					(*orprof)[ Iclassi[i][classi[i].orario[j] ] [ z+1 ].prof ].lab[j] =
Iclassi[i][classi[i].orario[j]][z+1].laboratorio+1;
				else
					(*orprof)[Iclassi[i][classi[i].orario[j]][z+1].prof].lab[j] = 0;
				// Bisogna controllare che non ritorni -1
				(*orprof)[Iclassi[i][classi[i].orario[j]][z+1].prof].cat[j] =
ncattedra(i,classi[i].orario[j],Iclassi[i][classi[i].orario[j]][z+1].prof);
			}
	// Carica tab _classi
	for (i=0; i<nclas-2; i++) {
		for (j=0; j<ORE; j++) {
			// Se è un laboratorio allora in classi[ idclasse ] .lab[ ora ] metto l'idi del laboratorio +1 altrimenti metto
0
			if (materie[classi[i].orario[j]] == LAB_GENERIC)
				(*clas)[i].lab[j] = (Iclassi[i][classi[i].orario[j]][1].laboratorio)+1;
			else
				(*clas)[i].lab[j] = 0;
			// metto l'id della cattedra corrispondente ai valori ....
			(*clas)[i].cat0[j] = ncattedra(i,classi[i].orario[j],Iclassi[i][classi[i].orario[j]][1].prof);
			if (Iclassi[i][classi[i].orario[j]][0].ore>1)
				(*clas)[i].cat1[j] = ncattedra(i,classi[i].orario[j],Iclassi[i][classi[i].orario[j]][2].prof);
		}
	}
}
```

Fine porting.h ; Inizio datatypev.h

```
#include <mpi.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include "Stack.h"
#include "lf_rand.h"
#ifdef FORKIAMO
	#include <sys/types.h>
	#include <unistd.h>
#endif
#include <signal.h>

#ifdef GNUPLOT
	#define MAXQUALITY 20000000
	#define NMEDIA 20
#endif
#ifndef M_E
	#define M_E		2.7182818284590452354
#endif

	// controllo fondamentale programma:
	// ATTENZIONE: questi valori modificano radicalmente il funzionamento della tempratura
#define SINGLE_VALORE_STALLO 50		// numero di iterazioni che si accettano senza che la qualità migliori (con
```

```
scambi singoli)
#define MULTIPLE_VALORE_STALLO 50       // numero di iterazioni che si accettano senza che la qualità migliori (con scambi
multipli)
#define EXTERNAL_LOOP 110000            // numero di cicli esterni di reannealing
#define MAX_PROFONDITA 80               // numero massimo di scambi in catena
#define ZOMPA_DOMINIO 3000              // numero di scambi effettuati per spostarsi nel dominio
#define STALLO_EXT_MAX 50
#define STEP 0.999995
#define INIZIALE 0.6
#define ESPONENTE 0.17
#define EPSILON 0.004
#define NMQUALITY 1500
        /*      Costanti da passare al compilatore
                per attivare i vari tipi di debug:
#define SCREEN_DEBUG    // debug a schermo (occhio ai numeri alti nei loop principali)
#define FILE_DEBUG      // debug su file html - orario iniziale e finale
# define STAT_DEBUG     // statistiche programma - influenzano il ciclo principale
#define FINAL_DEBUG             // stampa su file il resoconto dell'esecuzione anzichè stamparlo a schermo
#define GNUPLOT                 // gnuplot su file
#define FORKIAMO
*/
        // Materie:
#define ORA_DISPOSIZIONE -1

        // Laboratori:
#define LAB_ORA_INTOCCABILE -1          // utilizzi: - prof con problemi di orario entrata/uscita
                                                //      - prof part-time
                                                //      - "lucchetto" nella rifinitura
dell'orario
#define LAB_NONE 0                      // nessun lab -> lezione in class

#define ORE     36   // numero di ore nella settimana
#define GIORNI  6   // numero di giorni di scuola alla settimana
#define ORE_GIORNALIERE 6  // numero di ore di scuola al giorno

#include "orario.h"

struct classi_st {
        int lab[ORE];
        int cat0[ORE];
        int cat1[ORE];
};

struct prof_st {
        int lab[ORE];
        int cat[ORE];
};
struct catt_st {
        int prof;
        int classe;
        int materia;
        //int lab;
        int max_ore_consec;
        int marked;
};
   // matrici orario classi e orario professori
   // esempio di uso delle costanti: orario_classi[codice_classe].lab[codice_ora]
typedef struct classi_st *tab_classi;
typedef struct prof_st *tab_prof;
        // matrici dati cattedre e professori
        // tra i seguenti parametri dei professori non ci sono obblighi,
```

```
        // solo preferenze con un peso variabile arbitrariamente
typedef int **tab_dati_prof;
#define ID_TDP_PESO_GIORNO_LIB 0        // peso preferenza giorno libero
#define ID_TDP_GIORNO_LIB  1    // giorno libero preferito
#define ID_TDP_TEORICO  2       // flag teorico/non teorico


#define ID_PROF_TEORICO 1
#define ID_PROF_NON_TEORICO 0

        // dati singole materie:  - prof che insegna una certa materia
        //                        - max ore consecutive quella materia
typedef struct catt_st  *tab_dati_cattedre;
#define CAT_MARCATA 1
#define CAT_NON_MARCATA 0

        // costanti per controllo qualità
#define MAT_ORA_PREC_PRESENTE 1
#define MAT_ORA_PREC_NON_PRESENTE 0

        // costanti di utilita` comune
#define NO_LAB 0    // e` un'ora di lezione normale
#define PROF_LIBERO -1  // ora libera

#define RIUSCITO 0  // scambio riuscito
#define FALLITO 1   // scambio non effettuato

#define GIORNO_PREC_PRESENTE 1
#define GIORNO_PREC_NON_PRESENTE 0

#define VERO 0
#define FALSO 1

        //costanti di peso per la bontà
// ogni ora giornaliera > 4 per gli insegnanti
#define O_G_SUP4            3
// ogni giorno di insegnamento < 5
#define N_G_I_M5            8
// ogni ora in più giornaliera di materie tipo inglese
#define N_O_G_SUP1          5
// ogni ora nel giorno successivo per materie tipo inglese
#define N_O_G_SUCC          2
// ogni ora consecutiva di materia tipo matematica che non arrivi a max_ore_consec
#define N_O_NCONSEC     2
// ogni ora in più in un giorno di materia tipo matematica
#define N_O_G_SUP           2
// ogni ora giornaliera nel giorno dopo avere avuto max_ore_consec
#define N_O_G_AFT_MAXCONSEC 1
// ogni ora > 1 in un giorno di materia tipo matematica se abbiamo già max_ore_consec
#define N_O_G_SUP1_AFT_MAXCONSEC      2
// ogni ora in un giorno di materia tipo matematica se abbiamo già max_ore_consec e il giorno prec = VERO
#define  N_O_G_AFT_MAXCONSEC_SUCC    2
// ogni materia che non abbia max_ore_consec
#define NO_MAXCONSEC  20
// ogni prima o ultima ora "di troppo" alla settimana
#define O_TROPPO            5



        // numero di professori teorici
        // valore caricato a inizio main
```

```
int prof_teorici = 0, **ore_pgiornaliere, ***ore_mgiornaliere;
int **prev_ore_pg, ***prev_ore_mg;
unsigned long int dssize() ;
char * salvatutto(tab_prof *, tab_classi *,tab_dati_cattedre *cattedre);
void confrontasd(char *a, char *b );
/* Predichiarazione funzioni
*/
void loops0(tab_classi o_c, tab_classi p_o_c, tab_prof o_p, tab_prof p_o_p, tab_dati_cattedre d_c, tab_dati_cattedre p_d_c,
tab_classi b_o_c, tab_prof b_o_p, tab_classi w_o_c, tab_prof w_o_p, tab_dati_prof d_p, int STALLO, int livello, float *t, int
*best_quality, int *worst_quality,int *prev_quality);
void loops(tab_classi o_c, tab_classi p_o_c, tab_prof o_p, tab_prof p_o_p, tab_dati_cattedre d_c, tab_dati_cattedre p_d_c,
tab_classi b_o_c, tab_prof b_o_p, tab_classi w_o_c, tab_prof w_o_p, tab_dati_prof d_p, int STALLO, int livello, float *t, int
*best_quality, int *worst_quality,int *prev_quality);
void copia_spec (tab_classi *classi, tab_prof *prof,tab_dati_cattedre *cat);
void classi2prof (tab_classi *classi, tab_prof *prof,tab_dati_cattedre *cat);
void prof2classi (tab_classi *classi, tab_prof *prof);   // per stampare (oltre che aggiornare) l'orario
int scambia      (tab_classi *classi, tab_prof *prof, tab_dati_cattedre *,int verbose);
void copia_ore(int **pg1,int **pg2, int ***mg1, int ***mg2);
int effettua_scambio  (tab_classi *classi, tab_prof *prof, int ora1, int ora2, int cat1, int prof2, int classe,tab_dati_cattedre *cat);
int scambi_effettivi   (tab_classi *tab1, tab_classi *tab2);
int qualita (tab_classi *classi, tab_prof *prof, tab_dati_prof *dati_prof, tab_dati_cattedre *dati_cattedre);
int tenta_scambio  (tab_classi *classi, tab_prof *prof, int cat1, int ora1, int classe, int level,tab_dati_cattedre *);
void stampastrutturadati (char *nomefile ,tab_classi *orario_classi,tab_prof *orario_prof,tab_dati_cattedre *cattedre);
int controllo_ore_singolo_prof (tab_prof *tab1, tab_prof *tab2,tab_dati_cattedre *cat);
void restorefromfile(char *);
void backtrack( tab_classi *classi, tab_prof *prof, tab_dati_cattedre *cat);
void openfile_best_quality(FILE **fbq,int totproc,char * type);
void closefile_best_quality(FILE **fbq, int totproc);
void rebuild(char *toret, tab_classi *orario_classi, tab_dati_cattedre *cattedre, tab_prof *orario_prof);
#ifdef SCREEN_DEBUG
        void spaziatura  (int n);
#endif

/* Variabili ad uso statistico nelle funzioni di scambio
*/
#ifdef STAT_DEBUG
   unsigned int STAT_super_max_level = 0;
   unsigned int STAT_somma_levels = 0;
#endif


        // Malloc delle strutture dati
tab_prof _prof_malloc(int a ) {
        tab_prof  *toret;
        toret=(tab_prof *) malloc(sizeof(tab_prof));
        toret[0]=(struct prof_st *) malloc(a*sizeof(struct prof_st));
        return toret[0];
}

tab_classi _classi_malloc(int a) {
        tab_classi *toret;
        toret=(tab_classi *) malloc(sizeof(tab_classi));
        toret[0]=(struct classi_st *) malloc(a*sizeof(struct classi_st));
        return toret[0];
}

void azzeraprof(tab_prof *orario_prof) {
        int i,j;
        for(i=0; i<nprof; i++) {
                for(j=0; j<ORE; j++){
                        (*orario_prof)[i].cat[j] = PROF_LIBERO;
```

```
                        (*orario_prof)[i].lab[j] = NO_LAB;
                }
        }
}
```

.

Fine datatypev.h ; inizio orario.c

```
/*
   Programma di proprieta` della SCP

   Sviluppato da:
        Baccega Andrea          vekexasia@libero.it
        Bucciol Antonio         info@freshbuc.com
        Sartorello Enrico   enrico@sartorello.org

   Per informazioni visitare l'url www.scpvolterra.org
*/

#include "datatypev.h"
#include "porting.h"

//int m_quality[NMQUALITY];
int q_max=0,q_min=0;
stack *Backtrack;
int main(int argc, char **argv) {

        int id;
        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&id);


//      printf("Buondì from %d\ n",id);

        If_init(id,LF_TIME_RESEED);
        Backtrack= creastack(MAX_PROFONDITA+2);
        unsigned long int tot_cicli = (SINGLE_VALORE_STALLO + MULTIPLE_VALORE_STALLO) * EXTERNAL_LOOP;

        unsigned int start_time = time(0),interloop[2];
        unsigned long int i, n;                 // contatore loop principale
        int j, k,k1,cmd;                        // contatori
        float   t=1;                            // temperatura
        float t1=0;
        tab_classi first_timetable;  // conterra`  la timetable iniziale
        tab_classi best_timetable;   // conterra`  la miglior timetable trovata
        tab_classi worst_timetable; // peggiore
        tab_classi prev_timetable;   // conterra` la penultima timetable (per tempratura)
        char *ds1,*ds2;
        tab_prof first_proftable;
        tab_prof best_proftable;
        tab_prof worst_proftable;
        tab_prof prev_proftable;

        int quality;
        int best_quality;
        int worst_quality = 0;
        int prev_quality;
        int ext_quality;     // qualità all'inizio del ciclo esterno
        int ext_initial_quality;
        int ext_quality_difference = 0;
```

```
	int last_accepted_quality;
	int best_all;

	int n_cicli_esterni = EXTERNAL_LOOP;
	int same_ext_quality = 0;
	int initial_level;	// valore iniziale di `level' in tenta_scambio (passato da scambio())

	#ifdef STAT_DEBUG
		unsigned long int singoli = 0, multipli = 0; // numero di scambi riusciti
	#endif


		// dichiaro e inizializzo:
	tab_classi		orario_classi;	//	. matrice orario classi
	tab_prof	orario_prof;	//	. matrice orario prof
	tab_dati_prof		dati_prof;	//	. matrice dati prof
	tab_dati_cattedre	dati_cattedre;	//	. matrice dati cattedre
	tab_dati_cattedre	prev_cattedre;
	leggiorario("ORARIO.bsb");
	creaorario();

	first_timetable = _classi_malloc(nclas-2);
	best_timetable  = _classi_malloc(nclas-2);
	worst_timetable = _classi_malloc(nclas-2);
//	prev_timetable = _classi_malloc(nclas-2);
	orario_classi = _classi_malloc(nclas-2);

	first_proftable = _prof_malloc(nprof);
	best_proftable  = _prof_malloc(nprof);
	worst_proftable = _prof_malloc(nprof);
//	prev_proftable = _prof_malloc(nprof);
	orario_prof = _prof_malloc(nprof);

	ore_pgiornaliere=(int **) malloc(nprof*sizeof(int *));
	prev_ore_pg=(int **) malloc(nprof*sizeof(int *));
	ore_mgiornaliere=(int ***) malloc(nmat*sizeof(int **));
	prev_ore_mg=(int ***) malloc(nmat*sizeof(int **));
	dati_prof=(int **) malloc(nprof*sizeof(int *));
	for (i=0;i<nmat;i++) {
		ore_mgiornaliere[i]=(int **) malloc((nclas-2)*sizeof(int *));
		prev_ore_mg[i]=(int **) malloc((nclas-2)*sizeof(int *));
		for (j=0;j< nclas-2;j++){
			ore_mgiornaliere[i][j]=(int *) malloc(GIORNI *sizeof(int));
			prev_ore_mg[i][j]=(int *) malloc(GIORNI *sizeof(int));
			memset(ore_mgiornaliere[i][j],0,GIORNI*sizeof(int));
			memset(prev_ore_mg[i][j],0,GIORNI*sizeof(int));
		}
	}
	for (i=0;i<nprof;i++) {
		ore_pgiornaliere[i]=(int *) malloc(GIORNI*sizeof(int));
		prev_ore_pg[i]=(int *) malloc(GIORNI*sizeof(int));
		memset(ore_pgiornaliere[i],0,GIORNI*sizeof(int));
		memset(prev_ore_pg[i],0,GIORNI*sizeof(int));
		dati_prof[i]=(int *) malloc(3*sizeof(int));
	}

	dati_cattedre=(tab_dati_cattedre) malloc(ncattedre*sizeof(struct catt_st));
//	prev_cattedre=(tab_dati_cattedre) malloc(ncattedre*sizeof(struct catt_st));
	// azzeramento struttura dati
	for(i=0; i<nprof; i++) {
		for(j=0; j<ORE; j++){
```

```
                orario_prof[i].cat[j] = PROF_LIBERO;
                orario_prof[i].lab[j] = NO_LAB;
                }
        }

        for (i=0; i<nclas-2; i++)
                for (j=0; j<ORE; j++){
                        orario_classi[i].cat1[j] = ORA_DISPOSIZIONE;
                        orario_classi[i].cat0[j] = ORA_DISPOSIZIONE;
                }

        porting(&orario_classi, &orario_prof, &dati_prof, &dati_cattedre);

        for(i=0; i<nprof; i++) {
                for(j=0; j<GIORNI; j++){
                        for(k=0; k<ORE_GIORNALIERE; k++)  // se è un ora lavorativa allora aggiorno ore_giornaliere [ prof ] [
giorno ]
                                if(orario_prof[i].cat[k+(j*ORE_GIORNALIERE)] != -1) {
                                        ore_pgiornaliere[i][j]++;
                                        //bisogna controllare che modifichi la roba giusta

        ore_mgiornaliere[dati_cattedre[orario_prof[i].cat[k+(j*ORE_GIORNALIERE)]].materia][dati_cattedre[orario_prof[i].cat[k
+(j*ORE_GIORNALIERE)]].classe][j]++;
                                }
                }
                dati_prof[i][ID_TDP_TEORICO] = ID_PROF_NON_TEORICO;

                for(j=0; j<ORE; j++)
                        if(orario_prof[i].lab[j] == LAB_NONE){
                                dati_prof[i][ID_TDP_TEORICO] = ID_PROF_TEORICO;
                                prof_teorici++;
                                break;
                        }
        }

        classi2prof(&orario_classi, &orario_prof, &dati_cattedre);

        best_quality = qualita(&orario_classi, &orario_prof, &dati_prof, &dati_cattedre);
        prev_quality = best_quality;
        ext_quality = best_quality;
        ext_initial_quality = best_quality;
        last_accepted_quality = best_quality;
        char *buff_orario;
        int tmpb;
        if (id==0) {
                tab_prof tmp_prof = _prof_malloc(nprof);
                tab_classi tmp_classi = _classi_malloc(nclas-2);
                tab_dati_cattedre tmp_catt =(tab_dati_cattedre) malloc(ncattedre*sizeof(struct catt_st));
                int totproc,best_quality_proc;
                FILE *command;
                FILE **fbq;
                int idb;
                MPI_Status status;

                #ifdef FILE_DEBUG
                        stampastrutturadati("debug1.html",&orario_classi,&orario_prof,&dati_cattedre);
                #endif

                /* snapshot delle tabelle iniziali, per confronti successivi */
                memcpy(first_timetable, orario_classi, (nclas-2)*3*ORE*sizeof(int));
                memcpy(first_proftable, orario_prof, nprof*2*ORE*sizeof(int));
```

```
copia_ore(prev_ore_pg, ore_pgiornaliere, prev_ore_mg, ore_mgiornaliere);
void    termination_handler (int signum)  {
        int scelta;
        FILE *fa;
        unsigned int totaltime = time(0) - start_time;
        #ifdef FORKIAMO
                printf("%d signum\ n",signum);
                if (signum==10){
                        printf("\ t -!-\ t stampa l'orario best per ora trovato\ n");
                        stampastrutturadati("best.html",&best_timetable,&best_proftable,&dati_cattedre);
                }
                if (signum==16) {
                        printf("\ t -!-\ t stampa l'orario worst per ora trovato\ n");
                        stampastrutturadati("worst.html",&best_timetable,&best_proftable,&dati_cattedre);
                }
                if (signum==30) {
                        printf("\ t -!-\ t stampa l'orario iniziale\ n");
                        stampastrutturadati("start.html",&best_timetable,&best_proftable,&dati_cattedre);
                                                printf("\ t -!-\ t Modifica numero di cicli
esterni\ n");
                }
                if (signum==31) {
                        fa=stdout;
                        fprintf(fa, "Start time %ui\ n", start_time);
                        fprintf(fa, "Total time: %u secondi\ n", totaltime);
                        fprintf(fa, "%lu scambi tentati.\ n", tot_cicli);
                        fprintf(fa, "Temperatura %12f\ n",t);
                        fprintf(fa, "prev_quality = %d , quality=%d\ n",prev_quality, quality);
                        fprintf(fa, "Il cambiamento assoluto e` di %d caselle\ n",
scambi_effettivi(&orario_classi, &first_timetable));
                        fprintf(fa, "Il miglior valore di qualita` raggiunto e` stato %i ed il peggiore %i\ n",
best_quality, worst_quality);
                }
        #else
                do {
                        printf("\ t -1-\ t stampa l'orario best per ora trovato\ n");
                        printf("\ t -2-\ t stampa l'orario worst per ora trovato\ n");
                        printf("\ t -3-\ t stampa l'orario iniziale\ n");
                        printf("\ t -4-\ t Continua l'esecuzione\ n");
                        printf("\ t -5-\ t Cambia il numero di cicli esterni\ n");
                        printf("\ t -6-\ t Termina l'esecuzione\ n");
                        scanf("%d",&scelta);
                        switch (scelta) {
                                case 5: do {
                                                printf("Inserire il nuovo numero di cicli esterni: ");
                                                scanf("%d", &n_cicli_esterni);
                                        } while (n_cicli_esterni<0);
                                                break;
                                case 3:
stampastrutturadati("start.html",&first_timetable,&first_proftable,&dati_cattedre);
                                                break;
                                case 2:
        stampastrutturadati("worst.html",&worst_timetable,&worst_proftable,&dati_cattedre);
                                                break;
                                case 1:
        stampastrutturadati("best.html",&best_timetable,&best_proftable,&dati_cattedre);
                                                break;
                                case 6:
                                                #ifdef FINAL_DEBUG
                                                        char *filename = "final_results.txt";
```

```
                                                                        if((fa = fopen(filename, "a")) == NULL)
                                                                                fa = fopen(filename, "w");
                                                                        else
                                                                                fa = fopen(filename, "a");
                                                                #else
                                                                        fa=stdout;
                                                                #endif
                                                                fprintf(fa, "Start time %ui\ n", start_time);
                                                                fprintf(fa, "Total time: %u secondi\ n", totaltime);
                                                                fprintf(fa, "%lu scambi tentati.\ n", tot_cicli);
                                                                #ifdef STAT_DEBUG
                                                                        fprintf(fa, "%lui scambi singoli e %lui scambi
multipli riusciti\ n", singoli, multipli);
                                                                        fprintf(fa, "%u scambi non riusciti perche`
fuori limite max\ n", STAT_super_max_level);
                                                                        fprintf(fa, "Media valore 'level': %.3f (abs
%u)\ n", ( ((float)STAT_somma_levels) / (MULTIPLE_LOOP*EXTERNAL_LOOP)), STAT_somma_levels);
                                                                #endif
                                                                fprintf(fa, "Il cambiamento assoluto e` di %d caselle\ n",
scambi_effettivi(&orario_classi, &first_timetable));
                                                                fprintf(fa, "Il miglior valore di qualita` raggiunto e`
stato %i ed il peggiore %i\ n", best_quality, worst_quality);
                                                                fprintf(fa, "\ n\ n");
                                                                fclose(fa);
                                                                getchar();
                                                                exit(1);
                                                        default : printf("\ n\ nWARNING -> Impossibile soddisfarti sessualmente :)
\ n\ n");
                                                                }
                                                        } while(scelta!=4);
                                                #endif
                                        }
                                        // Inserisco la possibilita di cambiare l'esecuzione del programma
                                        #ifdef FORKIAMO
                                                pid_t pid;
                                                pid=fork();
                                                if (pid==0) {
                                                        signal(30,termination_handler);
                                                        signal(16,termination_handler);
                                                        signal(10,termination_handler);
                                                        signal(31,termination_handler);
                                        #else
                                        signal(SIGINT,termination_handler);
                                        #endif

                                        interloop[0]=time(0);

                                        copia_spec (&orario_classi, &orario_prof,&dati_cattedre);
                                        //t=((double)1)/pow(0.005,ESPONENTE);
                                        same_ext_quality=0;
                                //      memcpy(prev_timetable, orario_classi, (nclas-2)*3*ORE*sizeof(int));
                                //      memcpy(prev_proftable, orario_prof, nprof*2*ORE*sizeof(int));
                                //      memcpy(prev_cattedre, dati_cattedre, ncattedre*sizeof(struct catt_st));
                                        int n_quality=0;
                                        MPI_Comm_size(MPI_COMM_WORLD,&totproc);
                                        fbq=(FILE **) malloc(totproc*(sizeof(FILE *)));
                                        // creo i file per gnuplot con i best quality
                                        for(k=1,k1=1; ; k++,k1++){
                                        //copia_spec (&orario_classi, &orario_prof);
                                                interloop[(k+1)%2]=time(0);
                                        //      slf_rand(0);
```

```
                        n_quality=0;
                        ext_quality_difference=0;
                        t=((double)1)/pow(k1,ESPONENTE);
                        if (t< 0.18){
                                k1=1.25;
                                t=((double)1)/pow(k1,ESPONENTE);
                        }
                        printf("Starting %i out of %i reannealing loops. Best quality %d ,time %d, temp %.12f\ n", k,
n_cicli_esterni,best_quality,interloop[(k+1)%2]-interloop[k%2],t);

                        ext_initial_quality = prev_quality;
                        q_max=q_min=prev_quality;
                        loops0(orario_classi, first_timetable, orario_prof, first_proftable, dati_cattedre, dati_cattedre,
best_timetable, best_proftable, worst_timetable, worst_proftable, dati_prof, SINGLE_VALORE_STALLO, 0, &t, &best_quality,
&worst_quality, &prev_quality);

                        loops0(orario_classi, first_timetable, orario_prof, first_proftable, dati_cattedre, dati_cattedre,
best_timetable, best_proftable, worst_timetable, worst_proftable, dati_prof, MULTIPLE_VALORE_STALLO, MAX_PROFONDITA-
1, &t, &best_quality, &worst_quality, &prev_quality);


                        ext_quality_difference=q_max-q_min;
                        printf ("Temperatura finale al ciclo esterno = %12f, pq %d,  QD %d
\ n",t,prev_quality,ext_quality_difference);
                        if (ext_quality == best_quality) {
                                if (++same_ext_quality > STALLO_EXT_MAX) { //fermo da troppo tempo: e qui ci
incazziamo di brutto
                                        if (ext_quality_difference  < 3){
                                                printf("Spostamento carichi pesanti\ n");
                                                k1=k1-STALLO_EXT_MAX+1;
                                                for (i=0; i<ZOMPA_DOMINIO;) {
                                                        if (scambia(&orario_classi, &orario_prof,&dati_cattedre,
0) == RIUSCITO)
                                                                i++;
                                                        _strash(Backtrack);
                                                }
                                                copia_ore(prev_ore_pg, ore_pgiornaliere, prev_ore_mg,
ore_mgiornaliere);
                        //                      memcpy(prev_timetable, orario_classi, (nclas-
2)*3*ORE*sizeof(int));
                        //                      memcpy(prev_proftable, orario_prof, nprof*2*ORE*sizeof(int));
                        //                      memcpy(prev_cattedre, dati_cattedre, ncattedre*sizeof(struct
catt_st));
                                                same_ext_quality=0;
                                                n_quality=0;
                                                prev_quality=qualita(&orario_classi, &orario_prof, &dati_prof,
&dati_cattedre);
                                        }else {
                                                same_ext_quality=0;
                                                printf("Spostamento carichi pesanti EVITATO\ n");
                                        }
                                }
                        } else {
                                ext_quality=best_quality;
                                same_ext_quality=0;
                        }

                        MPI_Barrier(MPI_COMM_WORLD);

                        command = fopen("cmd.txt","r");
                        fscanf(command,"%d ",&cmd);
```

```
		fclose(command);

		MPI_Bcast(&cmd,1,MPI_INT,0,MPI_COMM_WORLD);
		if (cmd == 0 || cmd==1) {
			tmpb=best_quality*100+id;
			MPI_Allreduce(&tmpb,&best_all,1,MPI_INT,MPI_MIN,MPI_COMM_WORLD);
			if (best_all == tmpb) {
				printf("Qualità migliore da id 0\ n");
				stampastrutturadati("best.html",&best_timetable,&best_proftable,&dati_cattedre);
			} else {
				idb=best_all%100;
				//MPI_Recv(&idb,1,MPI_INT,MPI_ANY_SOURCE,0,MPI_COMM_WORLD,&status);
				printf("Qualità migliore da id %d: %d\ n",idb, best_all/100);
				buff_orario=(char *) malloc(dssize());
				MPI_Recv(buff_orario, dssize(), MPI_CHAR, idb, 0, MPI_COMM_WORLD, &status);
				rebuild(buff_orario, &tmp_classi, &tmp_catt, &tmp_prof);
				stampastrutturadati("best.html", &tmp_classi, &tmp_prof, &tmp_catt);
				free(buff_orario);
			}
			MPI_Barrier(MPI_COMM_WORLD);

			if (cmd==0)
				break;
			else {
				command = fopen("cmd.txt","w");
				fprintf(command,"2\ n");
				fclose(command);
			}
		}
		openfile_best_quality(fbq, totproc, "a");
		fprintf(fbq[0], "%d\ n", best_quality);
		for (i=1; i<totproc; i++) {
			MPI_Recv(&best_quality_proc,1,MPI_INT,i,0,MPI_COMM_WORLD,&status);
			fprintf(fbq[i], "%d\ n", best_quality_proc);
		}
		closefile_best_quality(fbq, totproc);
//		t+=0.1;
} // fine EXTERNAL_LOOP
#ifdef FILE_DEBUG
		stampastrutturadati("start.html",&first_timetable,&first_proftable,&dati_cattedre);
		stampastrutturadati("worst.html",&worst_timetable,&worst_proftable,&dati_cattedre);
		stampastrutturadati("best.html",&best_timetable,&best_proftable,&dati_cattedre);
#endif

unsigned int totaltime = time(0) - start_time;
FILE *fa;
#ifdef FINAL_DEBUG
		char *filename = "final_results.txt";
		if((fa = fopen(filename, "a")) == NULL)
			fa = fopen(filename, "w");
		else
			fa = fopen(filename, "a");
#else
		fa=stdout;
#endif
f printf(fa, "Start time %ui\ n", start_time);
fprintf(fa, "Total time: %u secondi\ n", totaltime);
fprintf(fa, "%lu scambi tentati.\ n", tot_cicli);
#ifdef STAT_DEBUG
		fprintf(fa, "%lui scambi singoli e %lui scambi multipli riusciti\ n", singoli, multipli);
		fprintf(fa, "%u scambi non riusciti perche` fuori limite max\ n", STAT_super_max_level);
```

```
                        fprintf(fa, "Media valore 'level': %.3f (abs %u)\ n", ( ((float)STAT_somma_levels) /
(MULTIPLE_LOOP*EXTERNAL_LOOP)), STAT_somma_levels);
                #endif
                fprintf(fa, "Il cambiamento assoluto e` di %d caselle\ n", scambi_effettivi(&orario_classi, &first_timetable));
                fprintf(fa, "Il miglior valore di qualita` raggiunto e` stato %i ed il peggiore %i\ n", best_quality, worst_quality);
                //fprintf(fa, "\ n\ n");

                #ifdef FORKIAMO
                        char nomefile[256];
                        sprintf(nomefile,"tar -czf b%dw%d.tgz *.html",best_quality,worst_quality);
                        //printf("%s\ n",nomefile);
                        system(nomefile);
                        fclose(fa);
                        system("rm -rf *.html");
                #endif
                getchar();
                #ifdef FORKIAMO
                        } else {
                                void term_child(int signum) {
                                        kill(pid,SIGINT);
                                        exit(1);
                                }
                                signal(SIGINT,term_child);
                                printf("%d Pid del sottoprocesso\ n",pid);
                                char c;
                                do {
                                        scanf("%c",&c);
                                        if (c=='b')
                                                kill(pid,10);
                                        else if (c=='w')
                                                kill(pid,16);
                                        else if(c=='i')
                                                kill(pid,30);
                                        else if(c=='s')
                                                kill(pid,31);
                                        else if (c=='q')
                                                kill(pid,SIGINT);
                                } while (c!='q');
                        }
                #endif

        } else {
                copia_spec (&orario_classi, &orario_prof,&dati_cattedre);
                same_ext_quality=0;
                int n_quality=0;
                for(k=1,k1=1; ; k++,k1++) {
                        interloop[(k+1)%2]=time(0);
                        n_quality=0;
                        ext_quality_difference=0;
                        t=((double)1)/pow(k1,ESPONENTE);
                        if (t< 0.18){
                                k1=1.25;
                                t=((double)1)/pow(k1,ESPONENTE);
                        }
                        ext_initial_quality = prev_quality;
                        q_max=q_min=prev_quality;
                        loops(orario_classi, first_timetable, orario_prof, first_proftable, dati_cattedre, dati_cattedre,
best_timetable, best_proftable, worst_timetable, worst_proftable, dati_prof, SINGLE_VALORE_STALLO, 0, &t, &best_quality,
&worst_quality, &prev_quality);

                        loops(orario_classi, first_timetable, orario_prof, first_proftable, dati_cattedre, dati_cattedre,
```

```
best_timetable, best_proftable, worst_timetable, worst_proftable, dati_prof, MULTIPLE_VALORE_STALLO, MAX_PROFONDITA-
1, &t, &best_quality, &worst_quality, &prev_quality);

                                ext_quality_difference=q_max-q_min;
                                if (ext_quality == best_quality) {
                                        if (++same_ext_quality > STALLO_EXT_MAX) { //fermo da troppo tempo: e qui ci incazziamo
di brutto
                                                if (ext_quality_difference  < 3) {
                                                        k1=k1-STALLO_EXT_MAX+1;
                                                        for (i=0; i<ZOMPA_DOMINIO;) {
                                                                if (scambia(&orario_classi, &orario_prof,&dati_cattedre, 0) ==
RIUSCITO)
                                                                        i++;
                                                                _strash(Backtrack);
                                                        }
                                                        copia_ore(prev_ore_pg, ore_pgiornaliere, prev_ore_mg, ore_mgiornaliere);
                                                        same_ext_quality=0;
                                                        n_quality=0;
                                                        prev_quality=qualita(&orario_classi, &orario_prof, &dati_prof,
&dati_cattedre);
                                                } else {
                                                        same_ext_quality=0;
                                                }
                                        }
                                } else {
                                        ext_quality=best_quality;
                                        same_ext_quality=0;
                                }

                                MPI_Barrier(MPI_COMM_WORLD);
                                MPI_Bcast(&cmd,1,MPI_INT,0,MPI_COMM_WORLD);
                                if (cmd == 0 || cmd==1) {
                                        tmpb=best_quality*100+id; // Questo è per la best_quality che sia veramente best
                                        MPI_Allreduce(&tmpb, &best_all, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
                                        if (best_all == tmpb) {
                                                //MPI_Send(&id, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
                                                buff_orario=salvatutto(&orario_prof, &orario_classi, &dati_cattedre);
                                                MPI_Send(buff_orario, dssize(), MPI_CHAR, 0, 0, MPI_COMM_WORLD);
                                                free(buff_orario);
                                        }
                                        MPI_Barrier(MPI_COMM_WORLD);
                                        if (cmd==0)
                                                break;
                                }
                                MPI_Send(&best_quality, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        //              t+=0.1;
                        } // fine EXTERNAL_LOOP


        }

        MPI_Finalize();
}
void openfile_best_quality(FILE **fbq,int totproc,char * type) {
        int i;
        char BUFF[100];
        for (i=0; i<totproc; i++) {
                sprintf(BUFF, "quality%d.txt", i);
                fbq[i]=fopen(BUFF, type);
        }
}
void closefile_best_quality(FILE **fbq, int totproc) {
```

```
	int i;
	for (i=0; i<totproc; i++)
		fclose(fbq[i]);
}
int scambia (tab_classi *classi, tab_prof *prof,tab_dati_cattedre *cat, int level){
		/* scelta di un'ora in modo casuale */
	int ora1;
	ora1 = lf_rand()%ORE;

		/* scelta casuale del giorno in cui avverrÃ  lo scambio */
	int classe = lf_rand()%(nclas-2);

	int cat1 = (*classi)[classe].cat0[ora1];

	#ifdef SCREEN_DEBUG
		printf("Ora di scambio di partenza %i \ t nella classe %i\ n", ora1, classe);
	#endif

	return tenta_scambio(classi, prof, cat1, ora1, classe, level,cat);
}

void rebuild(char *toret, tab_classi *orario_classi, tab_dati_cattedre *cattedre, tab_prof *orario_prof) {
	int i,j;
	j=0;
	for (i=0; i<ncattedre; i++,j+=sizeof(struct catt_st))
		memcpy(&(*cattedre)[i], &toret[j], sizeof(struct catt_st));
	for (i=0; i<nprof; i++, j+=sizeof(struct prof_st))
		memcpy(&(*orario_prof)[i], &toret[j], sizeof(struct prof_st));
	for (i=0; i<nclas-2; i++,  j+=sizeof(struct classi_st))
		memcpy(&(*orario_classi)[i], &toret[j], sizeof(struct classi_st));

}

int tenta_scambio (tab_classi *classi, tab_prof *prof, int cat1, int ora1, int classe, int level,tab_dati_cattedre *cat) {
	if (level >= MAX_PROFONDITA){
		#ifdef STAT_DEBUG
			STAT_super_max_level++;
		#endif
		return FALLITO;
	}

	#ifdef STAT_DEBUG
		STAT_somma_levels += level;
	#endif

	int result;
	int ora2;

	do {
		ora2 = lf_rand()%ORE;
	} while(ora1 == ora2);

	int cat2 = (*classi)[classe].cat0[ora2];
	int prof1= (*cat)[cat1].prof;
	int prof2= (*cat)[cat2].prof;
	#ifdef SCREEN_DEBUG
		spaziatura(level);
		printf(" %i\ n", level);
	#endif

	do{
```

```
                if (prof1 == prof2 || (*prof)[prof1].lab[ora1] != NO_LAB || (*prof)[prof2].lab[ora2] != NO_LAB)
                        return FALLITO;
                else
                        if ((*prof)[prof1].cat[ora2] == PROF_LIBERO){
                                if ((*prof)[prof2].cat[ora1] == PROF_LIBERO){
                                        effettua_scambio(classi, prof, ora1, ora2, cat1, cat2, classe,cat);
                                                #ifdef SCREEN_DEBUG
                                                        spaziatura(level);
                                                        printf("  !\ n");
                                                #endif
                                        return RIUSCITO;
                                } else  {
                                        if ((*prof)[prof2].lab[ora1] != NO_LAB)
                                                return FALLITO;                 // prof2 occupato con lab
                                        result = tenta_scambio(classi, prof, cat2, ora1, (*cat)[cat2].classe, level+1,cat);
                                }
                        } else
                                if ((*prof)[prof2].cat[ora1] != PROF_LIBERO)
                                        return FALLITO;             // entrambi prof occupati
                                else{
                                        if ((*prof)[prof1].lab[ora2] != NO_LAB)
                                                return FALLITO;         // cat1 occupato lab
                                        result = tenta_scambio(classi, prof, cat1, ora2, (*cat)[cat2].classe, level+1,cat);
                                        }
                #ifdef SCREEN_DEBUG
                        spaziatura(level);
                        printf(" %i\ n", level);
                #endif
        }while (result != FALLITO);
        return result;
}

void copia_ore(int **pg1,int **pg2, int ***mg1, int ***mg2) {
        /* effettua la copia delle tabelle delle ore giornaliere di materie e professori, in altre due tabelle di backup */
        /* utile al backtracking per la tempratura */
        int i,j;
        for (i=0; i<nprof; i++)
                memcpy(pg1[i],pg2[i],GIORNI*sizeof(int));
        for (i=0; i<nmat; i++)
                for (j=0; j<nclas-2; j++)
                        memcpy(mg1[i][j], mg2[i][j], GIORNI*sizeof(int));
}

int effettua_scambio (tab_classi *classi, tab_prof *prof, int ora1, int ora2, int cat1, int cat2, int classe,tab_dati_cattedre *cat) {
        /* effettuo lo scambio tra i due professori */
        /* aggiorno la tabella orario_prof, la tabella orario_classi e le tabelle delle ore giornaliere di materie e professori */
        /* Implementazione del backtrack per velocizzare tutto */
        char forbacktrack[sizeof(int)*5];
        memcpy(&forbacktrack[sizeof(int)*0], &ora1, sizeof(int));
        memcpy(&forbacktrack[sizeof(int)*1], &ora2, sizeof(int));
        memcpy(&forbacktrack[sizeof(int)*2], &cat1, sizeof(int));
        memcpy(&forbacktrack[sizeof(int)*3], &cat2, sizeof(int));
        memcpy(&forbacktrack[sizeof(int)*4], &classe, sizeof(int));
        _spush( Backtrack, forbacktrack, sizeof(int)*5);
        /* FIne implementazione backtrack */
        //printf("Nel =%d ,Ora1 = %d , Ora2= %d, Cat1=%d, Cat2=%d, Classe= %d\ n", Backtrack->nel, ora1, ora2, cat1, cat2, classe);
        char tmp = (*cat)[(*prof)[(*cat)[cat1].prof].cat[ora1]].classe;
        if ((*cat)[(*prof)[(*cat)[cat2].prof].cat[ora2]].classe!= tmp){
                free(_spop(Backtrack));
        //      if ( _sempty(Backtrack)==0){
        //              printf("maybe here error\ n");
```

```
//                      backtrack(classi, prof, cat);
//          }
                return 0;
        }
        ore_pgiornaliere[(*cat)[cat2].prof] [ora2/6 ] --;
        ore_pgiornaliere[(*cat)[cat1].prof] [ora1/6 ] --;

        ore_mgiornaliere[(*cat)[cat1].materia][classe][ora1/6]--;
        ore_mgiornaliere[(*cat)[cat2].materia][classe][ora2/6]--;

        ore_pgiornaliere[(*cat)[cat2].prof] [ora1/6 ] ++;
        ore_pgiornaliere[(*cat)[cat1].prof] [ora2/6 ] ++;
        ore_mgiornaliere[(*cat)[cat1].materia][classe][ora2/6]++;
        ore_mgiornaliere[(*cat)[cat2].materia][classe][ora1/6]++;
        if ((*prof)[(*cat)[cat1].prof].cat[ora2] != PROF_LIBERO)
                printf("Errore prof = %d  ora %d \ n",(*cat)[cat1].prof,ora2);
        if ((*prof)[(*cat)[cat2].prof].cat[ora1] != PROF_LIBERO)
                printf("Errore prof = %d  ora %d \ n",(*cat)[cat2].prof,ora1);
        (*prof)[(*cat)[cat1].prof].cat[ora2] = cat1;
        (*prof)[(*cat)[cat2].prof].cat[ora1] = cat2;
        /* ... e anche la tabella orario_classi */
        (*classi)[classe].cat0[ora1] = cat2;
        (*classi)[classe].cat0[ora2] = cat1;
        /* libero le due ore diventate libere per i prof */
        if ((*prof)[(*cat)[cat1].prof].cat[ora1] == PROF_LIBERO)
                printf("Errore prof = %d  ora %d \ n",(*cat)[cat1].prof,ora1);
        if ((*prof)[(*cat)[cat2].prof].cat[ora2] == PROF_LIBERO)
                printf("Errore prof = %d  ora %d \ n",(*cat)[cat2].prof,ora2);
        (*prof)[(*cat)[cat1].prof].cat[ora1] = PROF_LIBERO;
        (*prof)[(*cat)[cat2].prof].cat[ora2] = PROF_LIBERO;

        return 1;
}

void backtrack( tab_classi *classi, tab_prof *prof, tab_dati_cattedre *cat) {
        while (ripristina_scambio(classi, prof, cat) == 1);
        _strash(Backtrack);
}

int ripristina_scambio  (tab_classi *classi, tab_prof *prof,tab_dati_cattedre *cat) {
        char *forbacktrack=_spop(Backtrack);
        if (forbacktrack!= NULL) {
                int ora1, ora2, classe, cat1, cat2;
                memcpy(&ora2, &forbacktrack[sizeof(int)*0], sizeof(int));
                memcpy(&ora1, &forbacktrack[sizeof(int)*1], sizeof(int));
                memcpy(&cat1, &forbacktrack[sizeof(int)*2], sizeof(int));
                memcpy(&cat2, &forbacktrack[sizeof(int)*3], sizeof(int));
                memcpy(&classe, &forbacktrack[sizeof(int)*4], sizeof(int));
                free(forbacktrack);
        //      printf("AOra1 = %d , Ora2= %d, Cat1=%d, Cat2=%d, Classe= %d\ n", ora1, ora2, cat1, cat2, classe);
                char tmp = (*cat)[(*prof)[(*cat)[cat1].prof].cat[ora1]].classe;
                if ((*cat)[(*prof)[(*cat)[cat2].prof].cat[ora2]].classe!= tmp){
                        printf("Errore nel ripristino scambio\ n");
                        return -1;
                }
                ore_pgiornaliere[(*cat)[cat2].prof] [ora2/6 ] --;
                ore_pgiornaliere[(*cat)[cat1].prof] [ora1/6 ] --;

                ore_mgiornaliere[(*cat)[cat1].materia][classe][ora1/6]--;
                ore_mgiornaliere[(*cat)[cat2].materia][classe][ora2/6]--;
```

```
                ore_pgiornaliere[(*cat)[cat2].prof] [ora1/6 ] ++;
                ore_pgiornaliere[(*cat)[cat1].prof] [ora2/6 ] ++;
                ore_mgiornaliere[(*cat)[cat1].materia][classe][ora2/6]++;
                ore_mgiornaliere[(*cat)[cat2].materia][classe][ora1/6]++;
                if ((*prof)[(*cat)[cat1].prof].cat[ora2] != PROF_LIBERO)
                        printf("_Errore prof = %d  ora %d \ n",(*cat)[cat1].prof,ora2);
                if ((*prof)[(*cat)[cat2].prof].cat[ora1] != PROF_LIBERO)
                        printf("_Errore prof = %d  ora %d \ n",(*cat)[cat2].prof,ora1);
                (*prof)[(*cat)[cat1].prof].cat[ora2] = cat1;
                (*prof)[(*cat)[cat2].prof].cat[ora1] = cat2;
                /* ... e anche la tabella orario_classi */
                (*classi)[classe].cat0[ora1] = cat2;
                (*classi)[classe].cat0[ora2] = cat1;
                /* libero le due ore diventate libere per i prof */
                if ((*prof)[(*cat)[cat1].prof].cat[ora1] == PROF_LIBERO)
                        printf("_Errore prof = %d  ora %d \ n",(*cat)[cat1].prof,ora1);
                if ((*prof)[(*cat)[cat2].prof].cat[ora2] == PROF_LIBERO)
                        printf("_Errore prof = %d  ora %d \ n",(*cat)[cat2].prof,ora2);
                (*prof)[(*cat)[cat1].prof].cat[ora1] = PROF_LIBERO;
                (*prof)[(*cat)[cat2].prof].cat[ora2] = PROF_LIBERO;

                return 1;
        } else
                return 0;
}
int qualita (tab_classi *classi, tab_prof *prof, tab_dati_prof *dati_prof, tab_dati_cattedre *dati_cattedre) {
        /* Dato un orario, ne restituisce la "bontà" con un numero naturale */
        /* TODO: - gestione delle preferenze (ora solo in preorario */
        int bonta = 0;      // bonta` dell'orario (peggiore al crescere)
        int i, j, k;

                // Controllo numero ore giornaliere insegnanti teorici (massimo quattro)
//      int daily_hours;
//      char giorno_libero;
        char giorni_insegnamento;
//      char ore_insegnamento;
        for(i=0; i<nprof; i++) {
                giorni_insegnamento = 0;

                for(j=0; j<GIORNI; j++){
                        if(ore_pgiornaliere[i][j] > 4) // numero di ore max
                                bonta += (ore_pgiornaliere[i][j]-4)*O_G_SUP4; // accrescimento var bonta
                        if (ore_pgiornaliere[i][j] >0)
                                giorni_insegnamento++;
                }
                if(giorni_insegnamento < 5)
                        bonta += (5-giorni_insegnamento)*N_G_I_M5;
        }
    /* Controllo distribuzione decente delle ore: questa parte controlla che
       - le materie con meno ore non appaiano piu' di una volta al giorno, e non appaiano due giorni consecutivi
            - le materie con piu' ore non appaiano piu' di una volta al giorno, ma abbiano un giorno con due ore attaccate, e non appaiano
due giorni consecutivi
            Naturalmente ogni qualvolta accade uno di questi eventi, viene aumentata la bonta` secondo pesi ancora da vedere */

        int mat_curr;               //materia presa in considerazione
        int contmaterie = 0;
        int contmateriagg = 0;      //numero di ore che ha la materia quel giorno
        int giornodopo;
        int mat_giorno_prec = GIORNO_PREC_PRESENTE;     //indica se la materia era presente il giorno prima
//      char mat_ora_prec = MAT_ORA_PREC_PRESENTE;              //indice se la materia era presente l'ora prima
        int mat_oreconsec = FALSO;          //indica se la materia ha gia' o no un giorno con le ore richieste attaccate
```

```
	for (i=0; i<ncattedre; i++)		//azzero il mark di ogni materia, che mi servira' nella prossima valutazione
		(*dati_cattedre)[i].marked = CAT_NON_MARCATA;
	for (i=0; i<ORE; i++) {
		for (j=0; j<nclas-2; j++) {
			mat_curr
=(*dati_cattedre)[(*classi)[j].cat0[i]].materia;//########################################
			if (((*dati_cattedre)[(*classi)[j].cat0[i]].marked== CAT_NON_MARCATA) && ((*classi)[j].lab[i] ==
LAB_NONE)) {
				(*dati_cattedre)[(*classi)[j].cat0[i]].marked= CAT_MARCATA;
				contmaterie++;

				if ((*dati_cattedre)[(*classi)[j].cat0[i]].max_ore_consec == 1)  {    // la materia non ha bisogno
di piu' ore consecutive
					// Se l'ora non necessita di piu ore consecutive allora per ogni ora in piu aggiorna la
bonta.
					bonta += (ore_mgiornaliere [ mat_curr ]  [ j ] [ i /GIORNI] - 1)*N_O_G_SUP1;
					// Scalo di un giorno
					giornodopo = (i-1)/GIORNI + 1;
					if (giornodopo < GIORNI) {
						do {
							if (mat_giorno_prec == GIORNO_PREC_PRESENTE) {
								// Se la materia era presente 'ieri' allora
								// Aggiorno bonta facendo il num di ore di quella materia
presente nel giorno dopo in quella classe per 2
								// e se non ci sono ore nel giorno dopo allora marco
mat_giorno_prec come non presente.
								if ((ore_mgiornaliere[ mat_curr ] [ j ] [ giornodopo] ) >0)
									bonta+=(ore_mgiornaliere[ mat_curr ] [ j ] [
giornodopo] ) *N_O_G_SUCC;
								else
									mat_giorno_prec =
GIORNO_PREC_NON_PRESENTE;
							} else {
								// Se la materia non era presente allora controllo se è
presente oggi e aggiorno bonta
								if ((ore_mgiornaliere[ mat_curr ] [ j ] [ giornodopo ] )>0)
{
									bonta += ((ore_mgiornaliere[ mat_curr ] [ j ] [
giornodopo ] )-1)*N_O_G_SUCC;
									mat_giorno_prec = GIORNO_PREC_PRESENTE;
								}
							}
							giornodopo++;
						}
						while (giornodopo != GIORNI);
					}
				} else	{		// non e' contemplato il caso (inutile) che le ore di un prof siano <= 0
					// inizio controllo che il giorno stesso ci siano o meno altre ore della stessa materia
					k = i+1;
					contmateriagg = 1;
					while (k<(((i+1)/ORE_GIORNALIERE)+1)*ORE_GIORNALIERE) {
						if (mat_curr == (*dati_cattedre)[(*classi)[j].cat0[k]].materia) {
							contmateriagg++;
							if (contmateriagg ==
(*dati_cattedre)[(*classi)[j].cat0[k]].max_ore_consec) {
								mat_oreconsec = VERO;
								break;
							} else k++;
						} else {
							bonta += (contmateriagg-1)*N_O_NCONSEC;
```

```
                                                    break;
                                                }
                                            }
                                            for (k=k+1; k<(((i+1)/ORE_GIORNALIERE)+1)*ORE_GIORNALIERE; k++)
                                                if (mat_curr == (*dati_cattedre)[ (*classi)[j].cat0[k]].materia)
                                                    bonta += N_O_G_SUP;
                                            // fine controllo
                                            giornodopo = (i-1)/GIORNI + 1;
                                            if (giornodopo < GIORNI) {
                                                do {
                                                    if (mat_oreconsec == VERO) {
                                                        if (mat_giorno_prec == GIORNO_PREC_PRESENTE) {
                                                            // Se è presente nel giorno precedente
aggiorno bonta altrimenti non aggiorno bonta e metto non presente
                                                            if (ore_mgiornaliere [ mat_curr ] [ j ] [
giornodopo ]>0)
                                                                bonta+=(ore_mgiornaliere [ mat_curr ]
[ j ] [ giornodopo ])*N_O_G_AFT_MAXCONSEC;
                                                            else
                                                                mat_giorno_prec =
GIORNO_PREC_NON_PRESENTE;
                                                        } else {
                                                            // Se non è presente allora aggiorno bonta di 2
*
                                                            // ( il numero di ore della materia mat_curr
nella classe j nel giorno giornodopo)
                                                            if ((ore_mgiornaliere[ mat_curr ] [ j ] [
giornodopo ] )>0) {
                                                                bonta += ((ore_mgiornaliere[ mat_curr
] [ j ] [ giornodopo ] )-1)*N_O_G_SUP1_AFT_MAXCONSEC;
                                                                mat_giorno_prec =
GIORNO_PREC_PRESENTE;
                                                            }
                                                        }
                                                        giornodopo++;
                                                    } else {
                                                            contmateriagg = 0;
                                                            if (mat_giorno_prec ==
GIORNO_PREC_PRESENTE) {
                                                                mat_giorno_prec =
GIORNO_PREC_NON_PRESENTE;
                                                                for
(k=giornodopo*ORE_GIORNALIERE; k<(giornodopo+1)*ORE_GIORNALIERE; k++)
                                                                    if (mat_curr ==
(*dati_cattedre)[(*classi)[j].cat0[k]].materia) {
//
                                                                        mat_giorno_prec
== GIORNO_PREC_PRESENTE;
                                                                        if (mat_oreconsec
== FALSO) {
                                                                            k++;

        contmateriagg++;
                                                                        while
(k<(giornodopo+1)*ORE_GIORNALIERE) {
        if (mat_curr == (*dati_cattedre)[(*classi)[j].cat0[k]].materia) {
        contmateriagg++;
        if (contmateriagg == (*dati_cattedre)[(*classi)[j].cat0[k]].max_ore_consec) {
```

```
mat_oreconsec = VERO;

break;

} else k++;

} else {

bonta += (contmateriagg-1)*N_O_NCONSEC;

break;

}

}
} else bonta +=
N_O_G_AFT_MAXCONSEC_SUCC;

}
//          bonta--;  // va decrementato perche'
ci sta quasi bene (bonta += 1) un'ora in quel giorno
} else {
for
(k=giornodopo*ORE_GIORNALIERE; k<(giornodopo+1)*ORE_GIORNALIERE; k++)
if (mat_curr ==
(*dati_cattedre)[ (*classi)[j].cat0[k]].materia){
//
mat_giorno_prec =
GIORNO_PREC_PRESENTE;
if (mat_oreconsec
== FALSO) {
k++;

contmateriagg++;
while
(k<(giornodopo+1)*ORE_GIORNALIERE) {

if (mat_curr == (*dati_cattedre)[(*classi)[j].cat0[k]].materia) {

contmateriagg++;

if (contmateriagg == (*dati_cattedre)[(*classi)[j].cat0[k]].max_ore_consec) {

mat_oreconsec = VERO;

break;

} else k++;

} else {    // da controllare questo pezzo

bonta += (contmateriagg-1)*N_O_NCONSEC/2;

break;

}

}
} else bonta +=
N_O_G_AFT_MAXCONSEC_SUCC/2;

}
//          bonta--;
}
}
giornodopo++;
```

```
                                                } while (giornodopo != GIORNI);
                                        }
                                        if (mat_oreconsec == FALSO)
                                                bonta += NO_MAXCONSEC;          //la materia non ha le ore consecutive
richieste
                                        }
                                }
                        }
                }
        //  Controllo "orizzontale"
        // Controlla che una stessa cattedra non sia ripetuta più mattine sempre
        // alla prima o all'ultima ora
        // controlla la prima e l'ultima ora in modo indipendente così si
        // possono assegnare pesi diversi ai due tipi di ripetizione
        int first_cat, last_cat, first_cat_count, last_cat_count;

        for(i=0; i<nprof; i++)
                for(j=0; j<GIORNI; j++) {
                        first_cat = (*prof)[i].cat[ORE_GIORNALIERE*k];
                        last_cat = (*prof)[i].cat[ORE_GIORNALIERE*(k+1)-1];
                        first_cat_count=0;
                        last_cat_count=0;

                        for(k=j+1; k<GIORNI; k++) {
                                if( (*prof)[i].cat[ORE_GIORNALIERE*j] == first_cat)
                                        first_cat_count++;

                                if( (*prof)[i].cat[ORE_GIORNALIERE*(k+1)-1] == last_cat)
                                        last_cat_count++;
                        }

                        if (first_cat_count>1)
                                bonta += (first_cat_count-1) * O_TROPPO;
                        if (last_cat_count>1)
                                bonta+= (last_cat_count-1) * O_TROPPO;
                }
    return bonta;
}
#ifdef SCREEN_DEBUG
        void spaziatura(int n) {
                int i;
                for(i= 0; i<n; i++)
                        printf(" ");
        }
#endif

int scambi_effettivi (tab_classi *tab1, tab_classi *tab2) {
        /* conteggio delle celle differenti dati due orari */
        /* utile per capire, dati due orari, quanto essi siano distanti nel dominio */
        int celle_cambiate = 0, i, j;

        for(i=0; i<nclas-2; i++)
                for(j=0; j<ORE; j++)
                   /* togliere la seconda parte della OR nel caso in cui non si possano "toccare" le ore di laboratorio */
                        if (((*tab1)[i].cat0[j] != (*tab2)[i].cat0[j]) || ((*tab1)[i].lab[j] != (*tab2)[i].lab[j]))
                          celle_cambiate++;
        return celle_cambiate;
}
int controllo_ore_singolo_prof (tab_prof *tab1, tab_prof *tab2, tab_dati_cattedre *cat) {
        /* Controllo della correttezza del numero di ore di ogni singolo prof */
        /* Da utilizzare solo in fase di testing */
```

```
	int i, j;
	int ore_prof_inizio, ore_prof_fine;
	for(i=0; i<nprof; i++){
		ore_prof_inizio = 0;
		ore_prof_fine = 0;

		for(j=0; j<ORE; j++){
			if( (*tab1)[i].cat[j] != PROF_LIBERO)
				ore_prof_inizio++;
			if( (*tab2)[i].cat[j] != PROF_LIBERO)
				ore_prof_fine++;
		}

		if(ore_prof_inizio != ore_prof_fine){
			printf("Numero ore iniziali (%i) prof %i non combaciano con numero ore finali (%i)\ n", ore_prof_inizio, i,
ore_prof_fine);
			getchar();
		}
	}

	return 0;
}
void classi2prof (tab_classi *classi, tab_prof *prof,tab_dati_cattedre *cat) {
	/* dato l'orario per classe, restituisce l'orario dei prof */
	int i, j;

	for(i=0; i<nclas-2; i++)
		for(j=0; j<ORE; j++){
			(*prof)[(*cat)[ (*classi)[i].cat0[j]].prof ].cat[j] = (*classi)[i].cat0[j];
			(*prof)[(*cat)[(*classi)[i].cat0[j]].prof  ].lab[j] = (*classi)[i].lab[j];
			if ( (*classi)[i].cat1[j] != ORA_DISPOSIZIONE) {
				(*prof)[(*cat)[(*classi)[i].cat1[j]].prof  ] .cat[j] =(*classi)[i].cat0[j];
				(*prof)[(*cat)[(*classi)[i].cat1[j]].prof  ].lab[j] =(*classi)[i].lab[j];
			}
		}
}
void prof2classi (tab_classi *classi, tab_prof *prof) {
	/* dato l'orario dei prof, restituisce l'orario per classe */
	int i, j;
	for(i=0; i<nprof; i++)
		for(j=0; j<ORE; j++){
			if ((*classi)[ (int)(*prof)[i].cat[j] ].cat0[j]== ORA_DISPOSIZIONE) {
				(*classi)[ (int)(*prof)[i].cat[j] ].cat0[j] = i;
				(*classi)[ (int)(*prof)[i].cat[j] ].lab[j] = (*prof)[i].lab[j];
			} else {
				(*classi)[ (int)(*prof)[i].cat[j] ].cat1[j] = i;
				(*classi)[ (int)(*prof)[i].cat[j] ].lab[j] = (*prof)[i].lab[j];
			}
		}
}


void stampastrutturadati(char *nomefile ,tab_classi *orario_classi,tab_prof *orario_prof,tab_dati_cattedre *cattedre) {
	FILE *fw;
	int i,j,z;
	fw=fopen(nomefile,"w");
	// stampa classi:
	fprintf (fw,"<html>\ n\ t<head><title> Debug Mode on by Veke </title></head><body>\ n ");
	fprintf(fw,"CLASSI <br>\ n");
	fprintf (fw,"<table BORDER=2>\ n");
	fprintf (fw,"<tr bgcolor=orange><td>Classe</td>\ n");
```

```
	for (i=0;i<ORE;i++) fprintf(fw,"<td width=15px> %d </td>",i);
	fprintf(fw,"</tr>");
	for (i=0;i<nclas-2;i++) {

		for(j=1;j<2;j++) {
			fprintf (fw,"<tr bgcolor=#%s><td>%s</td>\ n",j%2?"ffffff":"dddddd",_Corrispondenze[CLASSI][i]);
			for (z=0;z<ORE;z++) {
				if (j%2)
					fprintf (fw,"<td bgcolor=#%s > %s
</td>\ n",(z/6)%2?"ffffff":"dddddd",_Corrispondenze[PROF][(*cattedre)[(*orario_classi)[i].cat0[z]].prof]);
				else
					fprintf (fw,"<td bgcolor=#%s > %s
</td>\ n",(z/6)%2?"c0c0c0":"969696",_Corrispondenze[PROF][(*cattedre)[(*orario_classi)[i].cat0[z]].prof]);
			}
			fprintf (fw,"</tr>\ n");
		}
	}
	fprintf (fw,"</table>\ n");
	fprintf (fw,"<table BORDER=2>\ n");
	fprintf (fw,"<tr bgcolor=orange><td>Classe</td>\ n");
	for (i=0;i<ORE;i++) fprintf(fw,"<td width=15px> %d </td>",i);
	fprintf(fw,"</tr>");
	for (i=0;i<nclas-2;i++) {
		fprintf (fw,"<tr bgcolor=#%s><td>%s</td>\ n",j%2?"ffffff":"dddddd",_Corrispondenze[CLASSI][i]);
		for (z=0;z<ORE;z++) {
			if (j%2)
				fprintf (fw,"<td bgcolor=#%s > %s
</td>\ n",(z/6)%2?"ffffff":"dddddd",(*orario_classi)[i].lab[z]!=LAB_NONE?_Corrispondenze[LABORATORI][(*orario_classi)[i].lab[z]
-1]:"---");
			else
				fprintf (fw,"<td bgcolor=#%s > %s
</td>\ n",(z/6)%2?"c0c0c0":"969696",(*orario_classi)[i].lab[z]!=LAB_NONE?_Corrispondenze[LABORATORI][(*orario_classi)[i].lab[z
]-1]:"---");
		}
		fprintf (fw,"</tr>\ n");

	}
	fprintf (fw,"</table>\ n");
	fprintf (fw,"<br><br><br>");
	for (j=0;j<2;j++) {
		fprintf (fw,"<table BORDER=1>\ n");
		fprintf (fw,"<tr bgcolor=orange><td>Classe</td>\ n");
		for (z=0;z<ORE;z++)
			fprintf(fw,"<td width=15px>%d</td>",z);
		fprintf(fw,"</tr>");
		for (i=0;i<nprof;i++) {

			fprintf (fw,"<tr bgcolor=#%s><td>%s</td>\ n",j%2?"c0c0c0":"dddddd",_Corrispondenze[PROF][i]);
			for (z=0;z<ORE;z++) {
			//	printf("nprof=%d i=%d ,z=%d orario=%d \ n",nprof,i,z,(*orario_prof)[i].lab[z]);
				if (j%2)
					fprintf (fw,"<td bgcolor=#%s>
%s</td>\ n",(z/6)%2?"ffffff":"dddddd",(*orario_prof)[i].lab[z]>0?_Corrispondenze[LABORATORI][(*orario_prof)[i].lab[z]-1]:"---");
				else
					fprintf (fw,"<td bgcolor=#%s> %s
</td>\ n",(z/6)%2?"c0c0c0":"969696",(*orario_prof)[i].cat[z]>0?_Corrispondenze[CLASSI][(*cattedre)[(*orario_prof)[i].cat[z]].class
e]:"---");
			}
			fprintf (fw,"</tr>\ n");
		}
		fprintf (fw,"</table>\ n");
```

```
	}
	fprintf (fw,"</body></html>");
	fclose (fw);
}
void copia_spec (tab_classi *classi, tab_prof *prof,tab_dati_cattedre *cat){
	/* dato un orario, ne crea la copia "speculare */
	/* utile per esplorare una nuova zona di dominio (ancora da verificare) */
	int i,j;
	tab_classi tmp;
	tab_classi *orario_spec=(tab_classi *) malloc(sizeof(tab_classi));;
	(*orario_spec)=_classi_malloc(nclas-2);

	for (i=0; i<nclas-2; i++) {
		for (j=0; j<ORE; j++) {
			(*orario_spec)[i].cat0[ORE-1-j] = (*classi)[i].cat0[j];
			(*orario_spec)[i].lab[ORE-1-j] = (*classi)[i].lab[j];
			(*orario_spec)[i].cat1[ORE-1-j] = (*classi)[i].cat1[j];
		}
	}
	tmp=(*orario_spec);
	(*orario_spec)=(*classi);
	(*classi)=tmp;
	free(*orario_spec);
	azzeraprof(prof);
	classi2prof(classi,prof,cat);
	//stampastrutturadati("prova2.html",classi,prof);
	//poi bisogna chiamare classi2prof
}
unsigned long int dssize() {
	return ncattedre*sizeof(struct catt_st) + nprof*sizeof(struct prof_st) + (nclas-2)*sizeof(struct classi_st) ;
//+ nmat*(nclas-2)*GIORNI*sizeof(int)+ nprof*GIORNI*sizeof(int);
}
char * salvatutto(tab_prof *orario_prof, tab_classi *orario_classi,tab_dati_cattedre *cattedre) {
	char *toret;
	int i,j,k;
	toret=(char *) malloc(dssize());
	j=0;
	for (i=0; i<ncattedre; i++,j+=sizeof(struct catt_st))
		memcpy(&toret[j], &(*cattedre)[i], sizeof(struct catt_st));
	for (i=0; i<nprof; i++, j+=sizeof(struct prof_st))
		memcpy(&toret[j], &(*orario_prof)[i], sizeof(struct prof_st));
	for (i=0; i<nclas-2; i++, j+=sizeof(struct classi_st))
		memcpy(&toret[j], &(*orario_classi)[i], sizeof(struct classi_st));
/*	for (i=0; i<nmat; i++)
		for (k=0; k<nclas-2; k++, j+=GIORNI*sizeof(int))
			memcpy(&toret[j], &ore_mgiornaliere[i][k], GIORNI*sizeof(int));
	for (i=0; i<nprof;i++,j+=GIORNI*sizeof(int))
		memcpy(&toret[j],&ore_pgiornaliere[i], GIORNI*sizeof(int));*/
	return toret;
}
void confrontasd(char *a, char *b ) {
	int i,j;
	int checked=0;
	struct prof_st *p1,*p2;
	struct catt_st *c1,*c2;
	struct classi_st *cl1,*cl2;
	if (memcmp(a,b,ncattedre*sizeof(struct catt_st))!=0){
		for (i=0; i<ncattedre; i++)
			if (memcmp(&a[i*sizeof(struct catt_st)], &b[i*sizeof(struct catt_st)],sizeof (struct catt_st))!=0) {
				c1=(struct catt_st *) &a[i*sizeof(struct catt_st)];
				c2=(struct catt_st *) &b[i*sizeof(struct catt_st)];
```

```
                                        printf ("Cattedra %d, Prima: \ n",i);
                                        printf ("prof: %d , classe %d, materia %d , maxorecon %d, marked %d\ n", c1->prof, c1->classe,
c1->materia, c1->max_ore_consec, c1->marked);
                                        printf ("prof: %d , classe %d, materia %d , maxorecon %d, marked %d\ n", c2->prof, c2->classe,
c2->materia, c2->max_ore_consec, c2->marked);
                                }
                        printf ("Cattedre ... NO!\ n");
                        getchar();
                }
                checked=ncattedre*sizeof(struct catt_st);
                if (memcmp(&a[checked], &b[checked], nprof*sizeof(struct prof_st))!=0){
                        printf("Prof .. NO\ n");
                        getchar();
                        for (i=0; i<nprof; i++)
                                if (memcmp (&a[checked +i*sizeof(struct prof_st)], &b[checked +i*sizeof(struct prof_st)],
sizeof(struct prof_st))!= 0) {
                                        p1=(struct prof_st *) &a[checked+i*sizeof(struct prof_st)];
                                        p2=(struct prof_st *) &b[checked+i*sizeof(struct prof_st)];
                                        printf ("p1 ... \ n");
                                        for (j=0; j<ORE; j++){
                                                if (p1->lab[j] != p2->lab[j])
                                                        printf (" LAb %d ->Prima : %d , Dopo : %d\ n", j,p1->lab[j], p2->lab[j]);
                                                if (p1->cat[j] != p2->cat[j])
                                                        printf (" Cat %d -> Prima: %d,  Dopo : %d\ n", j, p1->cat[j], p2->cat[j]);
                                        }
                                }
                }
                checked+=nprof*sizeof(struct prof_st);
                if (memcmp(&a[checked], &b[checked], (nclas-2)*sizeof(struct classi_st))!=0){
                        printf("Classi .. NO\ n");
                        getchar();
                }
                //if (memcmp(a,b
}

void loops0(tab_classi o_c, tab_classi p_o_c, tab_prof o_p, tab_prof p_o_p, tab_dati_cattedre d_c, tab_dati_cattedre p_d_c,
tab_classi b_o_c, tab_prof b_o_p, tab_classi w_o_c, tab_prof w_o_p, tab_dati_prof d_p, int STALLO, int livello, float *t, int
*best_quality, int *worst_quality,int *prev_quality) {
                        int n=0;
                        int quality;
                        char *ds1,*ds2;
                //      FILE *fac=fopen("accepted.txt", "a");
                        while (n<STALLO) {
                                if( scambia(&o_c, &o_p,&d_c, livello) == RIUSCITO){
                                        quality = qualita(&o_c, &o_p, &d_p, &d_c);
                                        //printf("%d\ n",quality);
                                        if ( (quality < *prev_quality) || (drand48() <= pow(M_E,(*prev_quality-quality)/(*t)) )) {
                //                              fprintf(fac, "%d\ n", quality);
                                                n = 0;
                                                if (quality > q_max)
                                                        q_max=quality;
                                                if (quality < q_min)
                                                        q_min=quality;
                                                *prev_quality = quality;
                                                //last_accepted_quality = quality;
                                                copia_ore(prev_ore_pg, ore_pgiornaliere, prev_ore_mg, ore_mgiornaliere);

                                                // Se lo sambio è andato a buon fine allora svuolo lo stack degli scambi.
                        //                      printf("Svuotamento Stack\ n");
                                                _strash(Backtrack);
```

```
                                        if (quality < *best_quality) {
                                                *best_quality = quality;
                                                memcpy(b_o_c, o_c, (nclas-2)*3*ORE*sizeof(int));
                                                memcpy(b_o_p, o_p, nprof*2*ORE*sizeof(int));
                                        }
                                } else  {
                                        copia_ore(ore_pgiornaliere, prev_ore_pg, ore_mgiornaliere, prev_ore_mg);
                                        n++;
                                        if(quality > *worst_quality){
                                                memcpy(w_o_c, o_c,(nclas-2)*3*ORE*sizeof(int));
                                                memcpy(w_o_p, o_p, nprof*2*ORE*sizeof(int));
                                                *worst_quality = quality;
                                        }
                        //              confrontasd(ds1,ds2);
                        //              free(ds2);
                                //      printf("BACK\ n");
                                        // Se lo scambio non è andato a buon fine allora ripristino la tabella.
                                        backtrack ( &o_c , &o_p, &d_c);
                                //      controllo_ore_singolo_prof(&o_p, &p_o_p, &d_c);

                                }
                                (*t)=(*t)*STEP;                 // la temperatura scende
                                                // considerazione: dobbiamo farla scendere piu' velocemente?
                        } else
                                backtrack(&o_c, &o_p, &d_c);
                //      free(ds1);
                } // fine MULTIPLE_LOOP
        //      fclose(fac);
}

void loops(tab_classi o_c, tab_classi p_o_c, tab_prof o_p, tab_prof p_o_p, tab_dati_cattedre d_c, tab_dati_cattedre p_d_c,
tab_classi b_o_c, tab_prof b_o_p, tab_classi w_o_c, tab_prof w_o_p, tab_dati_prof d_p, int STALLO, int livello, float *t, int
*best_quality, int *worst_quality,int *prev_quality) {
                int n=0;
                int quality;
                char *ds1,*ds2;
                while (n<STALLO) {
                        if( scambia(&o_c, &o_p,&d_c, livello) == RIUSCITO){
                                quality = qualita(&o_c, &o_p, &d_p, &d_c);
                                if ( (quality < *prev_quality) || (drand48() <= pow(M_E,(*prev_quality-quality)/(*t)) )) {
                                        n = 0;
                                        if (quality > q_max)
                                                q_max=quality;
                                        if (quality < q_min)
                                                q_min=quality;
                                        *prev_quality = quality;
                                        //last_accepted_quality = quality;
                                        copia_ore(prev_ore_pg, ore_pgiornaliere, prev_ore_mg, ore_mgiornaliere);

                                        // Se lo sambio è andato a buon fine allora svuolo lo stack degli scambi.
                        //              printf("Svuotamento Stack\ n");
                                        _strash(Backtrack);

                                        if (quality < *best_quality) {
                                                *best_quality = quality;
                                                memcpy(b_o_c, o_c, (nclas-2)*3*ORE*sizeof(int));
                                                memcpy(b_o_p, o_p, nprof*2*ORE*sizeof(int));
                                        }
                                } else  {
                                        copia_ore(ore_pgiornaliere, prev_ore_pg, ore_mgiornaliere, prev_ore_mg);
                                        n++;
```

```
                                        if(quality > *worst_quality){
                                                memcpy(w_o_c, o_c,(nclas-2)*3*ORE*sizeof(int));
                                                memcpy(w_o_p, o_p, nprof*2*ORE*sizeof(int));
                                                *worst_quality = quality;
                                        }
                //                      confrontasd(ds1,ds2);
                //                      f ree(ds2);
                        //              printf("BACK\ n");
                                        // Se lo scambio non è andato a buon fine allora ripristino la tabella.
                                        backtrack ( &o_c , &o_p, &d_c);
                                //      controllo_ore_singolo_prof(&o_p, &p_o_p, &d_c);

                                }
                                (*t)=(*t)*STEP;                 // la temperatura scende
                                                // considerazione: dobbiamo farla scendere piu' velocemente?
                        } else
                                backtrack(&o_c, &o_p, &d_c);
                //      free(ds1);
                } // fine MULTIPLE_LOOP
}
```

13. Ringraziamenti

Ringrazio il professor Carrer che ha dato l’idea del programma e ci ha dato le basi della programmazione parallela, inoltre ringrazio tutti i componenti della scuola di calcolo parallelo

- Sartorello Enrico 5 c
- Bucciol Antonio 4 c
- Masarin Fabio 5 a

Per aver contribuito alla realizzazione del programma.

Inoltre ringrazio i professori preparatori della squadra nazionale delle olimpiadi di informatica per avermi dato moltissime basi e conoscenze avanzate della programmazione.

Ringrazio il professore Simionato per la disponibilità e pazienza nella correzione dei problemi linguistici.

Ringrazio infine i professori della 5b per aver capito la mia situazione scolastica nel periodo pre olimpiadi.