Università degli Studi di Firenze
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in
Scienze dell'Informazione

Metaeuristiche per la costruzione degli orari dei corsi universitari

Tesi di Laurea di

Max Manfrin

Relatore:

Prof. Pierluigi Crescenzi

Correlatori:

Prof. Marco Dorigo
Ing. Mauro Birattari

Anno Accademico 2001/02

Sommario

Il lavoro di tesi è inquadrato nell'attività svolta dal *Metaheuristics Network*, una rete di addestramento alla ricerca finanziata dal programma *Improving Human Potential* della Comunità Europea. Abbiamo studiato, implementato e confrontato cinque metaeuristiche per la risoluzione del problema della Compilazione degli Orari dei Corsi Universitari: Ant Colony Optimization, Computazione Evolutiva, Ricerca Locale Iterata, Tabu Search e Simulated Annealing.
I contributi originali sono:

- La realizzazione della prima implementazione in letteratura di un approccio Ant Colony Optimization al problema della Compilazione degli Orari dei Corsi Universitari.

- L'implementazione di F-Race, una metodologia sperimentale per la configurazione automatica delle metaeuristiche.

- I risultati dei confronti delle cinque metaeuristiche sono stati presentati alla conferenza internazionale PATAT 2002. Tale lavoro (Rossi-Doria *et al.*, 2002a) è stato selezionato per la pubblicazione in un volume della collana *Lecture Notes in Computer Science*.

- I risultati del confronto di due implementazioni della metaeuristica Ant Colony Optimization sono stati presentati alla conferenza internazio-

nale EvoCOP 2003. Tale lavoro (Socha *et al.*, 2003) è stato selezionato per la pubblicazione nel volume 2611 della collana *Lecture Notes in Computer Science*.

I risultati mostrano che nessuna metaeuristica, tra quelle implementate, è migliore su tutte le istanze del problema considerato. Inoltre, anche quando le istanze sono molto simili, dal punto di vista del generatore che le produce, non è possibile prevedere quale sarà la metaeuristica migliore, anche se emergono alcune tendenze quando ci concentriamo su istanze di una particolare classe. I risultati evidenziano quanto sia difficile trovare la migliore metaeuristica, anche per classi molto ristrette del problema.

Ringraziamenti

Ringrazio il Prof. Pierluigi Crescenzi per l'opportunità che mi ha dato di andare a svolgere la tesi a Bruxelles e per avermi seguito dall'Italia, oltre che il Prof. Marco Dorigo e il Prof. Hugues Bersini per avermi accolto presso il laboratorio di ricerca IRIDIA (Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle) e per avermi offerto tutto l'indispensabile e non solo. Un ringraziamento particolare merita Mauro Birattari che ha seguito tutto il lavoro di stesura e con cui ho collaborato strettamente, in particolar modo sulla parte relativa alla configurazione delle metaeuristiche.

Un ringraziamento lo devo anche a tutti i partecipanti del progetto *Metaheuristics Network*, con i quali ho condiviso gli entusiasmi e le fatiche del lavoro realizzato, in particolare ringrazio Joshua Knowles, Michael Sampels, Christian Blum e Krzysztof Socha per le fruttuose conversazioni e gli innumerevoli scambi di idee.

Sono debitore con il personale di IRIDIA per aver dimostrato sempre disponibilità e simpatia nei miei confronti. La mia permanenza a Bruxelles è stata fantastica soprattutto per la gente che mi ha circondato: Carlotta Piscopo, Shervin Nouyan, Vito Trianni, Halva Labella, Erol Sahin, Mark Zlochin,

Elio Tuci, Gianni Di Caro, Roderich Groß, Morgan Tamplin, Bruno Marchal.

La tesi è il culmine di anni di studio; per questo vorrei ringraziare coloro che hanno condiviso con me questa "avventura" universitaria. In particolare i colleghi Riccardo Scorretti, Fabrizio Dini, Gianni Bagni, Marco Lastri, Gaia Innocenti, Francesca Uncini, Sara Capecchi, Roberto Daniello, Elena Scapecchi, Leonardo Sabatino, Paolo Lollini, Leonardo Ricci, Andrea Fornari e Daniele Falassi (il compagno di studi migliore che abbia mai avuto), oltre che tutti gli amici che mi hanno sostenuto in questi anni, nonostante la maggior parte, avesse il deprecabile vizio di domandare quando, finalmente, mi sarei laureato. Un grazie particolare a Jacopo De Luca, Raffaella Di Bari, Federica Paoletti, Stefano Dabizzi, Barbara Palazzi e Fabio Sonnati, oltre ad alcuni amici che, pur trovandosi lontano, mi sono comunque stati vicini, tra cui Fabiana Ceol, Silvia Scarabelli, Nelly Askaner, Silaine Lima, Bill Matthias e Milena Rossi.

Un grazie di cuore alla mia famiglia, in particolare a mia sorella Ester. Oltre ad avermi mantenuto agli studi per un periodo di tempo che solo il sistema italiano può richiedere, hanno sempre mostrato fiducia per quello che faccio, e, nonostante tutte le difficoltà, mi hanno sempre spronato ad andare avanti per la mia strada.

Questa tesi è dedicata alla memoria di mia madre, Carla, che è mancata due anni fa a causa di una rara malattia purtroppo ancora oggi incurabile. Da lei ho ricevuto l'educazione, l'esempio e l'affetto che più di tutti mi hanno aiutato a diventare quello che sono.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco degli Algoritmi

Introduzione

Il lavoro che presentiamo in questa tesi é stato svolto all'interno dell'attività del *Metaheuristics Network*,[1] una rete di addestramento alla ricerca finanziata dal programma *Improving Human Potential* della Comunità Europea.[2] Composta inizialmente da sei partecipanti, coinvolge attualmente cinque istituzioni:

- **IRIDIA - Univeristé Libre de Bruxelles - Bruxelles - Belgio**
- **INTELLEKTIK - Technische Universität Darmstadt - Darmstadt - Germania**
- **ECRG - Napier University - Edimburgo - Regno Unito**
- **IDSIA - Manno - Svizzera**
- **ANTOPTIMA - Lugano - Svizzera** è entrata nella rete nel 2001.

Due istituti hanno lasciato la rete rispettivamente dopo uno e due anni:

- **COG - Technische Universiteit Eindhoven - Eindhoven - Paesi Bassi** ha lasciato la rete il 31.08.2001;
- **EUROBIOS - Parigi - Francia** ha lasciato la rete il 31.08.2002.

[1] `http://www.metaheuristics.org`

[2] Contratto numero HPRN-CT-1999-00106.

Gli obiettivi del *Metaheuristics Network* sono di tre tipi: scientifici, ingegneristici e di addestramento.

- Obiettivi scientifici: il principale obiettivo scientifico del *Metaheuristics Network* è quello di migliorare la comprensione sul funzionamento delle metaeuristiche attraverso la ricerca teorica e sperimentale. Altri obiettivi scientifici comprendono la definizione di nuove ed efficienti metaeuristiche ibride, ossia, metaeuristiche che combinano componenti prese dai metodi esistenti, oltre alla definizione e l'uso di una metodologia sperimentale strettamente controllata, indipendente dalla particolare infrastruttura informatica, che permetta di eseguire dei confronti equi dei risultati sperimentali.

- Obiettivi ingegneristici: il primo obiettivo di tipo ingegneristico consiste nella definizione di alcune linee guida che possano essere utili nella scelta di una particolare metaeuristica, o di una particolare componente, quando si affronta un nuovo problema. Un secondo obiettivo della rete consiste nel verificare e validare le idee sviluppate su problemi presi dal mondo reale.

- Obiettivi di addestramento: i principali obiettivi di addestramento della rete sono (i) far studiare ai giovani ricercatori le tecniche delle metaeuristiche oltre che un numero di importanti problemi di ottimizzazione, (ii) insegnare ai giovani ricercatori a progettare ed eseguire gli esperimenti in modo rigoroso, e (iii) sviluppare nei giovani ricercatori abilità nella gestione di progetti per mezzo della partecipazione ad attività di ricerca internazionale.

Durante il periodo Marzo 2002 - Marzo 2003 il *Metaheuristics Network* ha studiato il problema della Compilazione degli Orari dei Corsi Uni-

VERSITARI, un problema che le università di tutto il mondo si trovano a dover fronteggiare ogni anno, e di cui ci occupiamo in questa tesi.

Una formulazione generale prevede che un certo numero di eventi (lezioni, esercitazioni, laboratori) debba essere assegnato ad un certo numero di intervalli temporali, così che siano rispettati un insieme di vincoli. I vincoli sono usualmente classificati in vincoli inderogabili e vincoli derogabili. I vincoli inderogabili sono vincoli che non possono essere violati in nessun caso, ad esempio uno studente non può seguire due corsi contemporaneamente. I vincoli derogabili sono vincoli che preferiremmo fossero soddisfatti, ma che possono essere accettati nell'orario, penalizzando ogni loro violazione, ad esempio uno studente che segue un solo evento in un giorno. Il problema generale della COMPILAZIONE DEGLI ORARI DEI CORSI UNIVERSITARI è stato dimostrato essere $\mathcal{NP}$-hard, così come molti dei sottoproblemi associati ai vincoli addizionali. Occorre tenere presente che istituti diversi hanno, in genere, vincoli diversi, in numero e in tipologia, in funzione delle strutture e dei corsi di studio che offrono, pertanto, le soluzioni algoritmiche proposte per questo problema sono, di solito, specifiche per una particolare classe di vincoli.

Il *Metaheuristics Network* ha considerato una particolare riduzione del problema della COMPILAZIONE DEGLI ORARI DEI CORSI UNIVERSITARI, e ha proposto il confronto delle prestazioni di cinque metaeuristiche, nelle stesse condizioni sperimentali, su istanze di questa riduzione del problema. Le metaeuristiche studiate includono: Ant Colony Optimization (ACO), Computazione Evolutiva (CE), Ricerca Locale Iterata (RLI), Simulated Annealing (SA) e Tabu Search (TS). Per garantire confronti equi e stesse condizioni sperimentali, le implementazioni di tutti gli algoritmi fanno uso della stessa rappresentazione diretta delle soluzioni e dello spazio di ricerca (tramite la

definizione di una struttura di vicinato comune e di una routine di ricerca locale comune). Inoltre, tutti gli algoritmi sono stati implementati usando lo stesso linguaggio di programmazione, facendo uso di una stessa libreria di oggetti comuni, e sono stati compilati nello stesso ambiente.

Abbiamo implementato F-Race (Birattari *et al.*, 2002), una procedura per la selezione automatica di una buona configurazione per le metaeuristiche, e l'abbiamo applicata a due algoritmi tra i quattro implementati da noi per la risoluzione del problema della Compilazione degli Orari dei Corsi Universitari.

Dunque, abbiamo lavorato ad un ciclo completo per la risoluzione di un problema di ottimizzazione combinatoria mediante l'uso di metaeuristiche. Ci teniamo a sottolineare che, essendo lo scopo della tesi il confronto di diverse metaeuristiche nelle stesse condizioni sperimentali, una maggiore libertà nell'uso della rappresentazione delle soluzioni, nell'uso delle informazioni euristiche, di un diverso linguaggio di programmazione con diverse strutture dati, potrebbe dare risultati diversi da quelli che riportiamo in questa tesi.

Contributi originali

I contributi originali sono:

- Il nostro algoritmo Ant Colony System è, assieme all'implementazione di Socha *et al.* (2002) che segue le specifiche $\mathcal{MAX} - \mathcal{MIN}$ Ant System, la prima implementazione di un approccio ACO al problema della Compilazione degli Orari dei Corsi Universitari, apparsa in letteratura.

- Abbiamo implementato e testato F-Race (Birattari *et al.*, 2002), una metodologia sperimentale ideata specificatamente per tenere conto delle

caratteristiche peculiari del problema di configurazione delle metaeuristiche.

- I risultati del confronto delle prestazioni delle cinque metaeuristiche implementate all'interno del *Metaheuristics Network* sono stati presentati alla conferenza internazionale PATAT 2002. Tale lavoro (Rossi-Doria *et al.*, 2002a) è stato selezionato per la pubblicazione in un prossimo volume della collana *Lecture Notes in Computer Science*.

- I risultati del confronto delle prestazioni di $\mathcal{MAX} - \mathcal{MIN}$ Ant System e Ant Colony System, sono stati presentati alla conferenza internazionale EvoCOP 2003. Tale lavoro (Socha *et al.*, 2003) è stato selezionato per la pubblicazione nel volume 2611 della collana *Lecture Notes in Computer Science*.

Struttura della tesi

La tesi è strutturata come segue:

- Nel Capitolo 1 si fornisce una breve introduzione alla complessità computazionale e una definizione formale di problema di ottimizzazione. Successivamente si esegue una breve panoramica sul problema della Compilazione degli orari, fino a giungere alla Compilazione degli Orari dei Corsi Universitari, del quale sono illustrate la formulazione classica e la riduzione da noi usata per questa tesi.

- Nella prima parte del Capitolo 2 si introduce il concetto di metaeuristiche, se mostra una classificazione e si illustra l'attuale Stato dell'Arte. Successivamente si descrivono le linee guida seguite per l'implementazione delle metaeuristiche, e si illustrano le nostre implementazioni per

la risoluzione del problema della Compilazione degli Orari dei Corsi Universitari.

- Nel Capitolo 3 si introduce il problema della configurazione delle metaeuristiche, dandone una definizione formale. Si illustrano le idee generali che stanno dietro ai metodi di *racing* e si presenta F-Race, un metodo ideato esplicitamente per tenere conto delle caratteristiche peculiari del problema di configurazione delle metaeuristiche. L'ultima parte del capitolo descrive gli esperimenti effettuati con F-Race per la configurazione di due algoritmi tra quelli da noi implementati per risolvere il problema della Compilazione degli Orari dei Corsi Universitari.

- Nel Capitolo 4 si illustrano gli esperimenti, effettuati nelle stesse condizioni sperimentali, inizialmente per il confronto delle metaeuristiche implementate all'interno del *Metaheuristics Network*, e successivamente esteso alle nostre implementazioni.

- Nelle Conclusioni si analizzano i risultati ottenuti mediante i diversi approcci studiati e si presentano alcuni possibili sviluppi futuri.

Capitolo 1

Definizione del problema

Adottando la terminologia presente in Ausiello *et al.* (1999), con il termine *problema* ci riferiamo informalmente ad una domanda generale a cui si deve rispondere, di solito con parametri e variabili con valori non specificati. Il termine *istanza* si riferisce al problema per il quale sono stati specificati i valori dei parametri e delle variabili. In generale, si esprime il problema in termini di una qualche *relazione matematica* $R \subseteq I \times S$, dove I è l'insieme delle istanze del problema e S è l'insieme delle soluzioni del problema. Alternativamente, si può considerare un predicato $P(x, s)$ che è vero se e solo se $(x, s) \in R$.

Un problema può essere formulato in tre differenti versioni:

Problema di decisione: si vuole determinare se una istanza $x \in I$ soddisfi o meno una certa condizione, ossia, se $Q(x)$ è verificato, dove Q è uno specifico predicato (unario). In questo caso, la relazione R si riduce ad una funzione $f : I \to S$, dove S è l'insieme binario $S = \{si, no\}$.

Problema di ricerca: data una qualunque istanza $x \in I$, si trovi una soluzione $s \in S$, tale che $(x, s) \in R$ è verificato;

Problema di ottimizzazione: data un'istanza $x \in I$, si trovi il valore della "migliore" soluzione s_{opt} (rispetto ad una qualche misura), tra tutte le soluzioni $s \in S$ tali che $(x, s) \in R$ è verificato.

In generale, siamo interessati a trovare l'algoritmo più *efficiente* per la risoluzione di un problema, dove per efficiente si intende, in senso generale, quello che abbisogna delle minori risorse. Poiché il tempo impiegato è spesso un fattore dominante ai fini pratici, normalmente con algoritmo più efficiente si intende l'algoritmo più veloce. Di questo si occupa la teoria della complessità computazionale (Garey e Johnson, 1979) ed in particolar modo, la teoria della $\mathcal{NP}$-completezza.

La complessità temporale di un algoritmo è misurata da una funzione che ritorna il massimo tempo di esecuzione di cui l'algoritmo necessita per risolvere istanze del problema di una data *dimensione*, dove la dimensione di un'istanza riflette la quantità di simboli necessari a descrivere l'istanza. Si assume quindi che esista un qualche *schema di codifica* usato per descrivere l'istanza sottoforma di stringa di caratteri su un qualche alfabeto finito. Poiché l'uso di schemi di codifica diversi produce, in generale, stringhe di lunghezza diversa per la stessa istanza, si assume che ad ogni problema sia associato un prefissato schema di codifica che mappa le istanze del problema nelle stringhe che le descrivono.

Un algoritmo opera in tempo polinomiale, se la sua complessità temporale è $O(p(n))$[1] per qualche polinomio p, dove n indica la dimensione dell'istanza; nel caso in cui non sia possibile trovare una tale limitazione, si dice che l'algoritmo opera in tempo esponenziale.

[1] Siano f e g due funzioni da $\mathbf{N} \to \mathbf{N}$; scriveremo $f(n) = O(g(n))$ se esistono due interi positivi c e n_0 tali che $\forall n > n_0$, $f(n) \leq c \cdot g(n)$.

Se il numero di passi necessari alla risoluzione dell'istanza cresce in modo più che polinomiale, si dice che il problema è *intrinsecamente difficile*. Spesso ci si riferisce a tali problemi come a problemi intrattabili da un punto di vista pratico. Per tali problemi dobbiamo spesso limitarci a calcolare una soluzione che, pur non essendo quella ottimale, è vicina (rispetto ad una qualche misura) all'ottimo, e può essere determinata in tempo polinomiale.

$\mathcal{NP}$-completezza

La teoria della $\mathcal{NP}$-completezza formalizza la distinzione tra problemi facili e difficili. Per far questo si usa considerare la formulazione come problema di decisione. Le conclusioni tratte non sono limitate da questo fatto poiché il problema di ricerca (e quindi il problema di ottimizzazione) non è più facile da risolvere del problema di decisione, e se il problema di ricerca può essere risolto grazie ad un algoritmo efficiente (nel senso generale detto prima, per cui abbisogna di poche risorse), allora lo stesso si può dire per il problema di decisione.

La teoria della $\mathcal{NP}$-completezza divide i problemi in due classi. La prima è la classe $\mathcal{P}$ dei problemi trattabili.

Definizione 1.1 *La classe $\mathcal{P}$ è la classe dei problemi di decisione che possono essere risolti da un algoritmo in tempo polinomiale.*

La classe $\mathcal{NP}$ può essere definita informalmente per mezzo degli algoritmi non-deterministici. Data un'istanza $x \in I$, un algoritmo non-deterministico può essere pensato composto di una parte in cui si *propone* una soluzione. La soluzione è quindi *controllata* nella seconda parte da un algoritmo deterministico polinomiale. La classe $\mathcal{NP}$ è la classe di problemi le cui soluzioni sono verificabili in tempo polinomiale.

Definizione 1.2 *La classe $\mathcal{NP}$ è la classe dei problemi di decisione che possono essere risolti da un algoritmo non-deterministico in tempo polinomiale.*

Un problema di decisione che può essere risolto da un algoritmo deterministico in tempo polinomiale può essere risolto anche da un algoritmo non-deterministico in tempo polinomiale: pertanto, $\mathcal{P} \subseteq \mathcal{NP}$. Probabilmente una delle più importanti questioni aperte in informatica teorica è decidere se $\mathcal{P} = \mathcal{NP}$. È largamente diffusa l'opinione che $\mathcal{P} \neq \mathcal{NP}$, nonostante non sia stata fornita ancora alcuna prova per questa congettura.

Poiché non è possibile mostrare che $\mathcal{NP} \backslash \mathcal{P}$ è non vuoto, a meno che sia fornita una prova per $\mathcal{P} \neq \mathcal{NP}$, la teoria della $\mathcal{NP}$-completezza fornisce risultati in una forma debole, sotto l'ipotesi di $\mathcal{P} \neq \mathcal{NP}$.

Definizione 1.3 *Un problema* Π *è* riducibile in tempo polinomiale *al problema* Π'*, se esiste un algoritmo che trasforma ogni istanza di* Π *in un'istanza di* Π' *in tempo polinomiale e che, per ogni istanza di* Π*, risponde a tale istanza "si" se e solo se la risposta della corrispondente istanza di* Π' *è "si".*

In modo informale questa definizione dice che se Π può essere ridotto a Π' in tempo polinomiale, allora il problema Π' è almeno altrettanto difficile da risolvere quanto il problema Π. Usando la nozione di riducibilità in tempo polinomiale possiamo procedere a definire la classe dei problemi $\mathcal{NP}$-completi.

Definizione 1.4 *Un problema* Π *è $\mathcal{NP}$-completo se e solo se* $\Pi \in \mathcal{NP}$ *e* $\forall$ $\Pi' \in \mathcal{NP}$ *vale che* Π' *è riducibile in tempo polinomiale a* Π.

La classe dei problemi $\mathcal{NP}$-completi è in un certo senso la classe dei problemi *più difficili* appartenenti a $\mathcal{NP}$. Se un problema $\mathcal{NP}$-completo può essere risolto da un algoritmo in tempo polinomiale, allora tutti i problemi in $\mathcal{NP}$

possono essere risolti in tempo polinomiale. Comunque, ad oggi, nessun problema $\mathcal{NP}$-completo è stato risolto da un algoritmo in tempo polinomiale. Nonostante non sia stato provato, ad oggi, che i problemi $\mathcal{NP}$-completi non possano essere risolti per mezzo di algoritmi che operano in tempo polinomiale, ci sono forti indizi a favore della possibilità che tali algoritmi non esistano. Da tutto questo si può derivare che, provata l'appartenenza di Π alla classe dei problemi $\mathcal{NP}$-completi, non esiste un modo efficiente di risolvere Π. Ad oggi molti problemi sono stati mostrati essere $\mathcal{NP}$-completi: un elenco lo si trova in Garey e Johnson (1979).

Come abbiamo osservato, i concetti base della teoria della complessità sono stati introdotti facendo riferimento ai problemi di decisione. Per quanto detto fin'ora, il problema di ottimizzazione non è più facile del problema di decisione associato. Pertanto, provare che il problema di decisione è $\mathcal{NP}$-completo implica che il problema di ottimizzazione sia difficile da risolvere. Problemi che sono difficili quanto problemi $\mathcal{NP}$-completi, ma non necessariamente elementi di $\mathcal{NP}$ sono detti problemi $\mathcal{NP}$-hard.

Definizione 1.5 *Un problema* Π *è* $\mathcal{NP}$*-hard se e solo se* $\forall\ \Pi' \in \mathcal{NP}$ *vale che* Π' *è riducibile in tempo polinomiale a* Π.

Pertanto, ogni problema $\mathcal{NP}$-completo è anche un problema $\mathcal{NP}$-hard. Dall'altra parte, se il problema di ottimizzazione combinatoria formulato come problema di decisione è $\mathcal{NP}$-completo, allora la formulazione come problema di ottimizzazione è $\mathcal{NP}$-hard.

Diamo una definizione formale di problema di ottimizzazione.

Definizione 1.6 *Un* problema di ottimizzazione Π_{opt} *è caratterizzato dalla seguente quadrupla di oggetti (I,SOL,m,goal) tale che:*

1. *I è l'insieme delle* istanze *di Π_{opt};*

2. *SOL è una funzione che associa ad una qualunque istanza $x \in I$, l'insieme delle* soluzioni ammissibili *di x;*

3. *Data un'istanza $x \in I$ ed una soluzione ammissibile $s \in SOL(x)$, $m(x, s) \in \mathbf{R}$ indica la misura di s. La funzione m è chiamata anche* funzione obiettivo.

4. *goal $\in$ {MAX,MIN} specifica se Π_{opt} è un problema di massimizzazione o minimizzazione.*

Data un'istanza x, identifichiamo con $SOL_{opt}(x)$ l'insieme di tutte le *soluzioni ottime* di x. Una soluzione ottima $s_{opt}(x) \in SOL_{opt}(x)$ è tale che $m(x, s_{opt}(x)) = goal\{m(x, z)|z \in SOL(x)\}$. Il valore di una soluzione ottima $s_{opt}(x)$ in x sarà denotato con $m_{opt}(x)$.

È importante notare che ogni problema di ottimizzazione Π_{opt} ha un problema di decisione associato Π_{opt_D}. Nel caso in cui Π_{opt} sia un problema di minimizzazione, Π_{opt_D} chiede se, data un'istanza $x \in I$ e un qualche valore $K > 0$, esiste una soluzione ammissibile $s(x) \in SOL(x)$ per cui valga $m(x, s(x)) \leq K$. Per i problemi di massimizzazione la definizione è analoga, con $m(x, s(x)) \geq K$. Oltre al problema di decisione, Π_{opt} ha associato anche un problema di valutazione Π_{opt_V}, il quale chiede, data un'istanza $x \in I$, quale sia il valore della soluzione ottima $m_{opt}(x)$.

Più precisamente, possiamo dire che la definizione di un problema di ottimizzazione Π_{opt} conduce alle formulazioni dei tre seguenti problemi:

- **Problema di costruzione (Π_{opt_C}):** data un'istanza $x \in I$, si trovi una soluzione ottima $s_{opt}(x) \in SOL_{opt}(x)$ e la sua misura $m_{opt}(x)$.

- **Problema di valutazione (Π_{opt_V}):** data un'istanza $x \in I$, si trovi il valore della soluzione ottima $m_{opt}(x)$.
- **Problema di decisione (Π_{opt_D}):** data un'istanza $x \in I$ e un valore $K \in \mathbf{R}^+$, si dica se $m_{opt}(x) \leq K$ (se $goal = MIN$) o se $m_{opt}(x) \geq K$ (se $goal = MAX$).

Si noti che, per ogni problema di ottimizzazione Π_{opt}, il corrispondente problema di decisione Π_{opt_D} non è più difficile del corrispondente problema di costruzione Π_{opt_C}. Infatti, data un'istanza x, per rispondere al problema Π_{opt_D} è sufficiente eseguire un qualche algoritmo che risolva Π_{opt_C}, ottenendo in tal modo una soluzione ottima $s_{opt}(x)$ e la sua misura $m_{opt}(x)$; è quindi sufficiente controllare se $m(x, s_{opt}(x)) \leq K$ (oppure $\geq K$).

Il resto del capitolo è organizzato come segue: il Paragrafo 1.1 introduce il problema della Compilazione degli Orari; il Paragrafo 1.2 introduce il problema della Compilazione degli Orari Scolastici; infine il Paragrafo 1.3 introduce il problema della Compilazione degli Orari dei Corsi Universitari.

1.1 La compilazione degli orari

Secondo il vocabolario della lingua italiana Treccani[2] con *orario* si intende:

1. *predisposizione dell'ordine in cui determinati avvenimenti, atti o operazioni debbono succedersi nel tempo, con indicazione dell'ora in cui hanno inizio e termine singolarmente o nel loro insieme*;
2. *con significato più concreto, il prospetto che illustra l'orario predisposto.*

[2]Istituto della Enciclopedia Italiana fondata da Giovanni Treccani, 1989

Le varie tipologie di orari esistenti giocano un ruolo importante nella nostra società: si pensi, ad esempio, ai mezzi di trasporto quali aerei, treni, navi e autobus, oppure agli istituti scolastici. Non stupisce quindi che una sempre crescente quantità di risorse sia stanziata per la loro compilazione. Questo è maggiormente vero per quelle aziende che possono operare in modo più efficiente ottimizzando la gestione temporale delle loro attività. Anche per questo la Compilazione degli Orari è uno dei problemi storici della Ricerca Operativa. Nella Compilazione degli Orari devono essere rispettati un certo numero di criteri, spesso identificati in letteratura come **vincoli inderogabili**, e nel contempo si deve minimizzare il numero di violazioni a criteri di minore importanza, detti anche **vincoli derogabili**.[3] Esempi di vincoli inderogabili sono l'uso non contemporaneo da parte degli aerei di una stessa pista per l'atterraggio o il decollo, oppure, nel caso degli orari scolastici, il fatto che un docente non possa tenere più di una lezione nello stesso momento. I vincoli derogabili sono invece caratteristiche che si vorrebbe non violare: ad esempio, avere una frequenza maggiore per i treni usati dai pendolari per recarsi e tornare dal lavoro, oppure assegnare un numero basso di ore di lezione consecutive agli studenti universitari.

Compilare manualmente gli orari richiede di solito un grosso dispendio di tempo. In situazioni reali, dove i vincoli inderogabili possono mutare abbastanza di frequente (si pensi, per esempio, ad un binario, una pista oppure un'aula che diventano inagibili per un certo intervallo di tempo) è spesso impossibile adattare il vecchio orario; pertanto, occorre compilarne uno nuovo. Per di più gli orari costruiti manualmente spesso non soddisfano alcuni criteri

[3] La maggior parte della letteratura sulla compilazione degli orari è in lingua inglese, dove si usa indicare i criteri con i termini *hard constraints* e *soft constraints*. Da questo punto in avanti useremo in modo consistente i rispettivi termini italiani *vincoli inderogabili* e *vincoli derogabili*.

(si pensi a quegli orari dei corsi universitari che prevedono due insegnamenti in contemporanea impedendo agli studenti di seguirli entrambi). Proprio per questo, durante gli ultimi quarant'anni la compilazione automatica degli orari è stata oggetto di una sempre crescente attenzione da parte dei ricercatori, che hanno cominciato ad applicare anche tecniche quali, ad esempio, gli Algoritmi Genetici, il Tabu Search ed il Simulated Annealing.

Nel 1995 ha preso il via una serie di conferenze internazionali biennali sulle tecniche di compilazione automatica degli orari: *The International Series of Conferences on the Practice and Theory of Automated Timetabling*[4] - PATAT (Burke *et al.*, 1995; Burke e Carter, 1997; Burke e Erben, 2000; Burke e De Causmaecker, 2002). Una menzione particolare merita anche l'associazione EURO (Association of European Operational Research Societies) di cui fa parte un gruppo di lavoro, EURO-WATT (EURO Working Group on Automated Timetabling), che si incontra una volta all'anno, gestisce un bollettino via posta elettronica e mantiene un sito web[5] con molte informazioni di interesse sui problemi di creazione degli orari, quali un'estesa bibliografia e molti benchmark.

1.2 La compilazione degli orari scolastici

Il problema della Compilazione degli Orari Scolastici consiste nel determinare una sequenza di eventi (incontri tra docenti e studenti) in un prefissato periodo di tempo, soddisfacendo una serie di vincoli. In letteratura

[4] Una selezione di articoli è pubblicata nella collana *Lecture Notes in Computer Science* edita dalla Springer-Verlag (Burke e Ross, 1996; Burke e Carter, 1998; Burke e Erben, 2001).

[5] `http://www.asap.cs.nott.ac.uk/watt/`

sono state proposte diverse classificazioni. Noi illustriamo quella adottata da Bardadym (1996):

- **Orari di facoltà**: siano dati un insieme di docenti, un insieme di lezioni e le disponibilità dei docenti per insegnare certi corsi. Il problema consiste nel distribuire i corsi tra i docenti sotto specifiche condizioni.

- **Orari classe-docente**: siano dati un insieme di classi e un insieme di docenti. Per ogni coppia <classe, docente> si definisce quali lezioni e in che numero sono richieste. Ogni singola lezione deve essere assegnata ad un intervallo temporale tale che nessun docente e nessuna classe abbiano due o più lezioni contemporaneamente.

- **Orari dei corsi**: sia dato un insieme di studenti. Per ogni studente sia definito l'insieme dei corsi che deve seguire. I corsi devono essere assegnati ad intervalli temporali in modo tale che nessuno studente abbia due o più corsi contemporaneamente.

- **Orari degli esami**: siano dati un insieme di studenti e un insieme di esami per ogni studente. Ogni esame deve essere assegnato ad un intervallo temporale tale che nessuno studente debba avere due o più esami contemporaneamente.

- **Assegnazione delle aule**: quando si assegnano le lezioni classe-docente, queste devono essere assegnate alle aule a seconda della data disponibilità. Nessuna aula deve essere usata contemporaneamente da classi diverse.

Spesso la compilazione di un orario scolastico implica la risoluzione di problemi appartenenti a categorie diverse. Ad esempio, adottando la classificazione di Bardadym, un'università dovrà provvedere alla compilazione degli orari dei

corsi, degli orari degli esami, degli orari di facoltà e dell'assegnazione delle aule. Inoltre occorre tenere presente che istituti diversi hanno vincoli diversi, in funzione delle strutture. Basti pensare a quelle facoltà che hanno aule e laboratori dislocati in diversi edifici, anche distanti tra loro (un esempio reale è dato proprio dal Corso di Laurea in Informatica a Firenze). Questo può introdurre ulteriori vincoli nel problema dell'assegnazione delle aule, se lezioni consecutive prevedono di spostarsi da un edificio all'altro. Ancora, si pensi a quei corsi di laurea che permettono agli studenti di proporre un piano di studio personalizzato. Questa maggiore libertà nella scelta dei corsi da seguire può rendere addirittura impossibile la creazione di un orario con il prefissato numero di intervalli temporali dove non vi siano sovrapposizioni tra corsi che hanno studenti in comune.

Il problema della Compilazione degli Orari Scolastici è stato dimostrato essere $\mathcal{NP}$-hard, così come lo sono molti dei sottoproblemi associati (Schaerf, 1995; Cooper e Kingston, 1996; de Werra, 1997; Willemen, 2002).

1.3 La compilazione degli orari dei corsi universitari

Il problema della Compilazione degli Orari dei Corsi Universitari consiste nell'assegnare una serie di eventi (lezioni, esercitazioni, laboratori) per ogni corso avendo a disposizione un certo insieme di aule e di intervalli di tempo. Gli aspetti che caratterizzano questo problema sono dati dal fatto che ogni studente ha una propria lista di corsi da seguire: quindi, se due corsi condividono uno studente questi non possono essere assegnati allo stesso intervallo temporale, pena la violazione di un vincolo inderogabile. Un altro vincolo inderogabile è la capienza dell'aula in cui tenere il corso: infatti,

occorre assegnare alla lezione solo aule con una capienza maggiore o uguale al numero degli studenti che seguono quel corso.

Il Paragrafo 1.3.1 illustra una formulazione classica del problema, data da de Werra (1995). Il Paragrafo 1.3.2 illustra la formulazione usata all'interno del *Metaheuristics Network*. Il Paragrafo 1.3.3 specifica quali classi di istanze sono state usate nella prima fase dell'attività di ricerca del *Metaheuristics Network*.

1.3.1 Formulazione classica del problema

Sono state fornite molte formulazioni per il problema della Compilazione degli Orari dei Corsi Universitari. Qui mostriamo quella data da de Werra (1995).

Supponiamo di avere q corsi $K_1, \ldots, K_q$ e che, per ogni i, il corso K_i sia composto da k_i lezioni. Siano definiti r indirizzi $S_1, \ldots, S_r$, ossia gruppi di corsi che hanno studenti in comune. Questo significa che i corsi in S_l devono essere assegnati a diversi intervalli temporali. Sia p il numero di intervalli temporali, e sia l_k il massimo numero di lezioni che possono essere assegnate nell'intervallo temporale k (ad esempio, il numero di aule disponibili nell'intervallo temporale k). La formulazione è la seguente:

$$\text{si trovino gli} \quad y_{ik} \quad (i = 1, \ldots, q; k = 1, \ldots, p) \quad \text{tali che}$$

$$\sum_{k=1}^{p} y_{ik} = k_i \qquad i = 1, \ldots, q \tag{1.1a}$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \qquad k = 1, \ldots, p \tag{1.1b}$$

$$\sum_{i \in S_l} y_{ik} \leq 1 \qquad l = 1, \ldots, r; k = 1, \ldots, p \tag{1.1c}$$

$$y_{ik} = 0 \text{ o } 1 \qquad i = 1, \ldots, q; k = 1, \ldots, p \tag{1.1d}$$

dove $y_{ik} = 1$ indica che una lezione del corso K_i è assegnata all'intervallo temporale k. I vincoli (1.1a) impongono che ogni corso sia composto dal giusto numero di lezioni. I vincoli (1.1b) servono per assicurarsi che in ogni momento non ci siano più lezioni che aule. I vincoli (1.1c) servono per assicurarsi che lezioni con studenti in comune non abbiano luogo nello stesso momento.

De Werra (1995) ha dimostrato che il problema della COMPILAZIONE DEGLI ORARI DEI CORSI UNIVERSITARI così formulato è $\mathcal{NP}$-Completo attraverso una riduzione al problema della COLORAZIONE DEI GRAFI.

1.3.2 Formulazione del Metaheuristics Network

Si illustra adesso la riduzione del problema adottata dal *Metaheuristics Network*.[6] Introdotta da Ben Paechter,[7] riflette alcuni degli aspetti reali nella compilazione degli orari dei corsi alla Napier University di Edimburgo.

Sia T l'insieme degli intervalli temporali (si considerano 45 intervalli, ossia una settimana composta di 5 giorni, ognuno diviso in 9 periodi).

[6] Per una descrizione della rete di addestramento alla ricerca *Metaheuristics Network* si veda l'Introduzione.

[7] Coordinatore del nodo **ECRG** di Edimburgo del *Metaheuristics Network*.

Sia R l'insieme delle aule. Ogni aula soddifa un certo numero di requisiti (ad esempio, dispone di un videoproiettore, di connessioni di rete per computer, e così via) e ha una certa capienza.

Sia E l'insieme degli eventi (lezioni, esercitazioni, laboratori). Ogni evento necessita di un insieme di requisiti che devono essere soddisfatti dall'aula in cui questo avrà luogo.

Sia S l'insieme degli studenti. Ad ogni studente è associata una lista di eventi da seguire.

Partendo da questi insiemi è possibile determinare gli indirizzi $I_1, \ldots, I_r$, ossia gli insiemi di eventi con studenti in comune. Gli eventi I_l devono essere assegnati a intervalli temporali diversi.

Si posso determinare anche gli insiemi $G_1, \ldots, G_{|E|}$, ossia gli insiemi di aule che soddisfano i requisiti richiesti dagli eventi. L'evento $e_i \in E$ può essere associato solo ad un'aula $r_{i_j} \in G_i$.

Il problema si formula nel seguente modo:

si trovino gli y_{ik} $(i = 1, \ldots, |E|; k = 1, \ldots, 45)$
ed i w_{ij} $(i = 1, \ldots, |E|; j = 1, \ldots, |R|)$ tali che

$$y_{ik} = 0 \text{ oppure } 1 \qquad \text{dove } i = 1, \ldots, |E|; k = 1, \ldots, 45 \quad (1.2a)$$

$$w_{ij} = 0 \text{ oppure } 1 \qquad \text{dove } i = 1, \ldots, |E|; j = 1, \ldots, |R| \quad (1.2b)$$

$$\sum_{k=1}^{45} y_{ik} = 1 \qquad \text{dove } i = 1, \ldots, |E| \quad (1.2c)$$

$$\sum_{e_{l_i} \in I_l} y_{ik} \leq 1 \qquad \text{dove } l = 1, \ldots, r; k = 1, \ldots, 45 \quad (1.2d)$$

$$\sum_{r_{i_j} \in G_i} w_{ij} = 1 \qquad \text{dove } i = 1, \ldots, |E| \quad (1.2e)$$

$$\sum_{i=1}^{|E|} y_{ik} \cdot w_{ij} \leq 1 \qquad \text{dove } j = 1, \ldots, |R|; k = 1, \ldots, 45 \quad (1.2f)$$

dove $y_{ik} = 1$ indica che l'evento $e_i \in E$ è assegnato all'intervallo temporale k e $w_{ij} = 1$ indica che l'evento $e_i \in E$ è associato all'aula $r_j \in R$. I vincoli (1.2c) impongono che ogni evento sia assegnato ad un intervallo temporale. I vincoli (1.2d) servono per assicurarsi che eventi con studenti in comune non abbiano luogo nello stesso momento. I vincoli (1.2e) impongono che ad ogni evento sia associato un'aula che soddisfi tutti i requisiti richiesti. I vincoli (1.2f) servono per assicurarsi che un solo evento abbia luogo nella stessa stanza nello stesso momento.

Una soluzione che soddisfi le equazioni 1.2a – 1.2f è una **soluzione ammissibile.**

Per il calcolo delle violazioni dei vincoli derogabili è opportuno definire i seguenti sottoinsiemi dell'insieme T degli intervalli temporali:

- $\text{Day}_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$;
- $\text{Day}_2 = \{10, 11, 12, 13, 14, 15, 16, 17, 18\}$;
- $\text{Day}_3 = \{19, 20, 21, 22, 23, 24, 25, 26, 27\}$;
- $\text{Day}_4 = \{28, 29, 30, 31, 32, 33, 34, 35, 36\}$;

- $\text{Day}_5 = \{37, 38, 39, 40, 41, 42, 43, 44, 45\}$;
- $T_{\text{row}} = T \setminus \{8, 9, 17, 18, 26, 27, 35, 36, 44, 45\}$;
- $T_{\text{last}} = \{9, 18, 27, 36, 45\}$;

Il primo vincolo derogabile prevede che ogni studente che abbia una sola lezione nel giorno causi una violazione, e lo si formula nel seguente modo:

$$\delta(s_z, e_i) = 0 \text{ oppure } 1 \qquad \text{dove } z = 1, \ldots, |S|; i = 1, \ldots, |E| \tag{1.3a}$$

$$X_{z,d} = \sum_{k \in \text{Day}_d} y_{ik} \cdot \delta(s_z, e_i) \qquad \text{dove } i = 1, \ldots, |E| \tag{1.3b}$$

$$\Delta(X_{z,d}) = 0 \text{ oppure } 1 \tag{1.3c}$$

$$SCV_1 = \sum_{z=1}^{|S|} \Delta(X_{z,d}) \qquad \text{dove } d = 1, \ldots, 5 \tag{1.3d}$$

dove $\delta(s_z, e_i) = 1$ indica che lo studente $s_z \in S$ segue l'evento $e_i \in E$. L'equazione (1.3b) conta il numero di eventi che lo studente s_z segue nel giorno d. $\Delta(X_{z,d}) = 1$ se e solo se $X_{z,d} = 1$, ossia se e solo se lo studente s_z nel giorno Day_d segue un solo evento. Infine l'equazione (1.3d) conta il numero totale di violazioni del primo vincolo derogabile.

Il secondo vincolo derogabile prevede che ogni studente che segue più di due eventi consecutivi nello stesso giorno causi una violazione per ogni evento in eccesso ai due (avremo una violazione per tre eventi consecutivi, due violazioni per quattro eventi consecutivi, e così via). La formulazione è

la seguente:

$$\gamma(s_z, k) = 0 \text{ oppure } 1 \qquad \text{dove } z = 1, \ldots, |S|; k = 1, \ldots, 45 \tag{1.4a}$$

$$\gamma_{\text{row}}(s_z, k) = \prod_{n=k}^{k+2} \gamma(s_z, n) \qquad \text{dove } z = 1, \ldots, |S| \tag{1.4b}$$

$$Z_{z,k} = \sum_{k \in T_{\text{row}}} \gamma_{row}(s_z, k) \qquad \text{dove } z = 1, \ldots, |S| \tag{1.4c}$$

$$SCV_2 = \sum_{z=1}^{|S|} Z_{z,k} \qquad \text{dove } k \in T_{\text{row}} \tag{1.4d}$$

dove $\gamma(s_z, k) = 1$ indica che lo studente $s_z \in S$ segue un evento nell'intervallo temporale $k \in T$. $\gamma_{\text{row}}(s_z, k) = 1$ se e solo se lo studente s_z segue tre eventi consecutivamente negli intervalli temporali $k, k+1, k+2$. L'equazione (1.4c) conta quanti gruppi di tre eventi consecutivi ha lo studente s_z durante i cinque giorni. Infine l'equazione (1.4d) conta il numero totale di violazioni del secondo vincolo derogabile.

Il terzo vincolo derogabile prevede che ogni studente che segue un evento nell'ultimo intervallo temporale del giorno causi una violazione. La formulazione è la seguente:

$$Y_z = \sum_{k \in T_{last}} y_{ik} \cdot \delta(s_z, e_i) \qquad \text{dove } i = 1, \ldots, |E| \tag{1.5a}$$

$$SCV_3 = \sum_{z=1}^{|S|} Y_z \tag{1.5b}$$

L'equazione (1.5a) conta il numero di volte che lo studente $s_z \in S$ segue un evento nell'ultimo intervallo temporale durante i cinque giorni. L'equazione (1.5b) conta il numero totale di violazioni del terzo vincolo derogabile.

Tra tutte le soluzioni ammissibili (ossia le soluzioni che soddisfano le equazioni 1.2a – 1.2f) se ne cerca una che minimizzi il numero di violazioni

dei tre vincoli derogabili e che, quindi, soddisfi anche l'equazione:

$$\min SCV = SCV_1 + SCV_2 + SCV_3 \tag{1.6a}$$

1.3.3 Le istanze del problema

Nella prima fase di attività del *Metaheuristics Network*, il nodo **ECRG** di Edimburgo ha fornito un generatore di istanze del problema. Il generatore produce istanze che ammettono sempre una soluzione perfetta, ossia che soddisfa tutti i vincoli inderogabili e non produce violazioni di vincoli derogabili. Il generatore necessita di otto parametri ed un seme casuale che servono a specificare vari aspetti dell'istanza generata.

Sono state definite tre classi di istanze, rispettivamente chiamate *small*, *medium* e *large*, le cui caratteristiche sono definite dai parametri riportati in Tabella 1.1. Differenti istanze di una classe possono quindi essere generate

Tabella 1.1: Valore dei parametri delle tre classi di istanze.

Classe	*small*	*medium*	*large*
Numero eventi	100	400	400
Numero aule	5	10	10
Numero requisiti	5	5	10
Numero medio requisiti per aula	3	3	5
Percentuale requisiti usati da evento	70	80	90
Numero studenti	80	200	400
Massimo numero eventi per studente	20	20	20
Massimo numero studenti per evento	20	50	100

passando gli otto parametri al generatore e cambiando il seme casuale. Per

una descrizione dettagliata del generatore di istanze si veda Rossi-Doria e Paechter (2002).

Per farsi un'idea della complessità di questi problemi si osservi che la dimensione dello spazio di ricerca è data dal numero di possibili orari, che si calcola con la formula $(|R| \cdot 45)^{|E|}$ dove con $|R|$ si intende la cardinalità dell'insieme delle aule, 45 è la cardinalità dell'insieme degli intervalli temporali e con $|E|$ si intende la cardinalità dell'insieme degli eventi.

Capitolo 2

Metaeuristiche

Il termine *metaeuristica* deriva dalla composizione di due parole greche. *Euristica* deriva dal verbo *heurísco* ($\epsilon\upsilon\rho\acute{\iota}\sigma\kappa\omega$) che significa "cercare", mentre il prefisso *meta* ($\mu\epsilon\tau\alpha$) significa "oltre, ad un livello più alto".

Il termine metaeuristica è stato usato in letteratura con diverse accezioni. Solo negli ultimi anni qualche ricercatore ha proposto una definizione generale (Osman e Laporte, 1996; Voss *et al.*, 1999). Citiamo ad esempio quella data da Stützle (1998):

> *Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal is to avoid the disadvantages of iterative improvement and, in particular, multiple descent by allowing the local search to escape from local optima. This is achieved by either allowing worsening moves or generating new starting solutions for the local search in a more "intelligent" way than just providing random initial solutions. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can*

> *be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). Many of the metaheuristic approaches rely on probabilistic decisions made during the search. But, the main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form.* Stützle (1998)

Il *Metaheuristics Network* ha adottato la seguente definizione:

> *A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimisation problems with relatively few modifications to make them adapted to a specific problem.* Metaheuristics Network (2000)

Blum e Roli (2001) hanno estrapolato una serie di proprietà caratteristiche delle metaeuristiche:

- Lo scopo delle metaeuristiche è quello di esplorare, in maniera efficace, lo spazio di ricerca per trovare soluzioni ottimali o vicine all'ottimo.
- Le metaeuristiche sfruttano meccanismi per esplorare lo spazio di ricerca, evitando di rimanere confinati in zone limitate, come il bacino di attrazione di un ottimo locale.
- Le metaeuristiche sono indipendenti dalle caratteristiche strutturali di un problema.
- Le metaeuristiche fanno spesso uso dell'esperienza di ricerche precedenti (memoria) per indirizzare le nuove ricerche.

- Le metaeuristiche fanno uso di conoscenze su un dominio specifico, tramite strategie di più alto livello. Queste strategie devono essere scelte in modo tale da bilanciare dinamicamente lo sfruttamento dell'esperienza accumulata con le ricerche precedenti, detto **intensificazione**, e l'esplorazione dello spazio di ricerca, detta **diversificazione**.[1] Questo bilanciamento è necessario, da un lato, per identificare velocemente regioni dello spazio di ricerca in cui si trovano buone soluzioni, dall'altro, per non perdere troppo tempo nella ricerca in regioni che sono già state esplorate o che sembrano non contenere buone soluzioni.

Il resto del capitolo è organizzato come segue: il Paragrafo 2.1 esamina alcuni criteri usati per classificare le metaeuristiche; il Paragrafo 2.2 descrive le strutture fondamentali di alcune metaeuristiche che sono oggi considerate lo Stato dell'Arte. In particolare nel Paragrafo 2.2.1 si analizza la Ricerca Locale Iterata (RLI), nel Paragrafo 2.2.2 si analizza il Simulated Annealing (SA), nel Paragrafo 2.2.3 si analizza il Tabu Search (TS), nel Paragrafo 2.2.4 si analizza la Computazione Evolutiva (CE) e infine nel Paragrafo 2.2.5 si analizza Ant Colony Optimization (ACO). Il Paragrafo 2.3 descrive le linee guida fornite dal *Metaheuristics Network* per l'implementazione degli algoritmi; il Paragrafo 2.4 descrive le implementazioni, effettuate dall'autore della tesi, di quattro metaeuristiche per la risoluzione del problema della Compilazione degli Orari dei Corsi Universitari.

[1] La maggior parte della letteratura sulle metaeuristiche è in lingua inglese, dove si usa indicare i due fattori da bilanciare con i termini *intensification* e *diversification*. Da questo punto in avanti useremo in modo consistente i rispettivi termini italiani *intensificazione* e *diversificazione*.

2.1 La classificazione delle metaeuristiche

In letteratura sono stati proposti diversi criteri secondo cui classificare le metaeuristiche. Qui si illustrano quelli adottati da Stützle (1998). A nostro parere nessuno di questi criteri, considerato singolarmente, è capace di fornire una classificazione chiara e netta per la moltitudine di metodi presenti in letteratura, soprattutto vista l'esistenza di molti algoritmi ibridi.

- **Metodi basati sulla popolazione vs. metodi basati sulla ricerca di un singolo punto**. Un criterio che può essere usato per la classificazione degli algoritmi è il numero di soluzioni manipolate ad ogni iterazione. Gli algoritmi che lavorano con una singola soluzione sono chiamati *metodi a traiettoria*. Fanno parte di questi metodi il Tabu Search (TS), il Simulated Annealing (SA) e la Ricerca Locale (RL). I metodi basati sulla popolazione, come ad esempio gli algoritmi di Computazione Evolutiva (CE) e Ant Colony Optimization (ACO), ad ogni iterazione manipolano un insieme di soluzioni. La caratteristica di questi metodi è quella di sfruttare il maggior numero di soluzioni per esplorare più velocemente lo spazio di ricerca.

- **Metodi con funzione obiettivo dinamica vs. metodi con funzione obiettivo statica**. Un ulteriore criterio di classificazione è l'uso che le metaeuristiche fanno della funzione obiettivo. Mentre alcuni algoritmi mantengono la funzione obiettivo costante nel corso di tutta la procedura di ricerca, altri, come la Ricerca Locale Guidata (RLG), la modificano nel corso dell'esecuzione. L'idea che sta dietro a questo approccio è quella di avere la possibilità di esplorare nuove zone dello spazio di ricerca anche dopo aver trovato un ottimo locale. Modifi-

che nella funzione obiettivo introducono cambiamenti nella forma dello spazio di ricerca, favorendo la diversificazione.

- **Metodi con singola struttura dell'intorno vs. metodi con strutture multiple dell'intorno**. Molte metaeuristiche lavorano con una singola struttura dell'intorno di una soluzione (la definizione di struttura dell'intorno è data nel Paragrafo 2.2). Ci sono alcune metaeuristiche, come la Ricerca con Intorni Variabili,[2] che lavorano con più di una struttura di intorno, dando all'algoritmo la possibilità di adottarne di differenti in momenti diversi del processo di ricerca, favorendo in questo modo la diversificazione.

- **Metodi con memoria vs. metodi senza memoria**. Un criterio di particolare importanza per la classificazione delle metaeuristiche è l'uso che si fa della storia della ricerca. I metodi che tengono in considerazione il processo di ricerca già svolto sono usualmente detti metodi con memoria. Gli algoritmi senza memoria eseguono un processo Markoviano, in quanto fanno uso della sola soluzione corrente per cercare la successiva. Tra gli algoritmi con memoria si identificano quelli con memoria a breve termine e quelli con memoria a lungo termine. Gli algoritmi con memoria a breve termine tengono traccia solo delle ultime soluzioni visitate. Gli algoritmi con memoria a lungo termine costruiscono indici o comunque accumulano grandi quantità di dati relativamente alle soluzioni visitate in precedenza. Al giorno d'oggi, la memoria è considerata essere una componente fondamentale per la creazione di una metaeuristica efficace.

[2]Con il termine *Ricerca con Intorni Variabili* ci riferiamo a quello che in letteratura inglese è indicato come *Variable Neighborhood Search*.

- **Metodi ispirati dalla natura vs. metodi non ispirati dalla natura**. L'origine degli algoritmi è un criterio minore, ma forse uno dei più intuitivi che si possano usare per la classificazione. Molti metodi traggono ispirazione da fenomeni naturali. Metodi quali la Computazione Evolutiva (CE), Ant Colony Optimization (ACO) e Simulated Annealing (SA) hanno chiaramente tratto ispirazione dal mondo naturale, a differenza di metodi quali Tabu Search (TS) e la Ricerca Locale (RL). Il problema principale, nell'uso di questo criterio, consiste nella difficoltà di classificare molti metodi ibridi.

  Si osservi che nelle descrizioni di molti metodi ispirati ai fenomeni naturali si usa spesso la stessa terminologia adottata per quegli stessi fenomeni. Ad esempio, nel caso della Computazione Evolutiva (CE), più precisamente nel caso degli Algoritmi Genetici (AG), è ormai consuetudine riferirsi alla codifica delle componenti delle soluzioni come a *geni*, e alla codifica delle soluzioni stesse come a *cromosomi*. Anche gli operatori di variazione casuale delle soluzioni sono indicati per mezzo di termini in uso nelle scienze biologiche, quali *cross-over* e *mutazione*. Nel caso degli Ant Colony Optimization (ACO), invece, si parla di *colonie di formiche artificiali* che costruiscono le soluzioni depositando il *feromone* sulle componenti. Pur riconoscendo come questa terminologia sia poco rigorosa, non si può fare a meno di notare come si sia diffusa anche in letteratura, probabilmente perché il parallelo con il mondo naturale rende la comprensione più intuitiva in molti casi.

2.2 Lo Stato dell'Arte

In questo paragrafo si descrivono alcune delle metaeuristiche facenti parte dell'attuale Stato dell'Arte. Nel seguito faremo riferimento al problema di ottimizzazione $\Pi_{opt} = (I, S, f, min)$ e alla struttura di intorno $\mathcal{N}(s)$.

Definizione 2.1 *Data un'istanza $x \in I$, una* struttura di intorno *è una funzione $\mathcal{N}$: $\mathcal{S}(x) \to 2^{\mathcal{S}(x)}$ che assegna ad ogni soluzione $s \in \mathcal{S}(x)$ un insieme di soluzioni vicine $\mathcal{N}(s) \subseteq \mathcal{S}(x)$. $\mathcal{N}(s)$ è detto l'*intorno *di s.*

La scelta di una struttura di intorno appropriata è cruciale per le prestazioni di molti algoritmi, e spesso deve essere fatta in funzione dello specifico problema che si vuole risolvere. La struttura d'intorno definisce l'insieme delle soluzioni che possono essere raggiunte da s tipicamente in un solo passo dell'algoritmo. La struttura di intorno usualmente non è definita enumerando esplicitamente i possibili insiemi di vicini, ma piuttosto è definita implicitamente per mezzo di operazioni applicabili alla soluzione s, per trasformarla in una soluzione diversa.

Tutti gli algoritmi che andiamo a descrivere fanno riferimento a *condizioni di terminazione*, le quali identificano i casi in cui l'algoritmo deve fermarsi e ritornare la soluzione candidata ad ottimo globale per l'istanza $x \in I$. Alcune di queste condizioni possono essere:

- il tempo fissato per l'esecuzione dell'algoritmo è esaurito;
- il massimo numero di iterazioni ammesse per l'algoritmo è raggiunto;
- nelle ultime n iterazioni non ci sono stati miglioramenti nel valore della funzione obiettivo.

2.2.1 Ricerca Locale Iterata

Nella Ricerca Locale si parte da una soluzione s, spesso generata in modo casuale, e si esplora il suo intorno $\mathcal{N}(s)$. Lo pseudocodice è mostrato in Algoritmo 1. La funzione Migliora(s, $\mathcal{N}(s)$) può essere una funzione *first*

Algoritmo 1 Ricerca Locale (RL)

input: Un'istanza $x \in I$ del problema Π_{opt}

$s \leftarrow$ Genera_Soluzione_Iniziale()

repeat

$s^* \leftarrow$ Migliora(s, $\mathcal{N}(s)$)

$s \leftarrow s^*$

until nessun miglioramento è possibile

$s_{best} \leftarrow s$

output: s_{best}, "candidato" a soluzione ottima per $x \in I$

improvement, in cui si esamina l'intorno $\mathcal{N}(s)$ e si sceglie la prima soluzione che migliora la funzione obiettivo, oppure una funzione *best improvement*, in cui si controlla l'intero intorno alla ricerca della soluzione che minimizza f. In entrambi i casi ci si ferma in un minimo locale, pertanto le prestazioni dell'algoritmo dipendono fortemente dalle definizioni di S, f e $\mathcal{N}(s)$. In definitiva RL definisce una corrispondenza tra l'insieme S e l'insieme più piccolo S^* delle soluzioni s^* minimi locali.

Uno dei metodi più evoluti di Ricerca Locale (RL) è rappresentato dalla metaeuristica Ricerca Locale Iterata (RLI) (Moscato, 1989; Stützle, 1998; Lourenço *et al.*, 2002). RLI applica una ricerca locale (RL) ad una soluzione iniziale, fino a trovare un minimo locale; dopodiché, si perturba questo minimo locale creando una nuova soluzione di partenza e si ripete la ricerca locale (RL). L'importanza della perturbazione è evidente: un cambiamento

troppo piccolo nella soluzione non permette la fuga dal bacino di attrazione del minimo locale appena trovato, mentre una perturbazione troppo grande rende il tutto simile ad un algoritmo *Random Restart Local Search (RRLS)*, in cui il successivo punto di partenza è generato ogni volta in modo casuale.

RLI tenta di effettuare una ricerca efficiente seguendo una traiettoria lungo i minimi locali dell'insieme S^*, invece che su punti dell'insieme S di tutte le soluzioni, applicando il seguente schema:

1. Esegui la ricerca locale (RL) partendo da una soluzione iniziale s fino ad ottenere il minimo locale s^*.

2. Perturba s^* e ottieni s'.

3. Esegui la ricerca locale (RL) partendo da s' fino ad ottenere il minimo locale s'^*.

4. Sulla base di un *criterio di accettazione* decidere se aggiornare o meno il valore di s^* con quello di s'^*.

5. Torna al punto 2.

Perturbare la soluzione s^* serve a produrre un punto di partenza per la ricerca locale (RL) che si trovi fuori dal bacino di attrazione del minimo locale s^*, ma non eccessivamente lontano da esso. Per tale motivo, spesso la perturbazione consiste nello scegliere una nuova soluzione appartenente ad una struttura di intorno definita per mezzo di operazioni diverse da quelle che definiscono la struttura di intorno usata dalla routine di ricerca locale. Nel caso in cui la perturbazione dipenda dalla *storia* degli s^* precedenti, si parla di ricerca con memoria. Il *criterio di accettazione* filtra e fornisce un feedback all'azione di perturbazione, in base alle caratteristiche del nuovo minimo locale, e controlla il bilanciamento tra l'intensificazione e la diversificazione. Ad esempio, un

criterio di accettazione in cui la nuova soluzione è accettata solo se il valore della funzione obiettivo è minore, dà luogo ad un'estrema intensificazione, mentre un criterio che accetta sempre la nuova soluzione, dà luogo ad una estrema diversificazione. Lo pseudocodice per RLI è mostrato in Algoritmo 2.

Algoritmo 2 Ricerca Locale Iterata (RLI)

input: Un'istanza $x \in I$ del problema Π_{opt}
$s \leftarrow$ Genera_Soluzione_Iniziale()
$s^* \leftarrow \text{RL}(s)$
repeat
  $s' \leftarrow$ Perturbazione(s^*, *storia*)
  $s'^* \leftarrow \text{RL}(s')$
  $s^* \leftarrow$ Applica_Criterio_Accettazione(s^*, s'^*, *storia*)
until le condizioni di terminazione non sono soddisfatte
$s_{best} \leftarrow s^*$
output: s_{best}, "candidato" a soluzione ottima per $x \in I$

In termini di implementazione, molta della potenziale complessità di RLI risiede nella dipendenza dalla *storia*. Nel caso in cui non se ne faccia uso, la ricerca è senza memoria: la perturbazione e il criterio di accettazione non dipendono da alcuna delle soluzioni visitate in precedenza, e si decide di accettare o meno s'^* usando una regola fissa. In questo caso si ottiene un cammino casuale su S^* che si configura come un processo Markoviano, ossia la probabilità di fare un particolare passo da s_1^* a s_2^* dipende esclusivamente da s_1^* e s_2^*. Si osservi che la scelta del punto di partenza non dovrebbe influire sui risultati ottenibili dall'algoritmo in caso di lunghi tempi di esecuzione.

2.2.2 Simulated Annealing

Il Simulated Annealing (SA) trae ispirazione dalla fisica del fenomeno di formazione delle strutture cristalline (Metropolis *et al.*, 1953), e si basa sul processo di tempra (*annealing*) dei solidi.

In questo processo, un solido è portato ad alta temperatura, così che tutte le sue molecole o atomi si dispongano in modo casuale nella fase liquida; il liquido, poi, è lasciato raffreddare lentamente, così che le molecole si portino in uno stato di minima energia (e quindi termodinamicamente più stabile), ovvero lo stato fondamentale del reticolo cristallino. L'abbassamento della temperatura avviene per piccoli passi, in modo che, ad ogni valore di temperatura, il sistema possa raggiungere l'equilibrio termico. Se il raffreddamento avviene troppo rapidamente, cioè se al solido non è permesso di raggiungere l'equilibrio termico per ogni valore della temperatura, invece della struttura di reticolo cristallino, si possono creare strutture non stabili. Il modo con cui la temperatura è abbassata si riassume nel *cooling schedule*, o schema di raffreddamento, che definisce la temperatura iniziale, la temperatura di congelamento e la diminuzione di temperatura dt tra due stati successivi.

Il Simulated Annealing è stato presentato come algoritmo di ricerca per problemi di ottimizzazione combinatoria da Kirkpatrick *et al.* (1983) e Cerný (1985). Lo stato termodinamico del sistema è analogo alla soluzione corrente del problema di ottimizzazione Π_{opt}, mentre l'equazione dell'energia per il sistema termodinamico è analoga alla funzione obiettivo. L'idea fondamentale è quella di permettere mosse verso soluzioni con valore della funzione obiettivo più alto (detti anche *passi in salita*) per poter uscire dal bacino di attrazione dei minimi locali. La probabilità di fare una tale mossa dipende da un parametro che svolge una funzione analoga a quella della temperatura nel processo fisico di tempra dei solidi. Lo pseudocodice per la metaeuristica

Simulated Annealing è mostrato in Algoritmo 3.

Algoritmo 3 Simulated Annealing (SA)

input: Un'istanza $x \in I$ del problema Π_{opt}
$T \leftarrow T_{iniziale}$
$s \leftarrow$ Genera_Soluzione_Iniziale()
while le condizioni di terminazione non sono soddisfatte **do**
  scegli $s' \in \mathcal{N}(s)$
  if $f(s') \leq f(s)$ **then**
    $s \leftarrow s'$;
  else
    accetta $s \leftarrow s'$ con probabilità $p(T, s', s)$
  end if
  Aggiorna_Temperatura(T)
end while
$s_{best} \leftarrow s$
output: s_{best}, "candidato" a soluzione ottima per $x \in I$

L'algoritmo comincia generando una soluzione iniziale (in modo casuale o con l'ausilio di euristiche) e inizializzando il parametro T della *temperatura*. Dopodiché si ripete il procedimento di ricerca fino a quando non sono soddisfatte le condizioni di terminazione. Si sceglie casualmente una soluzione $s' \in \mathcal{N}(s)$ e la si accetta come nuova soluzione corrente in funzione dei valori $f(s)$, $f(s')$ e T. s è rimpiazzato da s' se $f(s') < f(s)$, oppure, nel caso in cui $f(s') \geq f(s)$, è rimpiazzato con una probabilità che è funzione di T e $f(s') - f(s)$. La probabilità è usualmente calcolata in accordo alla distribuzione di Boltzmann, ovvero $p(T, s', s) = e^{-\frac{f(s')-f(s)}{T}}$.

La temperatura T è fatta decrescere durante il procedimento, secon-

do uno schema di raffreddamento definito all'interno della funzione Aggiorna_Temperatura(), così che all'inizio della ricerca la probabilità di accettare *passi in salita* sia alta e diminuisca gradualmente.

2.2.3 Tabu Search

L'idea base del Tabu Search (TS) è stata introdotta per la prima volta da Glover (1986). Una descrizione dettagliata del metodo si può trovare in Glover e Laguna (1997). TS usa in modo esplicito la *storia* della ricerca effettuata, sia per uscire dal bacino di attrazione dei minimi locali che per implementare delle strategie esplorative. Presentiamo per prima una versione semplificata di TS che ci permetta di introdurre i concetti di base. Successivamente, si analizza l'algoritmo TS nella sua forma più completa. L'algoritmo TS Sem-

Algoritmo 4 Tabu Search Semplice (TS Semplice)

input: Un'istanza $x \in I$ del problema Π_{opt}

$s \leftarrow$ GeneraSoluzioneIniziale()

$TabuList \leftarrow \emptyset$

while le condizioni di terminazione non sono soddisfatte **do**

  scegli $s \in \mathcal{N}(s) \setminus TabuList$

  Aggiorna($TabuList$)

end while

$s_{best} \leftarrow s$

output: s_{best}, "candidato" a soluzione ottima per $x \in I$

plice, il cui pseudocodice è mostrato in Algoritmo 4, applica una routine di ricerca locale e adopera una memoria a breve termine per uscire dal bacino di attrazione dei minimi locali ed evitare di "ciclare" su soluzioni già visitate. La memoria a breve termine è realizzata per mezzo di una *tabu list* che tiene

traccia delle soluzioni visitate più di recente, e vieta mosse che portino a loro. Dall'intorno della soluzione corrente si rimuovono le soluzioni che compaiono nella *tabu list*. Nel seguito ci riferiremo a questo insieme come all'*insieme delle soluzioni ammesse*. La nuova soluzione va quindi a sostituirne una della *tabu list* (di solito si adotta il criterio FIFO all'interno della *tabu list*). In questo senso il Tabu Search è un metodo con struttura di intorno variabile, in quanto la struttura di intorno di una soluzione dipende dalla storia della ricerca.

L'uso della *tabu list* evita di far tornare il processo di ricerca su soluzioni già visitate di recente, prevenendo i cicli.[3] La lunghezza l della *tabu list* controlla la memoria del procedimento di ricerca. Valori piccoli di l concentrano la ricerca in piccole aree dello spazio di ricerca, mentre grandi valori di l favoriscono l'esplorazione di grandi regioni dello spazio di ricerca, in quanto si vietano un maggior numero di soluzioni già visitate.

L'implementazione della memoria a breve termine come lista di soluzioni già visitate non è pratica, in quanto gestire una tale lista risulta essere poco efficiente. Pertanto, invece delle soluzioni, si usa memorizzare nella *tabu list* *attributi* delle soluzioni. Gli attributi sono di solito componenti delle soluzioni, come mosse e differenze tra due soluzioni. Dato che si può considerare più di un attributo, si introduce una *tabu list* per ognuno di essi. L'insieme degli attributi e delle relative *tabu list* definisce le *condizioni tabu* che sono usate per filtrare l'intorno di una soluzione e generare quindi l'insieme delle soluzioni ammesse.

[3]Dato che la *tabu list* ha una lunghezza finita l che è minore della cardinalità dello spazio di ricerca, cicli di ordine maggiore di l sono ancora possibili.

Memorizzare mosse invece che soluzioni complete è molto più efficiente, ma introduce una perdita di informazioni, in quanto proibire una mossa può significare assegnare la condizione tabu a più di una soluzione. Pertanto, è possibile che una soluzione non visitata di buona qualità sia esclusa dall'insieme delle soluzioni ammesse. Per ovviare a questo problema si definiscono i *criteri di aspirazione*, che permettono di includere una soluzione nell'insieme delle soluzioni ammesse anche quando proibita dalle condizioni tabu. I criteri di aspirazione maggiormente diffusi sono quelli che selezionano soluzioni con valore della funzione obiettivo minore della soluzione corrente. Lo pseudocodice per la metaeuristica Tabu Search è mostrato in Algoritmo 5.

Algoritmo 5 Tabu Search (TS)

input: Un'istanza $x \in I$ del problema Π_{opt}

$s \leftarrow$ Genera_Soluzione_Iniziale()

$TabuList \leftarrow 0$

$k \leftarrow 0$

while le condizioni di terminazione non sono soddisfatte **do**

Soluzioni_Ammesse(s, k) $\leftarrow \{z \in \mathcal{N}(s) : z \notin TabuList \vee z$ soddisfa almeno una condizione di aspirazione$\}$

$s \leftarrow$ scegli $s \in$ Soluzioni_Ammesse(s, k)

Aggiorna($TabuList$)

$k \leftarrow k + 1$

end while

$s_{best} \leftarrow s$

output: s_{best}, "candidato" a soluzione ottima per $x \in I$

2.2.4 Computazione Evolutiva

I metodi di Computazione Evolutiva (CE) traggono ispirazione dalla capacità della natura di far evolvere individui che ben si adattano all'ambiente che li circonda e sanno cooperare o competere con gli altri membri della popolazione.

Gli algoritmi di Computazione Evolutiva (CE) possono quindi caratterizzarsi come modelli computazionali di un processo evolutivo che si ispira alla naturale variabilità genetica e alla selezione naturale. Ad ogni iterazione si applicano un certo numero di operatori agli individui della popolazione corrente per generare gli individui della popolazione successiva. Di solito si usano operatori capaci di combinare due o più individui della popolazione per produrne uno nuovo, chiamati *operatori di ricombinazione*, e operatori che causano un auto-adattamento dell'individuo della popolazione, chiamati *operatori di mutazione*.[4] Il motore propulsivo negli algoritmi evolutivi è la selezione degli individui della popolazione per mezzo di una funzione basata sull'idoneità, o *fitness*, di ciascuno di essi. La *fitness* può essere il valore di una funzione obiettivo o una qualunque altra misura della qualità. Gli individui della popolazione con una *fitness* maggiore hanno una probabilità più alta di essere scelti come membri per la popolazione della successiva iterazione, o come *genitori* per la generazione dei nuovi individui della popolazione. In questo senso si ha una corrispondenza col principio di evoluzione naturale *della sopravvivenza del più adatto*.

Storicamente già negli anni cinquanta e sessanta comparvero i primi studi in merito. Fogel *et al.* (1966) hanno sviluppato la **Programmazione**

[4] Con i termini *operatori di ricombinazione* e *operatori di mutazione* ci riferiamo a quelli che in letteratura inglese sono chiamati rispettivamente *crossover operators* e *mutation operators*.

Evolutiva, una tecnica che consiste nel rappresentare le soluzioni candidate come macchine a stati finiti. Le si fanno evolvere mutandone casualmente i diagrammi di transizione di stato e si selezionano le più adatte. Rechenberg (1973) ha proposto le **Strategie Evolutive**. Holland (1975) ha creato gli **Algoritmi Genetici**. Cramer (1985) ha introdotto la **Programmazione Genetica**. Nel corso degli anni sono state proposte diverse panoramiche ed analisi sulla Computazione Evolutiva (Bäck *et al.*, 1997) oltre che una tassonomia degli algoritmi evolutivi (Calegari *et al.*, 1999).

Presentiamo una versione semplificata della metaeuristica Algoritmo Genetico che ci permetta di introdurre i concetti di base. Lo pseudocodice è mostrato in Algoritmo 6. In questo algoritmo, Pop_i denota l'insieme de-

Algoritmo 6 Algoritmo Genetico Semplice (AG Semplice)

input: Un'istanza $x \in I$ del problema Π_{opt}
$i \leftarrow 0$
$Pop_0 \leftarrow$ Genera_Popolazione_Iniziale()
Valuta(Pop_0)
while le condizioni di terminazione non sono soddisfatte **do**
  $i \leftarrow i + 1$
  seleziona Pop_i da Pop_{i-1}
  Ricombina(Pop_i)
  Muta(Pop_i)
  Valuta(Pop_i)
end while
$s_{best} \leftarrow$ migliore soluzione nella popolazione Pop_i
output: s_{best}, "candidato" a soluzione ottima per $x \in I$

gli individui della popolazione alla generazione i. La nuova generazione è

ottenuta mediante l'uso degli operatori di ricombinazione e mutazione. Le caratteristiche principali degli Algoritmi Genetici sono:

Rappresentazione delle soluzioni: gli Algoritmi Genetici maneggiano popolazioni di individui. Questi individui non devono essere necessariamente soluzioni. Possono essere soluzioni parziali, insiemi di soluzioni, o un qualunque oggetto che possa essere trasformato in una o più soluzioni in maniera strutturata. In quest'ultimo caso l'oggetto è detto *genotipo*, mentre la soluzione decodificata è detta *fenotipo*. La rappresentazione più usata nei problemi di ottimizzazione combinatoria è quella della soluzione come stringa di bit, o come permutazione di n numeri interi.

Processo Evolutivo: ad ogni iterazione occorre decidere quali individui della popolazione faranno parte della generazione successiva. Nel caso in cui si scelgano gli individui della popolazione solo tra la prole, si parla di processo evolutivo *generazionale*. Nel caso in cui sia possibile trasferire individui della popolazione corrente nella generazione successiva, si parla di processo evolutivo di *equilibrio dinamico*.[5] Molti Algoritmi Genetici lavorano con popolazioni di dimensione costante durante tutto il processo di ricerca, mentre altri fanno uso di popolazioni di dimensione variabile.

Struttura dell'intorno: la funzione $\mathcal{N}_{AG} : Pop \rightarrow 2^{Pop}$ definita sull'insieme degli individui della popolazione Pop associa ad ogni individuo $p \in Pop$ un insieme di individui della popolazione $\mathcal{N}_{AG}(p) \subseteq Pop$ autorizzati ad agire come *partner* per la ricombinazione. Se un individuo

[5]Con il termine *equilibrio dinamico* ci riferiamo al corrispondente termine della letteratura inglese *steady state*.

della popolazione può essere ricombinato con un qualunque altro individuo della popolazione si parla di popolazione *non strutturata*, altrimenti si parla di *popolazione strutturata*.

Non ammissibilità: una caratteristica importante degli Algoritmi Genetici è il modo in cui si trattano le soluzioni non ammissibili (ossia quelle soluzioni che violano qualche vincolo inderogabile). Quando si ricombinano gli individui della popolazione, può capitare che il risultato produca una soluzione non ammissibile. Fondamentalmente ci sono tre modi di affrontare la situazione. Il più semplice consiste nel rigettare la soluzione non ammissibile. Poiché per molti problemi può essere alquanto difficile trovare soluzioni ammissibili, può essere maggiormente efficace adottare la strategia che penalizza le soluzioni non ammissibili in funzione della misura dell'idoneità. La terza possibilità consiste nel tentare di *riparare* una soluzione non ammissibile, facendo uso di operatori particolari capaci di rimuovere le violazioni di vincoli inderogabili.

Strategia di intensificazione: gli Algoritmi Genetici che applicano una routine di ricerca locale sugli individui della popolazione sono usualmente indicati col nome di *Algoritmi Memetici* (Moscato, 1989, 1999). Mentre la presenza della popolazione assicura l'esplorazione dello spazio di ricerca, l'uso di una tecnica di ricerca locale aiuta ad identificare velocemente le aree dove si trovano buone soluzioni.

Strategia di diversificazione: una delle maggiori difficoltà degli Algoritmi Genetici, specialmente quando si applica la routine di ricerca locale agli individui della popolazione, è la convergenza prematura verso ottimi locali. Il meccanismo di diversificazione del procedimento di ricerca maggiormente diffuso è l'uso di un operatore di mutazione. Nella forma

più semplice, l'operatore di mutazione esegue una piccola perturbazione sull'individuo della popolazione, introducendo una sorta di *rumore*.

2.2.5 Ant Colony Optimization

Ant Colony Optimization (ACO) è stato introdotto per la prima volta da Dorigo (1992) e successivamente ampliato da Dorigo *et al.* (1996, 1999); Stützle e Dorigo (2002); Dorigo e Stützle (2002). In questo paragrafo ci atterremo alla descrizione di ACO data da Dorigo e Di Caro (1999).

La fonte di ispirazione di ACO è il comportamento che hanno in natura alcune formiche quando portano al nido del cibo. Questo comportamento, che è stato descritto da Deneubourg *et al.* (1990), permette loro di trovare il percorso più breve tra il nido e le fonti di cibo. Mentre si spostano dal cibo al nido, e viceversa, le formiche depositano sul terreno una sostanza chiamata *feromone*. Quando devono scegliere in che direzione muoversi, preferiscono seguire quelle marcate da una più alta concentrazione di feromone. Questo comportamento è la base per un'interazione cooperativa che porta all'emergenza del cammino di lunghezza minima.

La metaeuristica ACO si basa su di un modello probabilistico parametrizzato, il *modello del feromone*, usato per rappresentare la traccia chimica del feromone. Le soluzioni sono definite opportunamente per mezzo di componenti. Le formiche artificiali costruiscono incrementalmente la popolazione di soluzioni aggiungendo componenti alle soluzioni parziali via via considerate.[6] Per far ciò, le formiche artificiali eseguono un cammino casuale su un grafo completamente connesso $\mathcal{G} = (\mathcal{C}, \mathcal{L})$ i cui vertici sono le componenti $\mathcal{C}$ della soluzione, e l'insieme $\mathcal{L}$ sono gli archi. Questo grafo è comunemente chiamato

[6] In questo senso la metaeuristica ACO può essere applicata ad un qualunque problema di ottimizzazione combinatoria per cui si possa definire un'euristica costruttiva.

grafo di costruzione. Alle componenti $c_i \in \mathcal{C}$ e agli archi $l_{ij} \in \mathcal{L}$ si possono associare un *valore del feromone* τ (τ_i se associato alle componenti, τ_{ij} se associato agli archi), e un *valore euristico* η (η_i se associato alle componenti, η_{ij} se associato agli archi) che rappresenta informazione a priori sull'istanza del problema. Questi valori sono usati nel processo di costruzione per effettuare le decisioni probabilistiche sul come muoversi sul grafo di costruzione. Le probabilità riguardanti il come muoversi sul grafo di costruzione sono usualmente chiamate *probabilità di transizione*.

Una versione molto semplificata dello pseudocodice di ACO è mostrato in Algoritmo 7. Il processo iterativo dell'algoritmo consiste di tre parti.

Algoritmo 7 Ant Colony Optimization (ACO)

input: Un'istanza $x \in I$ del problema Π_{opt}

while le condizioni di terminazione non sono soddisfatte **do**

Gestisci_Attività_Formiche()

Gestisci_Feromone()

Azioni_Demone()

end while

$s_{best} \leftarrow$ migliore soluzione della popolazione di soluzioni

output: s_{best}, "candidato" a soluzione ottima per $x \in I$

L'ordine con cui queste parti sono eseguite, ed il modo in cui interagiscono tra loro, sono dipendenti da una serie di scelte fatte dall'implementatore dell'algoritmo.

Gestisci_Attività_Formiche(): le formiche artificiali costruiscono le soluzioni una componente dopo l'altra, muovendosi sul grafo di costruzione $\mathcal{G}$. Le mosse sul grafo sono effettuate per mezzo di decisioni probabilistiche che fanno uso del valore del feromone e del valore euristico sulle componenti e/o sugli archi del grafo di costruzione. Ogni formica artificiale tiene traccia

della soluzione parziale costruita in termini del cammino effettuato sul grafo di costruzione.

Gestisci_Feromone(): un modo per favorire la diversificazione consiste nel diminuire il valore del feromone sulle componenti e sugli archi usati dalla formica artificiale nel processo di costruzione della soluzione. Questa strategia, chiamata *aggiornamento locale del feromone*, è usata, ad esempio, dall'algoritmo Ant Colony System, descritto nel Paragrafo 2.4.2. Una volta che la formica artificiale ha costruito l'intera soluzione, ritraccia il percorso all'inverso e aggiorna i valori del feromone sulle componenti e sugli archi usati, in accordo alla qualità della soluzione costruita. Questa strategia si chiama *aggiornamento globale del feromone*. Un altro concetto importante in ACO è l'evaporazione del feromone. Si tratta del processo per mezzo del quale l'intensità del feromone sulle componenti e sugli archi diminuisce nel tempo. Da un punto di vista pratico, l'evaporazione del feromone è necessario per evitare una convergenza prematura dell'algoritmo verso ottimi locali. Con l'evaporazione si attenua la traccia del feromone, favorendo l'esplorazione di nuove aree dello spazio di ricerca.

Azioni_Demone(): qui si usa implementare tutte quelle azioni centralizzate che non possono essere effettuate da una singola formica artificiale. Ad esempio, l'applicazione alle soluzioni costruite di una routine di ricerca locale, o la raccolta di informazioni globali, come, ad esempio, la soluzione migliore costruita fino a quel momento (per depositare una quantità extra di feromone sulle sue componenti).

Recentemente i ricercatori hanno cercato di trovare similarità tra gli algoritmi ACO, gli algoritmi CE e altri algoritmi di probabilistic-learning. Un passo importante in questa direzione è stato fatto da Blum *et al.* (2001) con lo sviluppo di Hyper-Cube Framework per Ant Colony Optimization

(HC-ACO): mediante l'uso di una particolare regola di aggiornamento del feromone, è possibile determinare connessioni esplicite con gli algoritmi CE, ad esempio con un AG Semplice che faccia uso di un operatore di ricombinazione chiamato "Gene Pool Recombination" (Mühlenbein e Voigt, 1995).

2.3 Le linee guida del Metaheuristics Network

In questo paragrafo si analizzano le linee guida dettate dal *Metaheuristics Network* (Stützle e den Besten, 2000; Knowles e Rossi-Doria, 2001; Rossi-Doria *et al.*, 2002b) per la produzione del software relativamente al problema della Compilazione degli Orari dei Corsi Universitari, di cui si è illustrata la formulazione nel Paragrafo 1.3.2.

In molti degli articoli di ricerca presenti in letteratura ci si accorge di come la comparazione tra le metaeuristiche non sia eseguita in modo equo. Di solito i ricercatori confrontano i loro risultati con quelli presenti in letteratura non tenendo nella dovuta considerazione alcuni fattori che possono far commettere errori nel paragone:

1. gli articoli da cui si prendono i risultati con cui confrontarsi possono essere stati pubblicati molto tempo prima. Vista la velocità con cui cresce la potenza computazionale, la comparazione dei tempi di calcolo è quanto mai difficile;

2. gli algoritmi messi a confronto possono essere stati implementati con differenti strutture dati, che possono incidere molto sulle prestazioni degli stessi, rendendo difficile il paragone dei risultati ottenuti;

3. gli algoritmi messi a confronto possono essere stati implementati con differenti linguaggi di programmazione, che, al pari delle strutture dati, possono incidere di molto sulle prestazioni degli stessi.

Per evitare di incappare in questi errori, all'interno del *Metaheuristics Network* sono stati definiti i seguenti criteri:

- **Hardware e Sistema Operativo**: tutti gli esperimenti sono stati eseguiti a IRIDIA su un cluster Beowulf di 9 PC: 1 master e 8 slave. Tutti i PC sono dotati di una CPU AMD Athlon Thunderbird 1100 MHz. Il nodo master ha 768 MB di RAM, mentre gli 8 slave hanno 256 MB di RAM. I nodi sono connessi tramite una rete Fast Ethernet 100 Mbit/s grazie a interfacce D-Link DFE-530TX e switch D-Link DES-1016D. Su tutti i PC gira il sistema operativo Red Hat Linux 7.0 con kernel 2.2.16-22, DQS (distributed queueing system) 3.3.2, bWatch 1.0.2. I limiti di tempo per le singole esecuzioni degli algoritmi sono stati fissati in 90, 900, e 9000 secondi sul Beowulf, rispettivamente per le istanze *small*, *medium* e *large*.

- **Linguaggio di programmazione**: come linguaggio di programmazione per l'implementazione di tutte le metaeuristiche è stato scelto il C++. Tutte le metaeuristiche sono state compilate a IRIDIA, sul Beowulf, nel seguente ambiente: **gcc** 2.95.3, **glibc** 2.2.4, **binutils** 2.10.0.18-1.

- **Starter kit**: a tutti i nodi della rete di ricerca sono stati forniti i sorgenti delle classi in C++ per la gestione del problema con le strutture dati, le routine di input e output, metodi che definiscono la struttura del vicinato (che deve essere usata dalle implementazioni del Simulated

Annealing e del Tabu Search) e la routine per la RL (che deve essere usata dalle implementazioni della Computazione Evolutiva, di Ant Colony Optimization e della Ricerca Locale Iterata).

- **Funzione di valutazione**: tutte le metaeuristiche usano la stessa funzione di valutazione per il confronto della qualità delle soluzioni. Se entrambe le soluzioni sono inammissibili (ossia non soddisfano tutti i vincoli inderogabili), la migliore è quella con il minor numero di violazioni di vincoli inderogabili; se solo una soluzione è ammissibile (ossia soddisfa tutti i vincoli inderogabili), quella è la migliore; se entrambe le soluzioni sono ammissibili, la migliore è quella con il minor numero di violazioni di vincoli derogabili.

2.3.1 Il *Metaheuristics Network* Starter Kit

Lo Starter Kit contiene il codice C++ di alcune classi da usare nell'implementazione di tutte le metaeuristiche.

- **Classe Problema:** Questa classe implementa i metodi e le strutture dati per il caricamento da file dell'istanza del problema da risolvere. Ricordiamo la notazione che abbiamo usato per descrivere il problema nel Paragrafo 1.3.2:
  - T è l'insieme degli intervalli temporali;
  - R è l'insieme delle aule;
  - E è l'insieme degli eventi;
  - S è l'insieme degli studenti;
  - F è l'insieme dei requisiti richiesti dagli eventi e posseduti dalle aule.

I file contenenti le istanze del problema hanno il seguente formato:

- **Prima riga:** numero di eventi, numero di aule, numero di requisiti, numero di studenti.
- **Una riga per ogni aula:** capacità dell'aula.
  Ci sono $|R|$ righe per questa sezione.
- **Una riga per ogni studente/evento:** uno zero od un uno.
  Zero significa che lo studente non segue l'evento, uno significa che lo segue. Il primo gruppo di $|E|$ righe si riferisce al primo studente, il successivo gruppo di $|E|$ righe si riferisce al secondo studente, e così via. In totale ci sono $|S| \cdot |E|$ righe in questa sezione.
- **Una riga per ogni aula/requisito:** uno zero od un uno.
  Zero significa che l'aula non soddisfa il requisito, uno significa che lo soddisfa. Se $|F|$ è il numero di requisiti, il primo gruppo di $|F|$ righe si riferisce alla prima aula, il secondo gruppo di $|F|$ righe si riferisce alla seconda aula, e così via. In totale ci sono $|R| \cdot |F|$ righe in questa sezione.
- **Una riga per ogni evento/requisito:** uno zero od un uno.
  Zero significa che l'evento non richiede quel requisito, uno significa che lo richiede. Se $|F|$ è il numero di requisiti, il primo gruppo di $|F|$ righe si riferisce al primo evento, il secondo gruppo di $|F|$ righe si riferisce al secondo evento, e così via. In totale ci sono $|E| \cdot |F|$ righe in questa sezione.

- **Classe Soluzione:** Questa classe implementa i metodi e le strutture dati per la gestione delle soluzioni dell'istanza del problema da risolvere. La rappresentazione scelta per la soluzione è una rappresentazione diretta.

Una soluzione consiste in una lista ordinata di lunghezza $|E|$, in cui la posizione corrisponde agli eventi (la posizione i corrisponde all'evento e_i, per $i = 1, \cdots, |E|$). Un numero intero tra 0 e 44 in posizione i indica l'intervallo temporale in cui si svolge l'evento e_i. Ad esempio, la lista 3, 27, 43, $\cdots$, 10 indica che l'evento e_1 è assegnato al quarto intervallo temporale, l'evento e_2 è assegnato al ventottesimo intervallo temporale, e così via. Gli assegnamenti delle aule non fanno parte della rappresentazione esplicita. Abbiamo quindi che per ogni intervallo temporale si ha una lista di eventi che avranno luogo in quel momento, ed una lista di possibili aule in cui quegli eventi possono avere luogo. Gli assegnamenti delle aule agli eventi sono effettuati per mezzo di un algoritmo di *matching*. Inizialmente si ricava il *matching* di cardinalità massima tra i due insiemi usando un algoritmo deterministico di flusso su reti, e, nel caso siano rimasti esclusi degli eventi, li si seleziona secondo l'ordine di apparizione, e li si mette nella prima aula che soddisfi i requisiti e che contenga il minor numero di eventi. Questo modo di generare gli assegnamenti delle aule per un dato assegnamento eventi-intervalli temporali è una corrispondenza iniettiva. Pertanto ogni assegnamento eventi-intervalli temporali corrisponde ad un unico orario, ossia un assegnamento completo di intervalli temporali e aule agli eventi.

All'interno di questa classe sono definiti anche i metodi per il controllo dell'ammissibilità di una soluzione e per il conteggio delle violazioni di vincoli inderogabili e derogabili. La rappresentazione delle soluzioni sottoforma di lista di assegnamenti eventi-intervalli temporali permette di definire la struttura di vicinato di una soluzione $\mathcal{N}(s)$ usando mosse che coinvolgono solo gli intervalli temporali e gli eventi. Si considerano

due tipi di mosse, pertanto $\mathcal{N}(s) = \mathcal{N}_1(s) \cup \mathcal{N}_2(s)$. Il primo operatore definisce $\mathcal{N}_1(s)$ per mezzo di uno spostamento di un singolo evento in un diverso intervallo temporale, mentre il secondo operatore definisce $\mathcal{N}_2(s)$ per mezzo dello scambio degli intervalli temporali di due eventi. La routine di Ricerca Locale (RL) è del tipo *first improvement* e si basa sui due operatori appena descritti. La routine scorre la lista di tutti gli eventi coinvolti in violazioni di vincoli, provando su di loro tutte le mosse possibili, fino a che non si migliora il valore della funzione obiettivo della soluzione, o non sono esaurite le mosse. Fino a che la soluzione non è ammissibile, ci si occupa solo degli eventi che producono violazioni di vincoli inderogabili. Una volta che è stata raggiunta l'ammissibilità della soluzione, ci si occupa degli eventi che producono violazioni dei vincoli derogabili, senza però trasformare una soluzione ammissibile in una non ammissibile. Per velocizzare la ricerca all'interno del vicinato, dopo ogni mossa si riapplica l'algoritmo di *matching* agli intervalli temporali coinvolti nella mossa, e si calcola il nuovo valore della soluzione per mezzo di una *delta evaluation*. La routine di Ricerca Locale (RL) si ferma dopo che sono stati eseguiti un certo numero di passi (il numero massimo di passi è uno dei parametri che le varie implementazioni devono definire), o dopo che è passato un certo tempo limite (il tempo limite è di solito impostato al tempo residuo per l'esecuzione dell'algoritmo, rendendo di fatto il numero di passi l'unico parametro per il controllo del tempo di esecuzione). Per una descrizione dettagliata della routine di Ricerca Locale (RL) si veda Knowles e Rossi-Doria (2001) e Rossi-Doria *et al.* (2002b).

È stato definito un formato standard per l'output delle soluzioni. La singola soluzione deve essere scritta in un file di output con estensione

.sln rispettando il seguente formato:

  - **Una riga per ogni evento, seguendo l'ordine del file contenente l'istanza del problema:** il numero dell'intervallo temporale, il numero dell'aula.
    In totale ci sono $|E|$ righe nel file soluzione.

  Il numero dell'intervallo temporale è un intero compreso tra 0 e 44 e rappresenta l'intervallo temporale in cui ha luogo l'evento. Il numero dell'aula identifica l'aula in cui ha luogo l'evento. Le aule sono numerato secondo l'ordine del file contenente l'istanza del problema, partendo da 0.

- **Classe Controllo:** Questa classe implementa i metodi e le strutture dati per il *parsing* della linea di comando, e per la gestione dei tempi di esecuzione (numero di iterazioni, tempo residuo di calcolo per l'iterazione), oltre ad occuparsi dell'output delle soluzioni trovate.

- **Classe Timer, Classe Random e Classe Utilità:** Queste sono classi di servizio che implementano metodi e strutture dati rispettivamente per il calcolo dei tempi, la generazione di numeri pseudo-casuali[7] e la gestione di strutture dati quali Matrici.

2.4 Le nostre implementazioni

In questo paragrafo si illustrano le quattro metaeuristiche sviluppate dall'autore della tesi. Tutti gli algoritmi sono stati implementati facendo uso dello

[7]La classe Random implementa l'algoritmo `ran0` tratto da Numerical Recipes in C (Press *et al.*, 1993).

Starter Kit descritto nel Paragrafo 2.3.1, e seguendo i criteri forniti dal *Metaheuristics Network*, fatta eccezione per il Meta-Algoritmo Genetico illustrato nel Paragrafo 2.4.1, realizzato due mesi prima che fosse disponibile lo Starter Kit.

2.4.1 Meta-Algoritmo Genetico

Per la realizzazione di questo algoritmo ci si è basati sull'articolo di Blum *et al.* (2002). Si tratta di un Algoritmo Genetico in cui si fa evolvere una politica per l'uso di un insieme predefinito di euristiche per la compilazione dell'orario. La politica risultante determina quali euristiche applicare per la compilazione dell'orario e in quale ordine.

Per ridurre la complessità del procedimento di compilazione dell'orario, si è deciso di usare un approccio sequenziale, secondo il quale prima si sceglie un evento, successivamente si accoppia l'evento ad un'aula, ed infine si assegna la coppia <evento, aula> ad un intervallo temporale. Il problema è quindi decomposto in una serie di problemi di assegnamento. Ogni assegnamento richiede di (i) scegliere un evento $e \in E$ tra quelli ancora da assegnare, (ii) assegnare all'evento e un'aula $r \in R$, (iii) assegnare alla coppia $<e, r>$ un intervallo temporale $t \in T$. Per ognuno di questi tre passi le decisioni sono prese applicando delle regole euristiche.

L'algoritmo fa uso della rappresentazione indiretta delle soluzioni. In questo caso gli individui della popolazione sono liste di regole euristiche che, una volta applicate, permettono di costruire l'orario effettivo. Ogni individuo della popolazione è dotato di tre liste, ciascuna di lunghezza $|E|$. La posizione j nella prima lista specifica quale euristica usare nel passo (i) del j-esimo problema di assegnamento. Similmente la posizione j nella seconda e terza

lista specificano, rispettivamente, quale euristica usare nel passo (ii) e nel passo (iii) del j-esimo problema di assegnamento.

La costruzione passo-passo della soluzione richiede quindi l'uso di un certo numero di regole euristiche. Le regole che abbiamo implementato tengono in considerazione la soluzione parzialmente costruita e una varietà di caratteristiche del problema.

Per il problema di assegnamento (i) si comincia creando una lista di eventi, ordinata in base al numero di studenti che li segue (a partire quindi dagli eventi seguiti dal maggior numero di studenti, fino a quelli seguiti dal minor numero di studenti). A mano a mano che gli eventi sono assegnati, li si elimina dalla lista. L'evento e è scelto applicando uno dei seguenti criteri:

- si sceglie il primo evento della lista;
- si sceglie l'evento che può essere accoppiato al minor numero possibile di aule (rifacendosi alla formulazione del problema nel Capitolo 2, si sceglie l'evento e_i il cui insieme associato G_i abbia cardinalità minima).

Per il problema di assegnamento (ii) l'aula da associare all'evento e_i è scelta fra quelle dell'insieme G_i, ossia tra quelle aule r_{i_j} che soddisfano i requisiti dell'evento e_i, applicando uno dei seguenti criteri di scelta:

- si sceglie l'aula con capienza minore;
- si sceglie l'aula meno utilizzata nella soluzione parziale che si sta costruendo.

Infine per il problema di assegnamento (iii), la scelta dell'intervallo temporale $t \in T$ da associare alla coppia <evento e, aula r>, si effettua una doppia selezione. Per prima cosa si seleziona una lista di intervalli temporali che

producono il minor numero di violazioni del vincolo inderogabile *"un solo evento deve aver luogo nella stessa stanza nello stesso momento"*. Questa lista è quindi ridotta selezionando gli intervalli temporali che producono il minor numero di violazioni del vincolo inderogabile *"nessuno studente deve seguire più di un corso nello stesso momento"*. Dopodiché si applica uno dei seguenti criteri di scelta:

- si sceglie l'intervallo temporale in cui si sta svolgendo il maggior numero di eventi in parallelo;
- si sceglie l'intervallo temporale in cui si trova la classe con il più alto numero di studenti.

Lo pseudocodice del Meta-Algoritmo Genetico è mostrato in Algoritmo 8. Il simbolo Pop rappresenta l'insieme degli individui della popolazione, il cui

Algoritmo 8 Meta-Algoritmo Genetico (Meta-AG)

input: Un'istanza $x \in I$ del problema Π_{opt}
$Pop \leftarrow$ Genera_Popolazione_Iniziale()
Valuta(Pop)
while le condizioni di terminazione non sono soddisfatte **do**
  Ricombina(Pop)
  Valuta(Pop)
end while
$s_{best} \leftarrow$ migliore soluzione della popolazione Pop
output: s_{best}, "candidato" a soluzione ottima per $x \in I$

numero è pari a $2 \cdot |E|$, che si mantiene costante da una generazione all'altra. Qui di seguito si spiegano le componenti in maggior dettaglio.

Genera_Popolazione_Iniziale(): le tre liste di euristiche caratterizzanti ogni individuo della popolazione, sono inizializzate in maniera casuale.

Valuta(*Pop*): per ogni individuo della popolazione *Pop* si costruisce la soluzione effettuando le scelte in accordo ai criteri euristici presenti nelle tre liste. Dopodiché se ne calcola l'idoneità contando il numero di violazioni dei vincoli inderogabili e derogabili.

Ricombina(*Pop*): in questa fase si crea la nuova generazione di individui, partendo dalla popolazione corrente *Pop*. Uno alla volta ogni individuo della popolazione *Pop* è selezionato come primo *genitore*, mentre il secondo *genitore* è scelto con criterio probabilistico in funzione dell'idoneità (il cosiddetto metodo della "roulette pesata"). Dalla coppia di *genitori* si producono due *figli* applicando due volte l'operatore di ricombinazione. Tale operatore decide probabilisticamente per ogni posizione j nelle tre liste di euristiche, se usare l'informazione del primo genitore o del secondo. Sui due *figli* si applica la funzione **Valuta()** per costruire gli orari corrispondenti e calcolare le rispettive idoneità. Usando la funzione di valutazione definita dal *Metaheuristics Network* si sceglie la soluzione migliore. Questa è quindi confrontata con la soluzione generata dal primo *genitore* al fine di scegliere nuovamente la migliore. Chi ha prodotto la soluzione migliore entrerà a fare parte della nuova popolazione. In tal modo la dimensione della popolazione rimane costante nel corso di tutta l'esecuzione.

2.4.2 Ant Colony System

Ant Colony System (ACS), proposto inizialmente da Dorigo e Gambardella (1997), fa parte della famiglia delle metaeuristiche Ant Colony Optimization (ACO) che abbiamo descritto nel Paragrafo 2.2.5. L'elemento che caratterizza Ant Colony System è l'adozione di una strategia di *aggiornamento locale del feromone*, che favorisce l'esplorazione dello spazio di ricerca, riducendo la

quantità di *feromone* sulle componenti selezionate e sugli archi percorsi dalle *formiche artificiali* nel processo di costruzione delle soluzioni.

L'algoritmo che presentiamo qui è, assieme all'implementazione di Socha *et al.* (2002) che segue le specifiche $\mathcal{MAX} - \mathcal{MIN}$ Ant System (Stützle e Hoos, 2000), la prima implementazione di un approccio ACO al problema della Compilazione degli Orari dei Corsi Universitari, apparsa in letteratura (Rossi-Doria *et al.*, 2002a). Abbiamo eseguito uno studio comparativo dei risultati prodotti dalle implementazioni di Ant Colony System e $\mathcal{MAX} - \mathcal{MIN}$ Ant System, i cui risultati saranno presentati alla conferenza EvoCOP'2003 (Socha *et al.*, 2003).

Dati i vincoli sulla rappresentazione delle soluzioni imposti dall'uso dello Starter Kit, descritto nel Paragrafo 2.3.1, abbiamo dovuto decidere come trasformare il problema di assegnamento (l'assegnamento degli $|E|$ eventi ai $|T|$ intervalli temporali) in un problema di cammino ottimo che potesse essere affrontato dalle *formiche artificiali*. A questo scopo abbiamo definito un appropriato *grafo di costruzione* e una rappresentazione del *modello del feromone* e delle informazioni euristiche per influenzare le scelte nel cammino sul grafo compiuto dalle *formiche artificiali*.

La forma più diretta di rappresentazione del grafo di costruzione è data dal prodotto cartesiano $E \times T$. In Figura 2.1 è rappresentato il grafo di costruzione. La scelta di far muovere le *formiche artificiali* lungo la lista di eventi, scegliendo un intervallo temporale per ognuno di essi, ci ha permesso di adoperare un criterio euristico per la definizione di un ordinamento totale degli eventi, che è illustrato più avanti; in tal modo, gli eventi più "difficili" sono considerati per primi, quando ci sono ancora molti intervalli temporali con poche aule occupate.

Ad ogni iterazione dell'algoritmo, via via che ognuna delle *formiche artifi-*

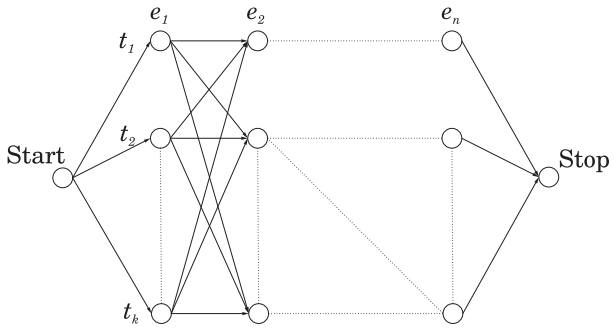

Figura 2.1: Ogni *formica artificiale* segue la lista degli eventi, e per ogni evento $e \in E$, sceglie un intervallo temporale $t \in T$. Ogni evento deve essere assegnato esattamente ad un intervallo temporale. Più eventi possono essere assegnati allo stesso intervallo temporale, in tal modo, ad ogni passo, la *formica artificiale* può scegliere ogni possibile transizione.

ciali attraversa il grafo di costruzione, si costruiscono assegnamenti completi degli eventi agli intervalli temporali. Per fare un singolo assegnamento di un evento ad un intervallo temporale, la *formica artificiale* sceglie un intervallo temporale in modo stocastico, guidata da due tipi di informazioni: (1) *informazioni euristiche*, ossia una valutazione delle violazioni dei vincoli che si commettono facendo quell'assegnamento, tenendo conto degli assegnamenti già fatti nella soluzione parziale, e (2) *informazioni stigmergiche* sottoforma di livelli di *feromone*, ossia una stima dell'utilità del fare quell'assegnamento in funzione dei risultati ottenuti nelle soluzioni delle iterazioni precedenti. L'informazione stigmergica è rappresentata per mezzo di una matrice dei valori del *feromone* $\tau : E \times T \rightarrow \mathbf{R}_{\geq 0}$, dove E è l'insieme degli eventi e T è l'insieme degli intervalli temporali. I valori degli elementi della matrice $\tau(e, t)$ sono inizializzati al valore τ_0, e poi modificati ad ogni iterazione dalle regole di *aggiornamento locale e globale del feromone*; in genere, una coppia evento-intervallo che è stata parte di buone soluzioni nelle iterazioni passate, avrà un alto valore del *feromone*, e di conseguenza avrà una probabilità più

alta di essere scelta anche nelle iterazioni future. Una volta attraversato il grafo di costruzione ed effettuati tutti gli assegnamenti evento-intervallo, si applica l'algoritmo di *matching* delle aule descritto nel Paragrafo 2.3.1, e si ottiene una soluzione candidata (un orario). La soluzione candidata è ulteriormente migliorata applicando la routine di Ricerca Locale (RL). Dopo che tutte le *formiche artificiali* hanno generato la loro soluzione candidata, si esegue *l'aggiornamento globale del feromone* usando la miglior soluzione trovata a partire dalla prima iterazione. L'intero processo di costruzione è quindi ripetuto, fino al raggiungimento del limite di tempo.

Lo pseudocodice di Ant Colony System è mostrato in Algoritmo 9, di cui spieghiamo qui di seguito le componenti. I seguenti dati sono precalcolati per tutti gli eventi $e, e' \in E$:

$$\begin{aligned} c(e,e') &:= \#\text{ studenti che seguono entrambi gli eventi } e \text{ ed } e', \\ d(e) &:= \left|\left\{e' \in E \setminus \{e\} \mid c(e,e') \neq 0\right\}\right|, \\ f(e) &:= \#\text{ requisiti richiesti dall'evento } e, \\ a(e) &:= \#\text{ studenti che seguono l'evento } e. \end{aligned}$$

Definiamo un ordinamento totale $\prec$ sugli eventi nel modo seguente:

$$\begin{aligned} e \prec e' \quad \Leftrightarrow \quad & d(e) > d(e') \vee \\ & d(e) = d(e') \wedge f(e) > f(e') \vee \\ & d(e) = d(e') \wedge f(e) = f(e') \wedge a(e) > a(e') \vee \\ & d(e) = d(e') \wedge f(e) = f(e') \wedge a(e) = a(e') \wedge l(e) < l(e') \ , \end{aligned}$$

dove $l : E \to \mathbf{N}$ è una funzione iniettiva usata solo per gestire le uguaglianze. L'insieme degli eventi totalmente ordinato si indica come $e_1 \prec e_2 \prec \ldots \prec e_n$, e si possono quindi definire i sottoinsiemi ordinati $E_i := \{e_1, \ldots, e_i\}$.

Algoritmo 9 Ant Colony System (ACS)

input: Un'istanza $x \in I$ del problema Π_{opt}

$\tau(e,t) \leftarrow \tau_0 \; \forall \; (e,t) \in E \times T$

calcola $c(e,e') \; \forall \; (e,e') \in E^2$

calcola $d(e), f(e), a(e) \; \forall \; e \in E$

ordina E secondo $\prec$, ottenendo $e_1 \prec e_2 \prec \cdots \prec e_n$

$j \leftarrow 0$

while limite di tempo non raggiunto **do**

  $j \leftarrow j + 1$

  for $a = 1$ **to** m **do**

    {assegnamenti della formica artificiale a}

    $A_0 \leftarrow \emptyset$

    for $i = 1$ **to** n **do**

      assegna all'evento e_i l'intervallo temporale t in accordo alla distribuzione di probabilità P

      aggiornamento locale del feromone per $\tau(e_i, t)$

      $A_i \leftarrow A_{i-1} \cup (e_i, t)$

    end for

    $s \leftarrow$ soluzione dopo aver applicato ad A_n l'algoritmo di *matching*

    $s \leftarrow$ soluzione dopo aver applicato a s la RL per $h(j)$ passi

    $s_{best} \leftarrow$ migliore tra s e s_{best}

  end for

  aggiornamento globale del feromone per $\tau(e,t) \; \forall \; (e,t) \in E \times T$

end while

output: s_{best}, "candidato" a soluzione ottima per $x \in I$

Per la costruzione degli assegnamenti evento-intervallo ogni *formica artificiale* assegna sequenzialmente gli intervalli agli eventi che sono considerati secondo l'ordine $\prec$. Questo significa che la *formica artificiale* costruisce assegnamenti $A_i : E_i \to T$ per $i = 0, \ldots, n$.

Si comincia dall'assegnamento vuoto $A_0 = \emptyset$. Dopo che A_{i-1} è stato costruito, l'assegnamento A_i è ottenuto come $A_i = A_{i-1} \cup \{(e_i, t)\}$ dove t è scelto in T in accordo alle seguenti probabilità:

$$P(t = t' \mid A_{i-1}, \tau) = \frac{\tau(e_i, t')^\alpha \cdot \eta(e_i, t')^\beta \cdot \pi(e_i, t')^\gamma}{\sum_{u \in T} \tau(e_i, u)^\alpha \cdot \eta(e_i, u)^\beta \cdot \pi(e_i, u)^\gamma} \ .$$

Il parametro α controlla il peso dell'informazione *stigmergica*, mentre i parametri β e γ controllano il peso delle informazioni euristiche relative ai vincoli inderogabili e derogabili, rispettivamente. La funzione euristica η, definita come segue:

$$\eta(e_i, t') := \frac{1}{1 + \sum_{e \in A_{i-1}^{-1}(t')} c(e_i, e)},$$

è usata per dare un peso maggiore a quegli intervalli temporali che producono un numero basso di violazioni del tipo "lo studente segue due eventi nello stesso momento". Per dare maggior peso a quegli intervalli che producono meno violazioni di vincoli derogabili si usa la funzione euristica π, definita come segue:

$$\pi(e_i, t') := \frac{1}{1 + L + S + R_{before} + R_{around} + R_{after}} \ , \text{dove}$$

$$
\begin{aligned}
L &:= \begin{cases} a(e_i) & \text{se } t' \text{ è l'ultimo intervallo della giornata} \\ 0 & \text{altrimenti,} \end{cases} \\
S &:= \#\text{ studenti che seguono l'evento } e_i\text{, ma nessun altro evento} \\
&\quad \text{nello stesso giorno di } t' \text{ in } A_{i-1}, \\
R_{before} &:= \#\text{ studenti che seguono l'evento } e_i \text{ ed anche eventi} \\
&\quad \text{nei due intervalli precedenti a } t' \text{ nello stesso giorno,} \\
R_{around} &:= \#\text{ studenti che seguono l'evento } e_i \text{ ed anche eventi} \\
&\quad \text{nell'intervallo precedente e successivo a } t' \text{ nello stesso giorno,} \\
R_{after} &:= \#\text{ studenti che seguono l'evento } e_i \text{ ed anche eventi} \\
&\quad \text{nei due intervalli successivi a } t' \text{ nello stesso giorno.}
\end{aligned}
$$

Dopo ogni assegnamento evento-intervallo $(e_i,\hat{t})$ si applica la regola di *aggiornamento locale del feromone* al corrispondente elemento nella matrice del feromone:

$$\tau(e_i, \hat{t}) \leftarrow (1 - \psi) \cdot \tau(e_i, \hat{t}) + \psi \cdot \tau_0 \ .$$

Il parametro $\psi \in [0, 1]$, detto anche parametro di decadimento del feromone, controlla la diversificazione nel processo di costruzione. Maggiore è il suo valore, minore è la probabilità di scegliere la stessa coppia <evento, intervallo> in futuro.

Dopo che l'assegnamento A_n è stato completato si esegue l'algoritmo di *matching* per l'assegnamento delle aule, al fine di generare un candidato a soluzione s. Ad s è applicata la routine di Ricerca Locale (RL) per un numero di passi $h(j)$ che dipende dal numero dell'iterazione corrente $j \in \mathbf{N}$.

Alla fine di ogni iterazione si applica la *regola di aggiornamento globale del feromone* nel modo seguente. Siano A_{best} gli assegnamenti del miglior candidato a soluzione s_{best} trovato fin dalla prima iterazione. Per ogni coppia

<evento, intervallo> (e, t) si calcola:

$$\tau(e,t) \leftarrow \begin{cases} (1-\rho)\cdot\tau(e,t) + \rho \cdot \frac{Q}{1+q(s_{best})} & \text{se } A_{best}(e) = t, \\ (1-\rho)\cdot\tau(e,t) & \text{altrimenti,} \end{cases}$$

dove Q e $\rho \in [0,1]$ sono parametri che controllano la quantità di feromone aggiunta dalla regola di aggiornamento, e la funzione $q(s)$ misura la qualità di una soluzione candidata s come somma pesata del numero di violazioni di vincoli inderogabili *hcv* e di violazioni di vincoli derogabili *scv*:

$$q(s) := \#\ hcv(s) * 1000 + \#\ scv(s)\ .$$

I valori dei parametri dell'algoritmo ACS sono stati scelti dopo una serie di esperimenti manuali su un insieme ristretto di istanze del problema e sono riportati nella Tabella 2.1. In un secondo momento, si è applicato il metodo automatico F-Race, descritto nel Paragrafo 3.2.2, per la determinazione dei migliori valori dei parametri da usare con le istanze appartenenti alla classe *small*, che sono riportati in Tabella 3.3.

2.4.3 Ricerca Locale Iterata

Come già detto nel Paragrafo 2.2.1, la Ricerca Locale Iterata apporta un miglioramento alla procedura di Ricerca Locale per mezzo di un'idea semplice ma efficace: ad ogni iterazione, si fornisce una nuova soluzione di partenza ottenuta mediante una "perturbazione" della soluzione corrente. Lo pseudo-codice è lo stesso di quello mostrato in Algoritmo 2, con la differenza che noi non abbiamo usato la *storia*.

Nel nostro caso, l'algoritmo genera una soluzione iniziale in modo casuale, senza usare alcuna informazione specifica del problema, seguendo il seguente schema: ad ogni evento $e_i \in E$ si associa un intervallo temporale $t_j \in T$ casualmente in accordo ad una distribuzione di probabilità uniforme. Una

Tabella 2.1: Valore dei parametri per l'Algoritmo ACS: m indica il numero di individui nella popolazione; τ_0 è il valore a cui si inizializzano gli elementi della matrice del feromone; α controlla il peso dell'informazione stigmergica; β e γ controllano il peso delle informazioni euristiche relative ai vincoli inderogabili e derogabili, rispettivamente; $\psi \in [0,1]$ controlla la diversificazione nel processo di costruzione; $h(j)$ indica il numero di passi per cui si applicata la routine di ricerca locale, in funzione del passo j; $\rho \in [0,1]$ e Q controllano la quantità di feromone aggiunta dalla regola di aggiornamento.

Classe	*small*	*medium*	*large*
m	15	15	10
τ_0	0.5	10.0	10.0
α	1.0	1.0	1.0
β	3.0	3.0	3.0
γ	2.0	2.0	2.0
ρ	0.1	0.1	0.1
ψ	0.1	0.1	0.1
$h(j)$	$\begin{cases} 5\,000 & j = 1 \\ 2\,000 & j \geq 2 \end{cases}$	$\begin{cases} 50\,000 & j \leq 10 \\ 10\,000 & j \geq 11 \end{cases}$	$\begin{cases} 150\,000 & j \leq 20 \\ 100\,000 & j \geq 21 \end{cases}$
Q	10^5	10^{10}	10^{10}

volta che tutti gli eventi sono stati associati ad un intervallo temporale, l'algoritmo di *matching*, descritto nel Paragrafo 2.3.1, assegna le aule a tutti gli eventi, ottenendo quindi la soluzione $s_0 \in S$. Questa soluzione è quindi adottata come punto di partenza per la routine di Ricerca Locale (RL). La routine di Ricerca Locale è quella fornita nello Starter Kit, come illustrato nella sezione 2.3.1. I parametri che controllano il tempo di esecuzione della Ricerca Locale (RL) sono impostati nel modo seguente: il limite temporale è posto uguale al tempo di esecuzione residuo dell'algoritmo, mentre il limite per il numero di passi è posto uguale al valore della costante INT_MAX.[8]
La funzione di idoneità $f(s)$, o *fitness*, di una soluzione s è data dalla somma pesata del numero di violazioni di vincoli inderogabili (hcv) e del numero di violazioni di vincoli derogabili (scv)

$$f(s) := \#\ hcv(s) * 1000000 + \#\ scv(s).$$

La funzione di valutazione per il confronto delle qualità delle soluzioni è quella descritta nel Paragrafo 2.3.

Perturbazione(s^*): la perturbazione della soluzione consiste nell'applicazione di una mossa scelta casualmente tra le tre seguenti:

- una mossa di tipo 1 sposta un evento in un diverso intervallo temporale;
- una mossa di tipo 2 scambia tra loro gli intervalli temporali di due eventi;
- una mossa di tipo 3 permuta tre eventi in tre intervalli temporali distinti in uno dei due modi possibili.

Tutte le scelte casuali sono fatte in accordo a distribuzioni di probabilità uniformi. Si tenga inoltre presente che la perturbazione fa uso di una struttura

[8]Il valore di INT_MAX è il massimo intero positivo rappresentabile dal calcolatore.

di intorno che è definita da una mossa in più rispetto a quelle che definiscono la struttura di intorno usata dalla routine di Ricerca Locale (RL) (che è ristretta soltanto a mosse di tipo 1 e 2).

Applica_Criterio_Accettazione(s^*,s'^*): la nuova soluzione s'^* è accettata solo se considerata "migliore" rispetto alla soluzione corrente s^*. A questo scopo si adopera la funzione di valutazione definita dal *Metaheuristics Network*, e descritta nella sezione 2.3.

2.4.4 Algoritmo Memetico

A differenza del Meta-Algoritmo Genetico descritto nel Paragrafo 2.4.1, in cui le soluzioni sono rappresentate indirettamente dagli individui della popolazione, questo algoritmo fa uso dello Starter Kit descritto nel Paragrafo 2.3.1 che prevede la rappresentazione diretta delle soluzioni. Si tratta di un Algoritmo Memetico (Moscato, 1989, 1999), con popolazione a dimensione costante, non strutturata, che implementa un processo evolutivo di equilibrio dinamico. Lo pseudocodice dell'Algoritmo Genetico è illustrato in Algoritmo 10.

La cardinalità dell'insieme degli individui della popolazione *Pop*, definita dal parametro m, è mantenuta costante da una generazione all'altra. La popolazione iniziale è costruita assegnando casualmente, per ogni individuo, un intervallo temporale ad ogni evento, in accordo ad una distribuzione di probabilità uniforme, e applicando l'algoritmo di *matching* per le assegnazione delle aule, descritto nel Paragrafo 2.3.1. Ogni individuo della popolazione iniziale è quindi migliorato, grazie alla routine di Ricerca Locale (RL) che si esegue per un numero di passi massimo pari al parametro h.

La funzione di idoneità $f(s)$, o *fitness*, di una soluzione s è data dalla somma pesata del numero di violazioni di vincoli inderogabili (hcv) e del numero di

violazioni di vincoli derogabili (*scv*)

$$f(s) := \#\ hcv(s) * 1000000 + \#\ scv(s).$$

La funzione di valutazione per il confronto delle qualità delle soluzioni è quella descritta nel Paragrafo 2.3. Ad ogni iterazione si opera sulla popolazione di soluzioni mediante tre passi principali: selezione, riproduzione e sostituzione. Nella fase di selezione si scelgono due individui della popolazione corrente, detti *genitori*, ognuno dei quali è selezionato per mezzo di un *torneo* di dimensione n: in pratica, il genitore è scelto in quanto miglior soluzione tra n selezionate casualmente, in accordo ad una distribuzione di probabilità uniforme. La riproduzione, in cui una sola soluzione *figlio* è generata dalla coppia di soluzioni *genitori*, avviene per mezzo del solo operatore di ricombinazione. Questo operatore assegna ad ogni evento $e_i \in E$, in accordo alla probabilità definita dal parametro $prob_{sel}$, il corrispondente intervallo temporale del primo o del secondo *genitore*. Una volta che a tutti gli eventi sono stati assegnati gli intervalli temporali, si applica l'algoritmo di *matching* per l'assegnazione delle aule, descritto nel Paragrafo 2.3.1, per ottenere una soluzione. Questa si migliora applicando la routine di Ricerca Locale (RL) per un numero di passi pari al parametro h. La soluzione *figlio* così generata è dapprima confrontata con la prima soluzione *genitore*, che viene rimpiazzata nel caso in cui la *fitness* della soluzione *figlio* sia minore; in caso contrario, la soluzione *figlio* è confrontata con la seconda soluzione *genitore*, che viene rimpiazzata in caso la *fitness* della soluzione *figlio* sia minore. Nel caso in cui una soluzione *genitore* sia stata sostituita dalla soluzione *figlio*, si controlla se la *fitness* della soluzione *figlio* sia minore della *fitness* dell'attuale soluzione candidata ad ottimo globale per l'istanza $x \in I$, e in caso affermativo, la soluzione *figlio* diventa la nuova soluzione candidata a ottimo globale per l'istanza $x \in I$. Nel caso in cui la soluzione *figlio* sia peggiore di entrambe

le soluzioni *genitore*, è semplicemente scartata e si continua con una nuova iterazione.

I valori dei parametri dell'Algoritmo Memetico sono stati scelti dopo una serie di esperimenti manuali su un insieme ridotto di istanze del problema, e sono riportati nella Tabella 2.2. Successivamente, si è applicato il metodo automatico F-Race, descritto nel Paragrafo 3.2.2, per la determinazione dei valori migliori dei parametri per le istanze appartenenti alla classe *small*, che sono riportati in Tabella 3.6.

Tabella 2.2: Valore dei parametri per l'Algoritmo Memetico (AM): m indica il numero di individui nella popolazione; n indica la dimensione del torneo con cui sono selezionati i genitori per la riproduzione; $prob_{sel}$ è la probabilità con cui si sceglie l'intervallo temporale da associare all'evento dal primo genitore; h indica il numero di passi per cui si applica la routine di ricerca locale.

Classe	*small*	*medium*	*large*
m	5	10	20
n	3	5	7
$prob_{sel}$	0.5	0.5	0.5
h	1e7	1e7	1e7

Algoritmo 10 Algoritmo Memetico (AM)

input: Un'istanza $x \in I$ del problema Π_{opt}

for $i = 1$ **to** m **do**

$s_i \leftarrow$ crea una soluzione casuale e applica la RL per h passi

end for

$s_{best} \leftarrow$ migliore soluzione della popolazione Pop

while le condizioni di terminazione non sono soddisfatte **do**

$s_{p_1} \leftarrow$ seleziona la prima soluzione *genitore* mediante un torneo di dimensione n

$s_{p_2} \leftarrow$ seleziona la seconda soluzione *genitore* mediante un torneo di dimensione n

for $j = 1$ **to** $|E|$ **do**

$c \leftarrow$ crea gli assegnamenti (e_j, t) per la soluzione *figlio* copiando l'intervallo temporale associato a e_j presente in s_{p_1} oppure in s_{p_2} in funzione di una probabilità $prob_{sel}$

end for

$s_c \leftarrow$ soluzione *figlio* dopo aver applicato l'algoritmo di *matching* a c

$s_c \leftarrow$ soluzione *figlio* dopo aver applicato a s_c la routine di RL per h passi

if la soluzione s_c è migliore della soluzione s_{p_1} **then**

$s_{p_1} \leftarrow s_c$; $s_{best} \leftarrow$ miglior soluzione tra s_{p_1} e s_{best}

else if la soluzione s_c è migliore della soluzione s_{p_2} **then**

$s_{p_2} \leftarrow s_c$; $s_{best} \leftarrow$ miglior soluzione tra s_{p_2} e s_{best}

else

scarta la soluzione *figlio* s_c

end if

end while

output: s_{best}, "candidato" a soluzione ottima per $x \in I$

Capitolo 3

Configurazione delle Metaeuristiche

Riprendiamo la definizione di metaeuristica data a pagina 27:

> *A metaheuristic is a set of concepts that can be used to define heuristic methods that can be applied to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimisation problems with relatively few modifications to make them adapted to a specific problem.* Metaheuristics Network (2000)

In quanto modello generale di algoritmo, una volta che è stato specificato il problema da risolvere, occorre istanziare le componenti necessarie alla metaeuristica per ottenere un algoritmo funzionante per quel problema specifico. Questo include anche l'assegnamento di un valore a tutti i parametri dell'algoritmo: la metaeuristica deve essere *configurata*. In questa tesi, seguendo la terminologia adottata da Birattari *et al.* (2002), ci riferiremo alla scelta della migliore configurazione della metaeuristica per un dato problema come al *problema di configurazione*.

Una modalità per configurare le metaeuristiche, adottata di frequente nella pratica, consiste nella scelta di un certo numero di configurazioni ritenute "promettenti" e nel confronto dei risultati ottenuti da queste basandosi su di un certo numero di esecuzioni dell'algoritmo su un certo numero di istanze del problema. Come è facilmente intuibile, un tale modo di procedere si basa molto sull'esperienza personale di chi sceglie le configurazioni, e, spesso, si traduce in una serie di esperimenti lunghi e tediosi che raramente si basano su di una qualche procedura statistica ben definita.

Birattari *et al.* (2002) hanno definito una procedura automatica per la selezione di una buona configurazione tra un'insieme predeterminato e finito di alternative, facendo uso di procedure statistiche che valutano e guidano gli esperimenti.

Abbiamo collaborato con gli autori di tale procedura (in modo particolare con Mauro Birattari) per applicarla ad alcuni algoritmi tra quelli implementati dall'autore della tesi per la soluzione del problema della Compilazione degli Orari dei Corsi Universitari.

Il resto del capitolo è organizzato come segue: il Paragrafo 3.1 introduce e dà una definizione formale del *problema di configurazione di una metaeuristica*; il Paragrafo 3.2 descrive l'idea generale che sta dietro ad un metodo di *racing*[1] e introduce F-Race, un metodo ideato specificatamente per tenere conto delle caratteristiche peculiari del *problema di configurazione delle metaeuristiche*; il Paragrafo 3.3 descrive gli esperimenti effettuati per la configurazione di due algoritmi tra quelli implementati dall'autore della tesi per risolvere il problema della Compilazione degli Orari dei Corsi

[1] Il corrispondente termine inglese, *racing method*, usato da Birattari *et al.* (2002), è stato ripreso dalla terminologia introdotta da Maron e Moore (1994), che si sono occupati della selezione dei modelli nel campo dell'apprendimento automatico.

Universitari.

3.1 Configurare una metaeuristica

In questo Paragrafo introduciamo e diamo una definizione formale del *problema di configurazione di una metaeuristica*. Prima di illustrare la definizione formale, chiariremo, con l'aiuto di un esempio, la tipologia di problemi a cui può applicarsi questa procedura di configurazione. Informalmente, questa metodologia si presta ad essere usata per configurare metaeuristiche ideate per la risoluzione di problemi ripetitivi, ossia, problemi per i quali si presentano istanze similari.

3.1.1 Un esempio: La consegna della posta

L'esempio che proponiamo è volutamente formulato in maniera semplicistica, ma permette di cogliere gli elementi essenziali del *problema di configurazione di una metaeuristica*.

Consideriamo il seguente **problema di consegna della posta**. Ogni mattina alle ore 07:00 l'ufficio postale riceve la posta che deve essere consegnata per quel giorno. Entro un tempo ragionevole, l'ufficio postale deve pianificare il percorso di consegna che ogni postino deve seguire all'interno delle rispettive zone prefissate, in modo da minimizzare la lunghezza del tragitto percorso da ogni postino. Per questo esempio assumiamo che i postini ricevano la posta da consegnare e il tragitto da seguire entro le ore 09:00.

In pratica, l'ufficio postale ogni giorno deve risolvere una nuova istanza del problema di consegna della posta per ogni postino, e deve trovare le migliori soluzioni possibili in una quantità limitata di tempo (nel nostro caso due ore). È molto probabile che ogni istanza sia diversa da quelle precedenti,

pur mantenendo una struttura similare. All'interno della stessa zona, infatti, ci saranno persone che ricevono posta tutti i giorni, e altre che ne riceveranno solo di quando in quando. Pertanto, ci aspettiamo una certa variabilità nella dimensione dell'istanza, ossia nel numero di persone a cui consegnare la posta.

La presenza di istanze diverse può essere rappresentata, in modo efficace, come il risultato di esperimenti casuali governati da una qualche misura di probabilità sconosciuta, diciamo P_I, definita sulla classe delle possibili istanze. Nell'esempio che presentiamo qui, è ragionevole assumere che i diversi esperimenti siano indipendenti e tutti governati dalla stessa misura di probabilità. In Birattari *et al.* (2002) si esamina come sia possibile affrontare il problema nel caso in cui queste assunzioni non siano ragionevoli.

L'ufficio postale decide, quindi, di commissionarci lo sviluppo di un programma per la generazione dei tragitti giornalieri dei postini, e ci concede un mese di tempo. Decidiamo, quindi, di implementare una metaeuristica per la risoluzione del problema della consegna della posta. Essendo, però, la metaeuristica un modello generale di algoritmo, la dobbiamo configurare per il problema specifico. In questo esempio, il problema che dobbiamo risolvere è quello di trovare una configurazione che produca le migliori soluzioni alle istanze del problema della consegna che si presentano *tipicamente* all'ufficio postale. Il concetto di *istanza tipica*, che qui usiamo in modo informale, deve essere posto in relazione alla misura di probabilità P_I, e sarà spiegato in termini matematici nel seguito.

Poiché P_I è sconosciuta, la sola informazione che si può usare per trovare la migliore configurazione deve essere estratta da un campione di istanze viste in precedenza dall'ufficio. Basandoci su questo insieme di istanze, cerchiamo la configurazione che si presume possa produrre le migliori soluzioni sull'intera classe delle possibili istanze che si possono presentare all'ufficio

postale.

Ipotizziamo di avere 15 giorni di tempo per effettuare la ricerca della configurazione migliore (sui 30 concessi in totale), e di disporre di una potenza di calcolo paragonabile a quella di cui è dotato l'ufficio postale.[2] Il numero di configurazioni che possiamo prendere in considerazione dipende, quindi, dal tempo residuo a nostra disposizione per effettuare la configurazione (15 giorni). Usando un approccio a *forza bruta*, come quello descritto nel Paragrafo 3.2, una volta scelto un insieme di valori dei parametri che vogliamo controllare, si può ricavare facilmente il numero di istanze su cui saranno eseguite le configurazioni. Usando un metodo di racing, come F-Race, il numero di istanze che si prende in considerazione sarà, molto probabilmente, maggiore, in quanto le configurazioni "peggiori" smettono di essere eseguite non appena si ricava sufficiente evidenza statistica contro di loro.

Nel contesto della configurazione delle metaeuristiche, il fatto di estendere i risultati ottenuti su un insieme generalmente piccolo di istanze ad un insieme più grande di istanze è giustificato dall'assunzione che la stessa misura di probabilità P_I governa la selezione di tutte le istanze: sia quelle usate per la configurazione, che quelle che saranno risolte in seguito. In questo senso, le istanze usate per la configurazione sono rappresentative dell'intera classe di istanze.

[2] È necessario allocare un tempo di esecuzione per ogni esperimento tale che, fissata un'istanza di riferimento, il numero di iterazioni eseguito dalla metaeuristica sul nostro PC, quando applicata a quella istanza di riferimento, è lo stesso rispetto al numero di iterazioni che la metaeuristica esegue sul PC dell'ufficio postale in due ore di tempo (il tempo che passa dal ricevimento della posta alla consegna dei tragitti ai postini).

3.1.2 Definizione del problema di configurazione

Per poter dare una definizione formale del *problema di configurazione di una metaeuristica* occorre considerare i seguenti oggetti:

- Θ è l'insieme finito delle configurazioni candidate;
- I è l'insieme delle istanze (eventualmente infinito);
- P_I è una misura di probabilità sull'insieme I delle istanze: con un abuso di notazione, indichiamo con $P_I(i)$ la probabilità che l'istanza $i \in I$ sia selezionata per essere risolta;[3]
- $t : I \to \mathbf{R}$ è una funzione che associa ad ogni istanza il tempo di esecuzione allocato per la sua risoluzione;[4]
- $\boldsymbol{c}(\theta, i) = \boldsymbol{c}(\theta, i, t(i))$ è una variabile casuale che rappresenta il costo della migliore soluzione trovata eseguendo per $t(i)$ secondi la configurazione $\theta \in \Theta$ sull'istanza $i \in I$;[5]
- $C \subset \mathbf{R}$ è l'intervallo dei valori che possono essere assunti da $\boldsymbol{c}$, ossia, i possibili valori di costo della miglior soluzione trovata in un'esecuzione della configurazione $\theta \in \Theta$ su un'istanza $i \in I$;

[3] Poiché una misura di probabilità è associata ad un (sotto)insieme e non ad un singolo elemento, la notazione corretta dovrebbe essere $P_I(\{i\})$. Il nostro abuso di notazione consiste, pertanto, nell'usare lo stesso simbolo i sia per l'elemento $i \in I$, che per il singleton $\{i\} \subset I$.

[4] Negli esperimenti effettuati per questa tesi, il tempo di esecuzione concesso ad ogni configurazione candidata per risolvere una istanza della classe *small*, definita nel Paragrafo 1.3.3, ammonta a 90 secondi.

[5] Nel seguito, al fine di alleggerire la notazione, considereremo implicita la dipendenza di $\mathbf{c}$ da t.

- P_C è una misura di probabilità sull'insieme C: con la notazione[6] $P_C(c|\theta, i)$ indichiamo la probabilità che c sia il costo della miglior soluzione trovata eseguendo la configurazione θ sull'istanza i per $t(i)$ secondi;

- $\mathcal{C}(\theta) = \mathcal{C}(\theta|\Theta, I, P_I, P_C, t)$ è il criterio che deve essere ottimizzato rispetto a θ. Nel caso più generale, misura in qualche senso la desiderabilità di θ.

Sulla base di questi concetti, il *problema di configurazione di una metaeuristica* può essere descritto formalmente dalla sestupla $(\Theta, I, P_I, P_C, t, \mathcal{C})$. La soluzione di questo problema è la configurazione θ^* tale che:

$$\theta^* = \arg\min_{\theta} \mathcal{C}(\theta). \tag{3.1}$$

Per quanto riguarda il criterio $\mathcal{C}$, sono possibili diverse alternative. In questa tesi abbiamo adottato l'ottimizzazione del valore atteso del costo $\boldsymbol{c}(\theta, i)$. Formalmente:

$$\mathcal{C}(\theta) = E_{I,C}[\boldsymbol{c}(\theta, i)] = \int_I \int_C c(\theta, i) \mathrm{d}P_C(c|\theta, i) \mathrm{d}P_I(i) \tag{3.2}$$

dove il valore atteso è considerato rispetto a P_I e P_C, e l'integrazione è intesa nel senso di Lebesgue (Billingsley, 1986).

Le misure P_I e P_C in genere non sono note, quindi non è possibile ricavare soluzioni analitiche per gli integrali, uno per ogni configurazione θ, dell'Equazione 3.2. Per sopperire a tale limitazione, gli integrali definiti nell'Equazione 3.2 sono stimati col metodo statistico Monte Carlo basandosi su un insieme di istanze disponibili per l'ottimizzazione dei parametri, come spiegato in maggior dettaglio nel Paragrafo 3.2.

Con una tale descrizione del *problema di configurazione* si assume che nessuna informazione sulle prestazioni delle varie configurazioni candidate

[6] Anche qui valgono le stesse considerazioni espresse per P_I.

possa essere ottenuta prima della loro esecuzione su una data istanza. In questo senso, l'ipotesi è che le istanze sono indistinguibili a priori. In molte situazioni pratiche, si hanno informazioni a priori sul fatto che si possono presentare vari tipi di istanze con caratteristiche diverse. In tal caso, tutte le conoscenze a priori dovrebbero essere usate al fine di raccogliere le istanze in classi omogenee, e trovare, per ogni classe, la miglior configurazione.

3.2 Metodi di racing

Al fine di mettere in evidenza alcune delle difficoltà associate al *problema di configurazione* descritto nell'Equazione 3.1, descriviamo un possibile approccio a *forza bruta*.

Un approccio a forza bruta può consistere nello stimare le quantità definite nell'Equazione 3.2 per mezzo di un numero *sufficientemente grande* di esecuzioni di ogni candidato su un insieme *sufficientemente grande* di istanze. Si selezionerà dunque la configurazione θ per cui il valore della stima di $\mathcal{C}(\theta)$ risulti essere il più piccolo.

Tale approccio presenta alcuni svantaggi: innanzitutto, occorre stabilire, preventivamente alla computazione e senza alcun criterio che ci aiuti, un insieme *sufficientemente grande* di istanze disponibili per la configurazione, esponendosi al doppio rischio di considerare poche istanze, che non permetterebbero di ottenere una stima attendibile, oppure di considerarne troppe, per cui si richiederebbe un eccessivo uso di potenza di calcolo. Manca, inoltre, un criterio per decidere quante esecuzioni per ogni configurazione debbano essere fatte su ogni istanza, per fronteggiare la natura stocastica delle metaeuristiche. Infine, ogni configurazione riceve la stessa quantità di risorse di calcolo: le configurazioni peggiori sono eseguite per tanto tempo quanto le

configurazioni migliori.

3.2.1 Gli algoritmi di racing

Gli algoritmi di racing sono progettati per permettere un'allocazione efficiente delle risorse di calcolo tra le configurazioni candidate, risolvendo, in tal modo, l'ultimo dei tre problemi che affliggono l'approccio a forza bruta sopra esposto. Allo stesso tempo, si fornisce una soluzione indiretta ai primi due problemi che affliggono l'approccio a forza bruta, ossia il dover fissare il numero di istanze disponibili per la configurazione e il numero di esecuzioni per ogni istanza.

Per fare questo, gli algoritmi di racing valutano in sequenza le configurazioni candidate e scartano quelle "peggiori" non appena si raccoglie sufficiente evidenza statistica a loro sfavore. L'eliminazione dei candidati peggiori velocizza il procedimento e permette di valutare le configurazioni promettenti su un maggior numero di istanze, ottenendo in tal modo una stima migliore del loro comportamento. La Figura 3.1 offre una rappresentazione grafica di come i due diversi metodi, F-Race e l'approccio a forza bruta, allocano le risorse di computazione alle configurazioni candidate.

Supponiamo di disporre di una sequenza casuale $\underline{i}$ di istanze disponibili per la configurazione, dove il generico termine k-esimo $\underline{i}_k$ è selezionato da I in accordo a P_I, in modo indipendente per ogni k. Immaginiamo che $\underline{i}$ possa essere estesa indefinitamente e con costo trascurabile, in modo da poter continuare a campionare da I. Con la notazione $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ indichiamo un vettore di k termini, il cui generico termine l-esimo è il costo $\boldsymbol{c}(\theta, \underline{\boldsymbol{i}}_l)$ della miglior soluzione trovata per la configurazione θ sull'istanza $\underline{\boldsymbol{i}}_l$ eseguita per $t(\underline{\boldsymbol{i}}_l)$ secondi. È quindi chiaro come sia possibile, per una data configurazione

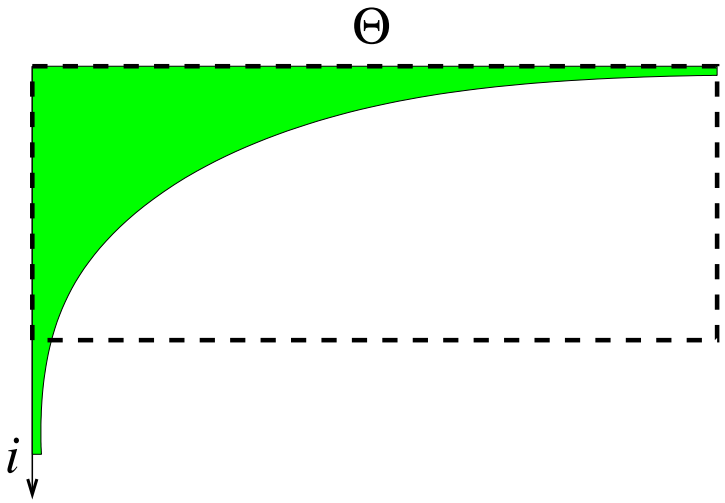

Figura 3.1: Rappresentazione visiva delle risorse di computazione allocate alle configurazioni candidate dai due metodi. La superficie racchiusa nel rettangolo tratteggiato rappresenta la quantità di computazione del metodo a forza bruta, mentre l'area colorata quella del metodo di racing. La figura appare in (Birattari *et al.*, 2002) e ne è stato gentilmente concesso l'uso da parte degli autori, per questa tesi.

θ, ottenere il vettore $\underline{\boldsymbol{c}}^k$, di lunghezza k, aggiungendo in coda a $\underline{\boldsymbol{c}}^{k-1}$ il costo della miglior soluzione trovata per la configurazione θ sull'istanza $\underline{\boldsymbol{i}}_{\boldsymbol{k}}$ di $\underline{\boldsymbol{i}}$.

Un algoritmo di racing affronta il problema di ottimizzazione descritto dall'Equazione 3.1 generando una sequenza di insiemi di configurazioni candidate, dove ogni insieme è incluso nel precedente:

$$\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \cdots$$

iniziando da $\Theta_0 = \Theta$. Il passaggio dall'insieme Θ_{k-1} a Θ_k è ottenuto scartando alcune configurazioni che sembrano essere subottimali sulla base delle informazioni disponibili al passo k.

Al passo k, quando l'insieme dei candidati è Θ_{k-1}, si considera una nuova istanza $\underline{\boldsymbol{i}}_k$. Ogni configurazione candidata $\theta \in \Theta_{k-1}$ opera su $\underline{\boldsymbol{i}}_k$, e ogni costo osservato $\boldsymbol{c}(\theta, \underline{\boldsymbol{i}}_k)$ è aggiunto in coda al rispettivo $\underline{\boldsymbol{c}}^{k-1}$ per formare i nuovi vettori $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$, uno per ogni θ. Il passo k termina con la definizione

dell'insieme Θ_k, eliminando da Θ_{k-1} le configurazioni che sembrano subottimali, alla luce di alcuni test statistici che confrontano i vettori $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$. La descrizione del test considerato in questa tesi è data nel Paragrafo 3.2.2. Si noti che, per ogni θ, ogni componente del vettore $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$, cioè ogni costo $\boldsymbol{c}(\theta, \boldsymbol{i})$ della migliore soluzione trovata da una singola esecuzione di θ su una generica istanza $\boldsymbol{i}$ estratta in accordo a P_I, è una stima di $\mathcal{C}(\theta)$, definito nell'Equazione 3.2. Il valor medio del campionamento di $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ è dunque esso stesso una stima di $\mathcal{C}(\theta)$, e può essere usato per confrontare le prestazioni ottenute dalle varie configurazioni.

La procedura fin qui descritta è iterata, e si ferma nel caso in cui tutte le configurazioni, tranne una, sono scartate, oppure quando è raggiunto un tempo predefinito T di computazione. Ossia, la procedura si ferma prima di considerare la $(k+1)$-esima istanza se $\sum_{l=1}^{k} t(\underline{\boldsymbol{i}}_{l+1}) \left|\Theta_l\right| > T$.

3.2.2 Il metodo F-Race

L'algoritmo di racing adoperato in questa tesi, e a cui ci riferiamo come F-Race, è basato sul test di Friedman, un metodo statistico per il test delle ipotesi, conosciuto anche con il nome di statistica di Friedman dell'analisi della varianza per ranghi con due criteri di classificazione[7] (Conover, 1999).

Per poter descrivere il test, assumiamo che F-Race abbia raggiunto il passo k, e che $n = |\Theta_{k-1}|$ configurazioni siano ancora presenti. Il test di Friedman assume che i costi osservati $(\underline{\boldsymbol{c}}^k(\theta_1, \underline{\boldsymbol{i}}_l), \underline{\boldsymbol{c}}^k(\theta_2, \underline{\boldsymbol{i}}_l), \ldots, \underline{\boldsymbol{c}}^k(\theta_n, \underline{\boldsymbol{i}}_l))$ siano k variabili casuali, n-variate, mutuamente indipendenti, chiamate *blocchi* (Dean e Voss, 1999), dove ogni blocco corrisponde ai risultati delle computazioni di ogni configurazione presente al passo k sull'istanza $\underline{\boldsymbol{i}}_l$. All'interno di ogni blocco si assegna il rango alle quantità $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}}_l)$, dalle più piccole alle

[7] Il corrispettivo termine inglese è *Friedman two-way analysis of variance by ranks.*

più grandi. In caso di parità di valore si assegna il rango medio. Per ogni configurazione $\theta_j \in \Theta_{k-1}$, sia R_{lj} il rango di θ_j all'interno del blocco l, e sia $R_j = \sum_{l=1}^{k} R_{lj}$ la somma dei ranghi su tutte le istanze $\underline{i}_l$, con $1 \leq l \leq k$. Il test di Friedman considera la seguente statistica (Conover, 1999):

$$T = \frac{(n-1)\sum_{j=1}^{n}(R_j - \frac{k(n+1)}{2})^2}{\sum_{l=1}^{k}\sum_{j=1}^{n} R_{lj}^2 - \frac{kn(n+1)^2}{4}}.$$

Sotto l'ipotesi nulla che tutti i possibili ordinamenti dei candidati all'interno di ogni blocco siano ugualmente probabili, T è distribuita approssimativamente come χ^2 con $n-1$ gradi di libertà. Se il valore osservato di T eccede il quantile $1-\alpha$ di tale distribuzione, l'ipotesi nulla è respinta con un *livello di significatività* α, in favore dell'ipotesi che almeno un candidato tende ad avere prestazioni migliori di almeno un altro.

Se l'ipotesi nulla è respinta, siamo giustificati nell'effettuare le comparazioni fra coppie di candidati. I candidati θ_j e θ_h sono considerati diversi se:

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k(1-\frac{T}{k(n-1)})(\sum_{l=1}^{k}\sum_{j=1}^{n} R_{lj}^2 - \frac{kn(n+1)^2}{4})}{(k-1)(n-1)}}} > t_{1-\frac{\alpha}{2}},$$

dove $t_{1-\frac{\alpha}{2}}$ è il quantile $1-\frac{\alpha}{2}$ della distribuzione t (Conover, 1999).

In F-Race, se al passo k l'ipotesi nulla delle comparazioni aggregate non è respinta, tutte le configurazioni candidate in Θ_{k-1} passano a Θ_k. D'altra parte, se l'ipotesi nulla è respinta, si eseguono le comparazioni tra il miglior candidato e ogni altra configurazione. Tutti i candidati che risultano significativamente peggiori del candidato migliore sono scartati, e non passano a Θ_k.

In F-Race, l'importanza del rango è duplice: da un lato per la natura non parametrica di un test basato sul rango, dall'altro per il modo naturale con

cui si implementa uno schema a blocchi (Dean e Voss, 1999). Esaminiamo, dunque, questi due aspetti.

Il merito maggiore di un'analisi non parametrica è che non si richiede la formulazione di alcuna ipotesi sulla distribuzione delle osservazioni. Per una trattazione dei pro e dei contro relativi agli approcci parametrici e non parametrici si rimanda alla lettura di Conover (1999). Qui ci limitiamo a menzionare alcuni fatti comunemente accettati sulle ipotesi dei test parametrici e non parametrici. Quando le ipotesi formulate dai test parametrici sono soddisfatte, i test parametrici hanno un maggior potere risolutivo rispetto ai test non parametrici, e in generale richiedono un minor sforzo computazionale. Inoltre, quando sono disponibili grandi quantità di dati, l'ipotesi per l'applicazione dei test parametrici tende ad essere soddisfatta, in virtù del teorema del limite centrale. Infine, è risaputo che il t-test, un classico test parametrico, è robusto anche in mancanza di alcune ipotesi, in particolare sulla forma normale della distribuzione dei dati: quando l'ipotesi della forma normale della distribuzione dei dati non è soddisfatta, il potere risolutivo del t-test degrada "dolcemente".

Per quanto concerne il *problema della configurazione delle metaeuristiche*, le argomentazioni a favore dei test parametrici non rivestono un ruolo determinante. Infatti, poiché noi vogliamo ridurre il numero di candidati il più velocemente possibile, ci troviamo ad avere a che fare con un numero piccolo di campioni, per cui non possiamo invocare il teorema del limite centrale. Proprio su questo numero piccolo di campioni vogliamo avere il massimo potere risolutivo. Anche il minor costo computazionale dei test parametrici rispetto ai test non parametrici non è un fattore determinante, in quanto la maggior parte del tempo è spesa nell'esecuzione delle configurazioni candidate.

L'importanza che riveste il rango in F-Race è anche quella di implementare in modo naturale uno schema a blocchi (Dean e Voss, 1999). Le variazioni nei costi osservati $\boldsymbol{c}$ è dovuta a tre fonti distinte: le metaeuristiche sono algoritmi intrinsecamente stocastici, le istanze possono essere molto differenti le une dalle altre, e, infine, alcune configurazioni hanno prestazioni migliori di altre. Quest'ultima fonte di variazione è quella che è di interesse nel *problema della configurazione*, mentre le altre possono essere considerate elementi di disturbo. I blocchi sono un modo efficace per normalizzare il costo osservato su istanze differenti. Concentrandoci solamente sul rango delle differenti configurazioni per ogni istanza, i blocchi eliminano il rischio che le variazioni dovute alle differenze tra le istanze spazzino via le variazioni dovute alle differenze tra le configurazioni.

3.3 Risultati sperimentali delle configurazioni

Abbiamo applicato F-Race a due algoritmi tra quelli che abbiamo implementato per risolvere il problema della Compilazione degli Orari dei Corsi Universitari: Ant Colony System (descritto nel Paragrafo 2.4.2) e Algoritmo Memetico (descritto nel Paragrafo 2.4.4). La classe di istanze considerate, per entrambi gli algoritmi, è quella classificata come *small* in accordo alla Tabella 1.1 di pagina 24, per la quale si vuole selezionare la "miglior configurazione possibile". Tramite il generatore di istanze fornito da Ben Paechter, illustrato nel Paragrafo 1.3.3, abbiamo generato 500 istanze appartenenti alla classe *small*, per formare l'insieme $\underline{\boldsymbol{i}}$ delle istanze disponibili per la configurazione. Il tempo di esecuzione concesso ad ogni configurazione candidata ammonta a 90 secondi per ogni istanza. Il livello di significatività

con cui si rigetta l'ipotesi nulla è 0.95,[8] e si può cominciare a scartare le configurazioni dal passo $k = 6$.

Per entrambi gli algoritmi il metodo F-Race è stato eseguito su una delle otto macchine componenti il Beowulf (descritto nel Paragrafo 2.3) di cui è dotato IRIDIA. Si tratta di un PC dotato di una CPU AMD Athlon Thunderbird 1100 MHz e di 256 MB di RAM, su cui è installato il sistema operativo Red Hat Linux 7.0 con kernel 2.2.16-22.

Il Paragrafo 3.3.1 illustra l'applicazione di F-Race all'algoritmo Ant Colony System, mentre il Paragrafo 3.3.2 illustra l'applicazione di F-Race all'Algoritmo Genetico.

3.3.1 La configurazione dell'Algoritmo Ant Colony System

Il tempo massimo di esecuzione per F-Race su ACS è stato fissato in una settimana, ed il tempo per cui ogni configurazione poteva operare su un'istanza appartenente alla classe *small* in 90 secondi. L'insieme Θ delle configurazioni candidate per Ant Colony System è stato definito specificando i valori per i parametri dell'algoritmo, riportati in Tabella 3.1, e generando tutte le possibili combinazioni di tali valori, ottenendo in tal modo 648 configurazioni candidate. La scelta dei valori per i parametri è stata discussa con Marco Dorigo,[9] l'ideatore, assieme a Luca Maria Gambardella, di Ant Colony System. I risultati dell'esperimento, il cui andamento parziale è riportato in Tabella 3.2 (l'andamento completo è riportato in Appendice B), hanno identificato,

[8] Ci sono al più 5 possibilità su 100 che si rigetti l'ipotesi che avrebbe dovuto essere accettata, cioè siamo "fiduciosi" al 95% di aver preso la decisione giusta.

[9] e-mail: mdorigo@ulb.ac.be

Tabella 3.1: L'insieme Θ delle configurazioni candidate per l'Algoritmo ACS è stato ottenuto generando tutte le possibili combinazioni dei valori dei parametri: m indica il numero di individui nella popolazione; τ_0 è il valore a cui si inizializzano gli elementi della matrice del feromone; α controlla il peso dell'informazione stigmergica; β e γ controllano il peso delle informazioni euristiche relative ai vincoli inderogabili e derogabili, rispettivamente; $\psi \in [0,1]$ controlla la diversificazione nel processo di costruzione; $h(j)$ indica il numero di passi per cui si applica la routine di ricerca locale, in funzione del passo j; $\rho \in [0,1]$ e Q controllano la quantità di feromone aggiunta dalla regola di aggiornamento.

Parametro	*valore 1*	*valore 2*	*valore3*
m	3	5	7
τ_0	0.5	10.0	
α	1.0		
β	0	1.5	2.5
γ	0	1.5	2.5
ρ	0.1	0.3	
ψ	0.1	0.3	
$h(j)$	$\begin{cases} 20\,000 & j = 1 \\ 10\,000 & j \geq 2 \end{cases}$		
Q	10^3	10^4	10^5

come configurazione migliore, quella con i valori dei parametri riportati in Tabella 3.3.

3.3.2 La configurazione dell'Algoritmo Memetico

Al pari della configurazione dell'algoritmo ACS, anche per l'Algoritmo Memetico è stato deciso di eseguire F-Race al massimo per una settimana, con ogni configurazione che poteva operare per 90 secondi sulle istanze appartenenti alla classe *small*. I valori dei parametri da verificare per l'Algoritmo Memetico, sono riportati in Tabella 3.4. La scelta dei valori è stata discussa con Joshua Knowles,[10] esperto nel settore della Computazione Evolutiva. L'insieme Θ delle configurazioni candidate per l'Algoritmo Memetico è stato definito generando tutte le possibili combinazioni di tali valori, ottenendo in tal modo 675 configurazioni candidate. I risultati dell'esperimento, il cui andamento parziale è riportato in Tabella 3.5 (l'andamento completo è riportato in Appendice B), hanno identificato, come configurazione migliore, quella con i valori dei parametri riportati in Tabella 3.6.

[10] e-mail: jknowles@ulb.ac.be

Tabella 3.2: Andamento di F-Race su ACS. Sono riportate soltanto le situazioni delle iterazioni in cui, rispetto all'iterazione precedente, o si propone una diversa configurazione candidata θ, oppure si riduce il numero di elementi nell'insieme Θ. Alla k-esima iterazione dell'algoritmo, $|\Theta_{k-1}|$ indica il numero di configurazioni candidate ancora in gara.

k	$\lvert\Theta_{k-1}\rvert$	Indice miglior configurazione	Numero di esperimenti effettuati
1	648	**5**	648
2	648	**493**	1296
3	648	**147**	1944
5	648	**596**	3240
6	**144**	596	3888
7	144	**361**	4032
8	144	**416**	4176
9	144	**596**	4320
10	144	**297**	4464
11	**68**	**181**	4608
15	68	**416**	4880
18	68	**278**	5084
20	68	**596**	5220
22	68	**278**	5356
32	**52**	**608**	6036
36	52	**214**	6244
40	52	**206**	6452

Tabella 3.3: Valori dei parametri della configurazione $\theta_{206} \in \Theta$ selezionata da F-Race per l'Algoritmo ACS. Per le definizioni dei parametri si consulti la Tabella 3.1.

Parametro	*valore*
m	5
τ_0	0.5
α	1.0
β	2.5
γ	2.5
ρ	0.1
ψ	0.3
$h(j)$	$\begin{cases} 20\,000 & j = 1 \\ 10\,000 & j \geq 2 \end{cases}$
Q	10^5

Tabella 3.4: L'insieme Θ delle configurazioni candidate per l'Algoritmo Memetico è stato ottenuto generando tutte le possibili combinazioni dei valori dei parametri: m indica il numero di individui nella popolazione; n indica la dimensione del torneo con cui sono selezionati i genitori per la riproduzione; $prob_{sel}$ è la probabilità con cui si sceglie dal primo genitore l'intervallo temporale da associare all'evento; h indica il numero di passi per cui si applica la routine di ricerca locale.

Parametro	*valore*
m	$2 \leq m \leq 10$
n	$(2 \leq n \leq 10) \wedge (n \leq m)$
$prob_{sel}$	{0.20, 0.35, 0.5, 0.65, 0.8}
h	{15*1e3, 15*1e4, 1e7}

Tabella 3.5: Andamento di F-Race su AM. Sono riportate soltanto le situazioni delle iterazioni in cui, rispetto all'iterazione precedente, o si propone una diversa configurazione candidata θ, oppure si riduce il numero di elementi nell'insieme Θ. Alla k-esima iterazione dell'algoritmo, $|\Theta_{k-1}|$ indica il numero di configurazioni candidate ancora in gara.

k	$\lvert\Theta_{k-1}\rvert$	Indice miglior configurazione	Numero di esperimenti effettuati
1	675	**14**	675
2	675	**101**	1350
3	675	**440**	2025
6	**119**	**22**	4050
8	119	**440**	4288
9	119	**15**	4407
10	119	**440**	4526
11	119	**39**	4645
14	**63**	39	5002
16	63	**5**	5128
19	63	**39**	5317
20	63	**5**	5380
31	63	**15**	6073
34	63	**5**	6262
38	63	**15**	6514

Tabella 3.6: Valori dei parametri della configurazione $\theta_{15} \in \Theta$ selezionata da F-Race per l'Algoritmo Memetico. Per le definizioni dei parametri si consulti la Tabella 3.4.

Parametro	*valore*
m	8
n	3
$prob_{sel}$	0.2
h	15000

Capitolo 4

Esperimenti

Parte del lavoro svolto per la compilazione della tesi è inquadrato nell'attività di ricerca del *Metaheuristics Network*[1] che, durante il periodo Marzo 2002 - Marzo 2003, si è occupato di studiare il problema della Compilazione degli Orari dei Corsi Universitari, di cui abbiamo illustrato la formulazione nel Paragrafo 1.3.2.

L'autore della tesi ha partecipato al lavoro di ricerca svolto a IRIDIA, ed è stato incaricato di implementare un algoritmo della metaeuristica ACO, illustrata nel Paragrafo 2.2.5, per la risoluzione del problema sotto studio. I risultati di quell'attività sono esposti nel Paragrafo 4.2. Nel restante tempo abbiamo implementato gli algoritmi descritti nel Paragrafo 2.4, oltre ad aver applicato il metodo F-Race, descritto nel Paragrafo 3.2, agli algoritmi Ant Colony System e Algoritmo Memetico. Le nuove implementazioni, e la versione "ottimizzata" di Ant Colony System, sono state sottoposte ad una parte degli stessi esperimenti effettuati nella fase precedente dal *Metaheuristics Network*, i cui risultati sono illustrati nel Paragrafo 4.3.

[1] Per una descrizione della rete di addestramento alla ricerca, si veda l'Introduzione.

4.1 Come leggere le tabelle ed i grafici

Per ogni istanza del problema considerata, abbiamo verificato se le differenze tra le soluzioni trovate dai diversi algoritmi fossero statisticamente significative. Abbiamo usato il test *Pairwise Wilcoxon rank sum* (si veda Conover (1999)) con correzione dei *p-value*[2] per mezzo del metodo di Holm (1979). Nelle tabelle associate ai grafici, sono riportati per ogni coppia (A,B) di algoritmi i *p-value* per l'ipotesi nulla "La distribuzione delle soluzioni generate da A e la distribuzione delle soluzioni generate da B sono uguali".

Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 non ci permettono di rifiutare l'ipotesi nulla.

I diagrammi mostrano le distribuzioni delle soluzioni ammissibili (che soddisfano tutti i vincoli inderogabili) che sono state trovate dai diversi algoritmi durante le esecuzioni indipendenti. I risultati di tutte le esecuzioni sulla stessa istanza sono ordinati per qualità della soluzione (numero di violazioni dei vincoli derogabili) per determinarne il rango (soluzioni con la stessa qualità hanno come rango la posizione media nell'ordinamento). Una soluzione non ammissibile (che viola almeno un vincolo inderogabile) è considerata peggiore di qualsiasi soluzione ammissibile, pertanto è ordinata successivamente a queste. Le soluzioni sono raggruppate per algoritmo. Nei *boxplot* la mediana è rappresentata da una barra, la *box* rappresenta l'intervallo compreso tra il quantile 25% e 75%; le "aste"[3] si estendono ai punti più estremi che non siano

[2]Il *p-value* del test è la probabilità che un valore più estremo di quello che si è attualmente verificato, nelle direzioni dell'ipotesi alternativa, potesse verificarsi se l'ipotesi nulla fosse stata vera.

[3]L'equivalente termine inglese è *whisker*.

più di 1,5 volte l'interquartile; i cerchietti rappresentano i punti che rimangono all'esterno di tale intervallo. La percentuale di soluzioni non ammissibili prodotta da ciascun algoritmo è rappresentata da un istogramma.

4.2 Il confronto interno al *Metaheuristics Network*

La prima fase dell'attività del *Metaheuristics Network* è durata da Marzo ad Aprile 2002. Durante tale fase sono stati implementati i seguenti algoritmi, facendo uso dello Starter Kit descritto nel Paragrafo 2.3.1:

- **IRIDIA:** l'autore della tesi ha implementato l'Algoritmo ACS, descritto nel Paragrafo 2.4.2. All'interno dei grafici è identificato dalla sigla **ACO**;
- **INTELLEKTIK:** Luis Paquete[4] ha implementato un algoritmo di Ricerca Locale Iterata, di cui abbiamo parlato nel Paragrafo 2.2.1. All'interno dei grafici è identificato dalla sigla **ILS**;
- **INTELLEKTIK:** Marco Chiarandini[5] ha implementato un algoritmo di Simulated Annealing, di cui abbiamo parlato nel Paragrafo 2.2.2. All'interno dei grafici è identificato dalla sigla **SA**;
- **ECRG:** Olivia Rossi-Doria[6] ha implementato un algoritmo di Computazione Evolutiva, di cui abbiamo parlato nel Paragrafo 2.2.4. All'interno dei grafici è identificato dalla sigla **GA**;

[4]e-mail: lpaquete@intellektik.informatik.tu-darmstadt.de
[5]e-mail: machud@intellektik.informatik.tu-darmstadt.de
[6]e-mail: o.rossi-doria@napier.ac.uk

- **IDSIA:** Monaldo Mastrolilli[7] ha implementato un algoritmo di Tabu Search, di cui abbiamo parlato nel Paragrafo 2.2.3. All'interno dei grafici è identificato dalla sigla **TS**.

Alla fine del mese di Aprile 2002 sono stati eseguiti degli esperimenti il cui scopo è stato quello di confrontare il comportamento delle varie implementazioni sulle tre classi di istanze del problema della COMPILAZIONE DEGLI ORARI DEI CORSI UNIVERSITARI, illustrate nel Paragrafo 1.3.3, le cui caratteristiche sono riportate in Tabella 1.1.

La definizione di una metodologia di sperimentazione, uno degli obiettivi scientifici del *Metaheuristics Network*, ha tenuto conto della potenza di calcolo a disposizione di IRIDIA, il nodo incaricato dell'esecuzione degli esperimenti. Come descritto nel Paragrafo 2.3, tutti gli esperimenti sono stati eseguiti sul cluster Beowulf di 8 PC (il nono PC è stato aggiunto alcuni mesi dopo la fine del primo gruppo di esperimenti). Tutti i PC sono dotati di una CPU AMD Athlon Thunderbird 1100 MHz con 256 MB di RAM, e adottano come sistema operativo Red Hat Linux 7.0 con kernel 2.2.16-22. Ogni nodo della rete di ricerca ha fornito ad IRIDIA i sorgenti in C++ degli algoritmi implementati, e gli eseguibili sono stati generati compilando il codice sul Beowulf nell'ambiente **gcc** 2.95.3, **glibc** 2.2.4, **binutils** 2.10.0.18-1.

Uno stesso insieme di istanze del problema, su cui eseguire esperimenti con tutti gli algoritmi implementati, è stato creato adoperando il generatore di istanze scritto da Olivia Rossi-Doria e Ben Paechter,[8] descritto nel Paragrafo 1.3.3. L'insieme è composto da 5 istanze appartenenti alla classe *small* (identificate nei grafici come `small01.tim`, `small02.tim`, `small03.tim`, `small04.tim` e `small05.tim`), 5 istanze appartenenti alla

[7] e-mail: monaldo@idsia.ch

[8] e-mail: b.paechter@napier.ac.uk

classe *medium* (identificate nei grafici come `medium01.tim`, `medium02.tim`, `medium03.tim`, `medium04.tim` e `medium05.tim`) e 2 istanze appartenenti alla classe *large* (identificate nei grafici come `large01.tim` e `large02.tim`).

Ognuno dei 5 algoritmi implementati è stato fatto operare: 500 volte, per 90 secondi ogni volta, su ognuna delle 5 istanze *small*; 50 volte, per 900 secondi ogni volta, su ognuna delle 5 istanze *medium*; 20 volte, per 9000 secondi ogni volta, su ognuna delle 2 istanze *large*.

Si noti che, nel caso in cui si fosse dovuto usare un singolo PC (di potenza equivalente ad uno dei nodi del cluster Beowulf) per l'esecuzione sequenziale di tutti gli esperimenti, i risultati sarebbero stati disponibili solamente dopo 47 giorni circa. L'uso dell'infrastruttura Beowulf, con i suoi 8 PC in parallelo, ci ha permesso di ridurre l'attesa a 6 giorni.

Di seguito riportiamo i grafici della sola prima istanza di ogni classe (`small01.tim`, `medium01.tim` e `large01.tim`) in quanto i risultati ottenuti dagli algoritmi all'interno delle singole classi sono similari. I risultati completi di tutte le istanze sono riportati in Appendice A.1.

4.2.1 Risultati istanze *small*

Ognuna delle 5 istanze *small* del problema su cui abbiamo eseguito gli esperimenti ha fornito risultati per i *p-value* molto piccoli, minori del valore 0.05 che ci permette di rifiutare l'ipotesi nulla che le distribuzioni delle soluzioni prodotte dalle coppie di algoritmi siano uguali. Uno sguardo ai grafici mostra come gli algoritmi ACO, ILS e SA riescano a produrre, con elevata frequenza, soluzioni ottime o vicine all'ottimo.

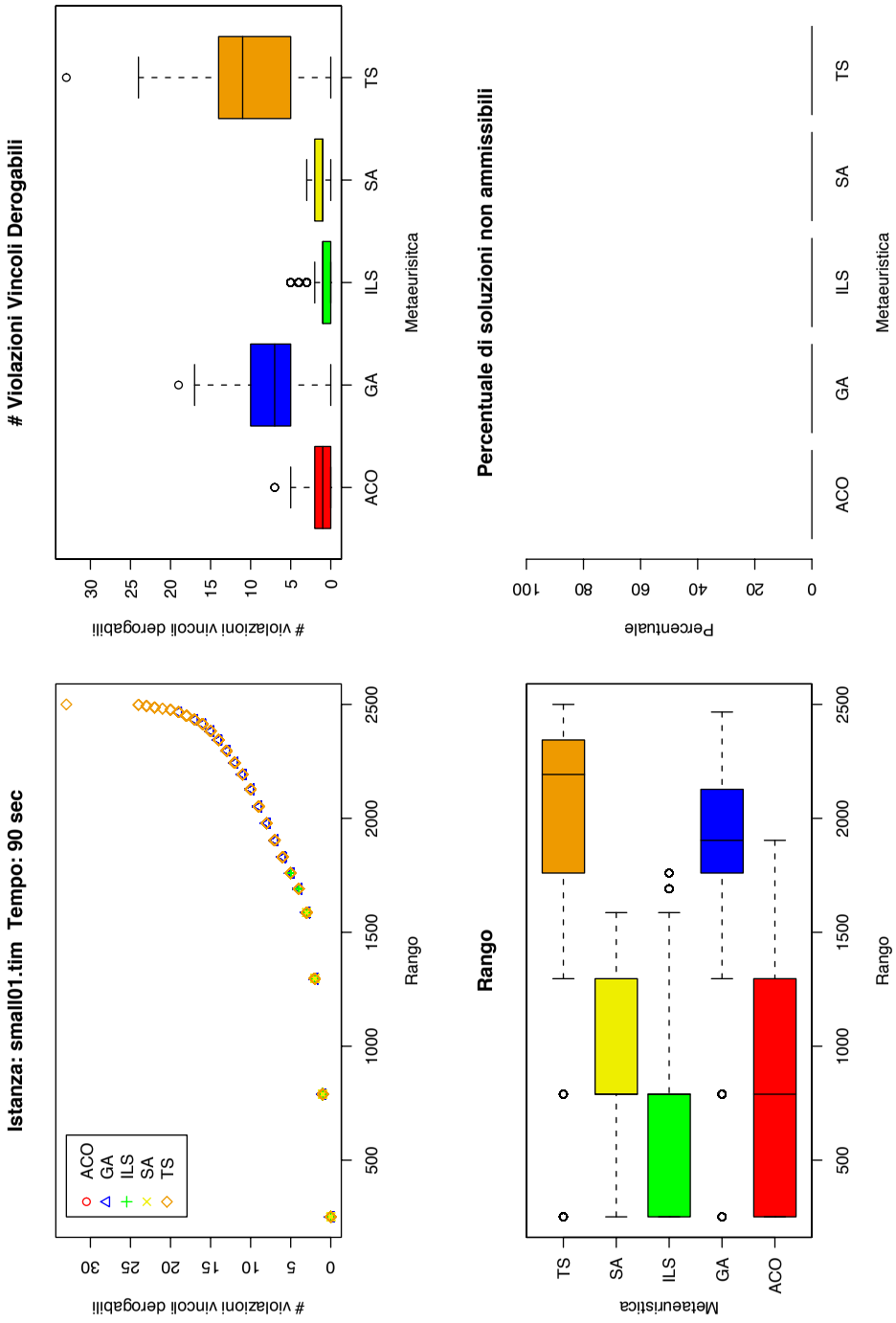

Figura 4.1: Risultati istanza `small01.tim`

Tabella 4.1: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	$< 2^{-16}$			
ILS	1.5^{-5}	$< 2^{-16}$		
SA	2^{-9}	$< 2^{-16}$	$< 2^{-16}$	
TS	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

4.2.2 Risultati istanze *medium*

Anche i risultati forniti dalle 5 istanze *medium* del problema su cui abbiamo eseguito gli esperimenti hanno prodotto risultati per i *p-value* molto piccoli, minori del valore di 0.05, che ci permette, anche in questo caso, di rifiutare l'ipotesi nulla che le distribuzioni delle soluzioni prodotte dalle coppie di algoritmi siano uguali, ad eccezione della coppia <GA, TS>, per cui non possiamo rifiutare l'ipotesi nulla. Dai grafici emerge chiaramente la superiorità dell'algoritmo SA sulle istanze considerate. A differenza di quanto visto per le istanze *small*, ACO risulta essere il peggiore algoritmo. È interessante notare come SA non sempre sia in grado di trovare una soluzione ammissibile.

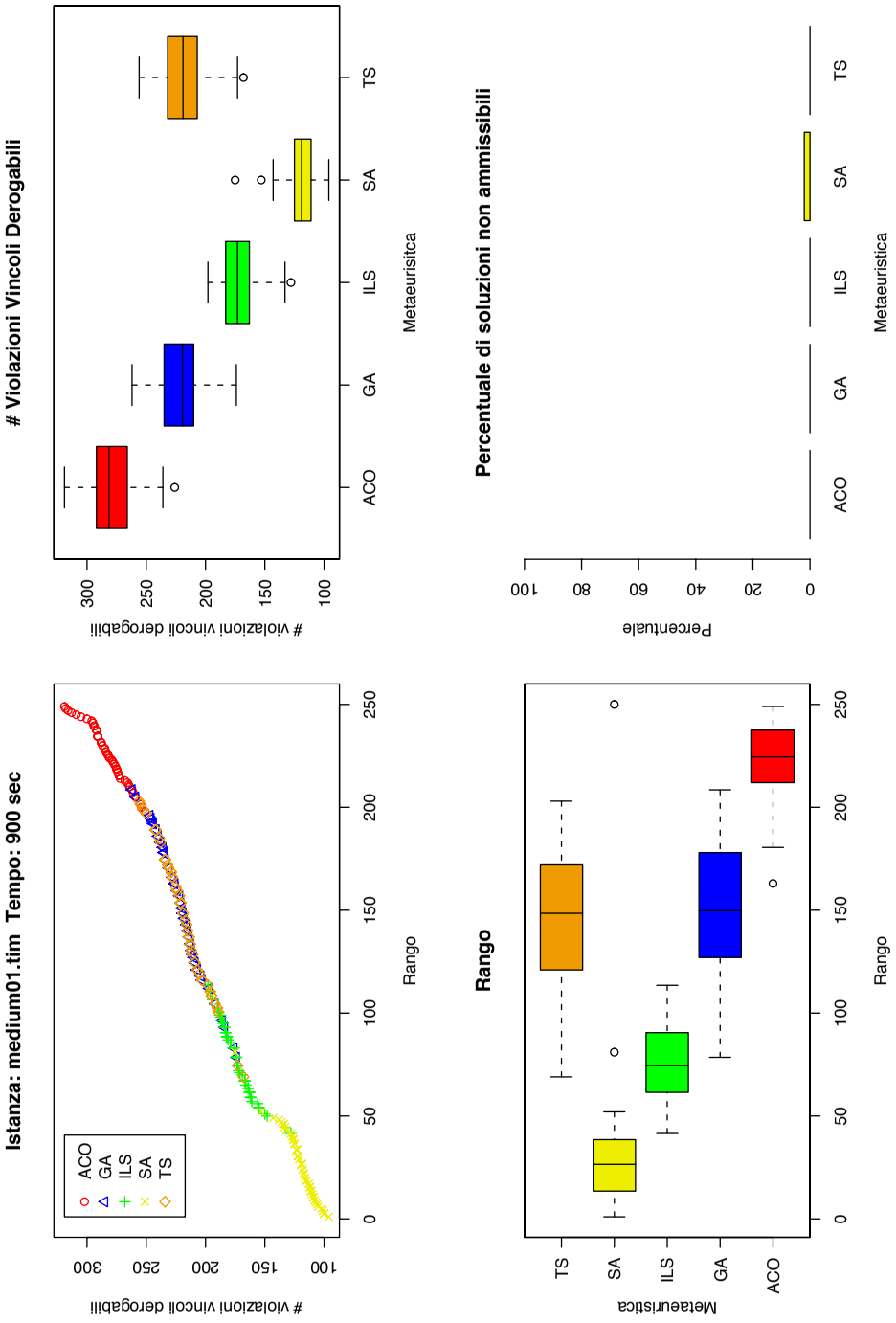

Figura 4.2: Risultati istanza `medium01.tim`

Tabella 4.2: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `medium01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	7.8^{-16}			
ILS	$< 2^{-16}$	1.4^{-15}		
SA	9.3^{-16}	9.3^{-16}	3.2^{-15}	
TS	5.8^{-16}	**0.54**	2^{-15}	1.1^{-15}

4.2.3 Risultati istanze *large*

L'elevato tempo di computazione richiesto per eseguire gli esperimenti sulle istanze della classe *large* ci ha permesso di considerarne soltanto 2. In entrambi i casi, gli algoritmi hanno avuto difficoltà a trovare soluzioni ammissibili. Le rare volte che le soluzioni sono trovate, ACO e ILS hanno prodotto il minor numero di violazioni dei vincoli derogabili. Si noti come SA, a differenza dei buoni risultati prodotti nelle istanze della classe *small* e *medium*, non sia mai riuscito a trovare una soluzione ammissibile in nessuna esecuzione sulle due istanze *large*.

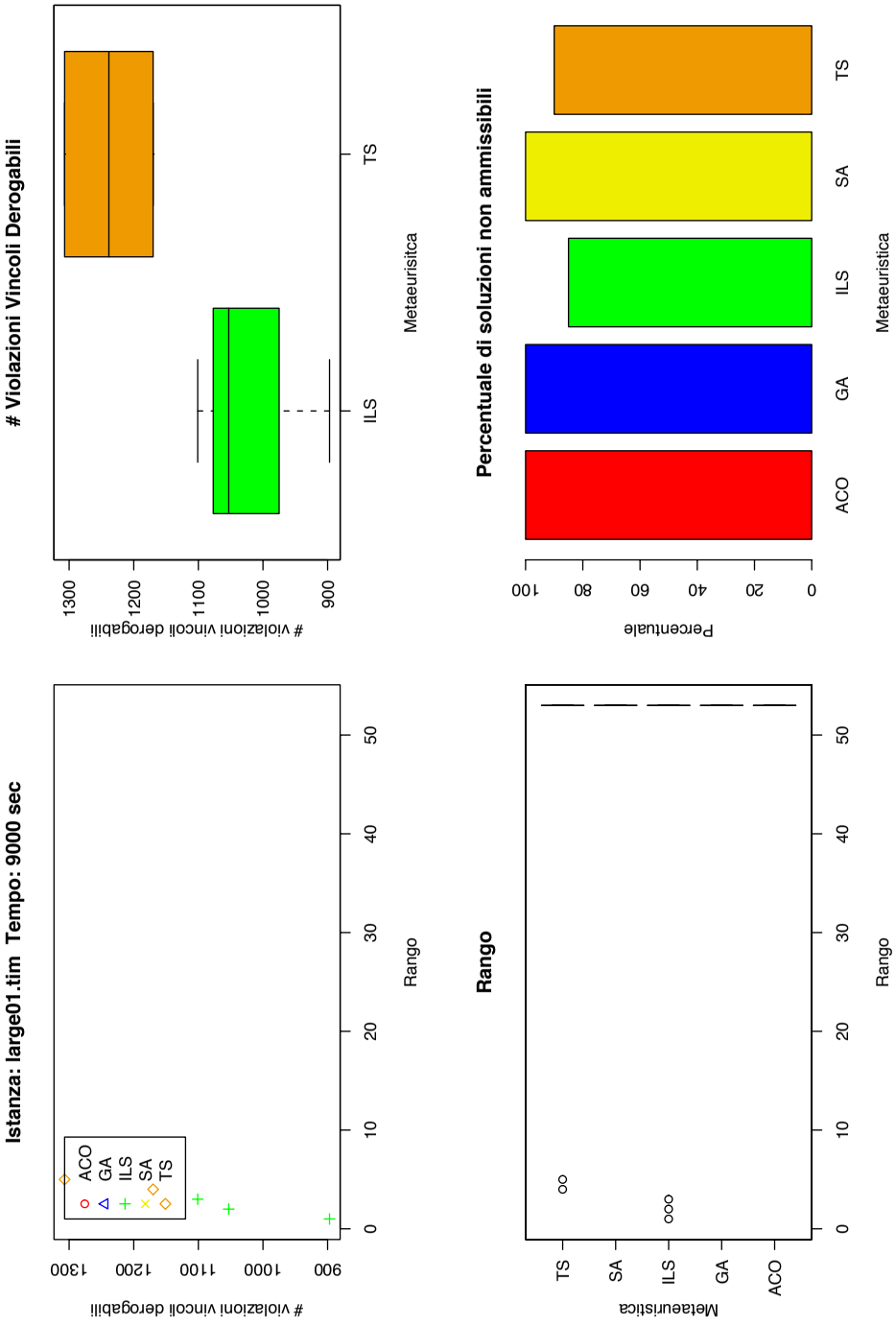

Figura 4.3: Risultati istanza `large01.tim`

Tabella 4.3: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `large01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	-			
ILS	**0.8**	**0.8**		
SA	-	-	**0.8**	
TS	**1**	**1**	**1**	**1**

4.3 Il confronto esteso alle nostre implementazioni

Dopo il mese di Aprile 2002, abbiamo implementato due ulteriori algoritmi: l'Algoritmo Memetico, descritto nel Paragrafo 2.4.4, e un algoritmo di Ricerca Locale Iterata, descritto nel Paragrafo 2.4.3. Si è anche proceduto alla ricerca della migliore configurazione dei parametri per gli algoritmi Ant Colony System e Algoritmo Memetico usando il metodo automatico F-Race, come descritto nel Paragrafo 3.3. La potenza computazionale a nostra disposizione non era adeguata per eseguire, in un tempo compatibile con le esigenze di stesura della tesi, F-Race sulle istanze di classe *medium* o *large* ed i successivi esperimenti; pertanto, dato lo scarso tempo a nostra disposizione, abbiamo deciso di applicare la metodologia F-Race alle sole istanze della classe *small*, pur restando valido il meccanismo che sarà applicato anche alle classi *medium* e *large* negli ulteriori studi previsti all'interno del *Metaheuristics Network*. Relativamente alla classe *small*, sono stati eseguiti gli stessi esperimenti del mese di Aprile 2002, in modo da poter mettere a confronto le nostre implementazioni (inclusa la versione di Ant Colony System con i parametri scelti grazie a F-Race) con i precedenti algoritmi. Facciamo presente il fatto che, per garantire le stesse condizioni sperimentali, durante la fase di sviluppo e di configurazione, abbiamo ignorato le istanze usate per gli esperimenti finali. In tal modo, al momento del confronto, i nuovi algoritmi si trovavano nelle stesse condizioni di quelli vecchi. Gli algoritmi sono identificati all'interno dei grafici nel seguente modo:

- **acs1** identifica l'algoritmo Ant Colony System, implementato dall'autore della tesi, in cui i valori dei parametri sono stati scelti dopo una serie di esperimenti manuali (si veda la Tabella 2.1);

- **acs2** identifica l'algoritmo Ant Colony System, implementato dall'autore della tesi, in cui i valori dei parametri sono stati selezionati con il metodo automatico F-Race (si veda la Tabella 3.3);
- **ga** identifica l'Algoritmo Genetico, implementato da Olivia Rossi-Doria presso il nodo ECRG;
- **ga1** identifica l'Algoritmo Memetico, implementato dall'autore della tesi, in cui i valori dei parametri sono stati selezionati con il metodo automatico F-Race (si veda la Tabella 3.6);
- **ils** identifica l'algoritmo Ricerca Locale Iterata, implementato da Luis Paquete presso il nodo INTELLEKTIK;
- **ils1** identifica l'algoritmo Ricerca Locale Iterata, implementato dall'autore della tesi;
- **rrls** identifica l'algoritmo *Random Restart Local Search*, usato come *benchmark*.
- **sa** identifica l'algoritmo Simulated Annealing, implementato da Marco Chiarandini presso il nodo INTELLEKTIK;
- **ts** identifica l'algoritmo Tabu Search, implementato da Monaldo Mastrolilli presso il nodo IDSIA.

Un criterio visivo per l'identificazione degli algoritmi implementati dall'autore della tesi è la presenza di un numero nella sigla identificativa dell'algoritmo. Di seguito riportiamo il grafico della sola prima istanza (`small01.tim`[9]) in

[9] Pur comparendo nei grafici i nomi `easy01.tim`, `easy02.tim`, `easy03.tim`, `easy04.tim` e `easy05.tim`, si tratta delle stesse istanze usate nei confronti interni al *Metaheuristics Network*.

quanto i risultati ottenuti dagli algoritmi all'interno della classe *small* sono similari. I risultati completi di tutte le istanze della classe *small* sono riportati in Appendice A.2.

4.3.1 Risultati istanze *small*

Ognuna delle 5 istanze *small* del problema su cui abbiamo eseguito gli esperimenti ha fornito risultati per i *p-value* molto piccoli, minori del valore 0.05 che ci permette di rifiutare l'ipotesi nulla che le distribuzioni delle soluzioni prodotte dalle coppie di algoritmi siano uguali, con qualche eccezione per le coppie <**ils**, **ils1**>, <**acs2**, **sa**>, <**ga**, **ts**>. Uno sguardo ai grafici mostra come gli algoritmi **acs1, acs2, ga1, ils, ils1** e **sa** riescano a produrre, con elevata frequenza, soluzioni ottime o vicine all'ottimo.

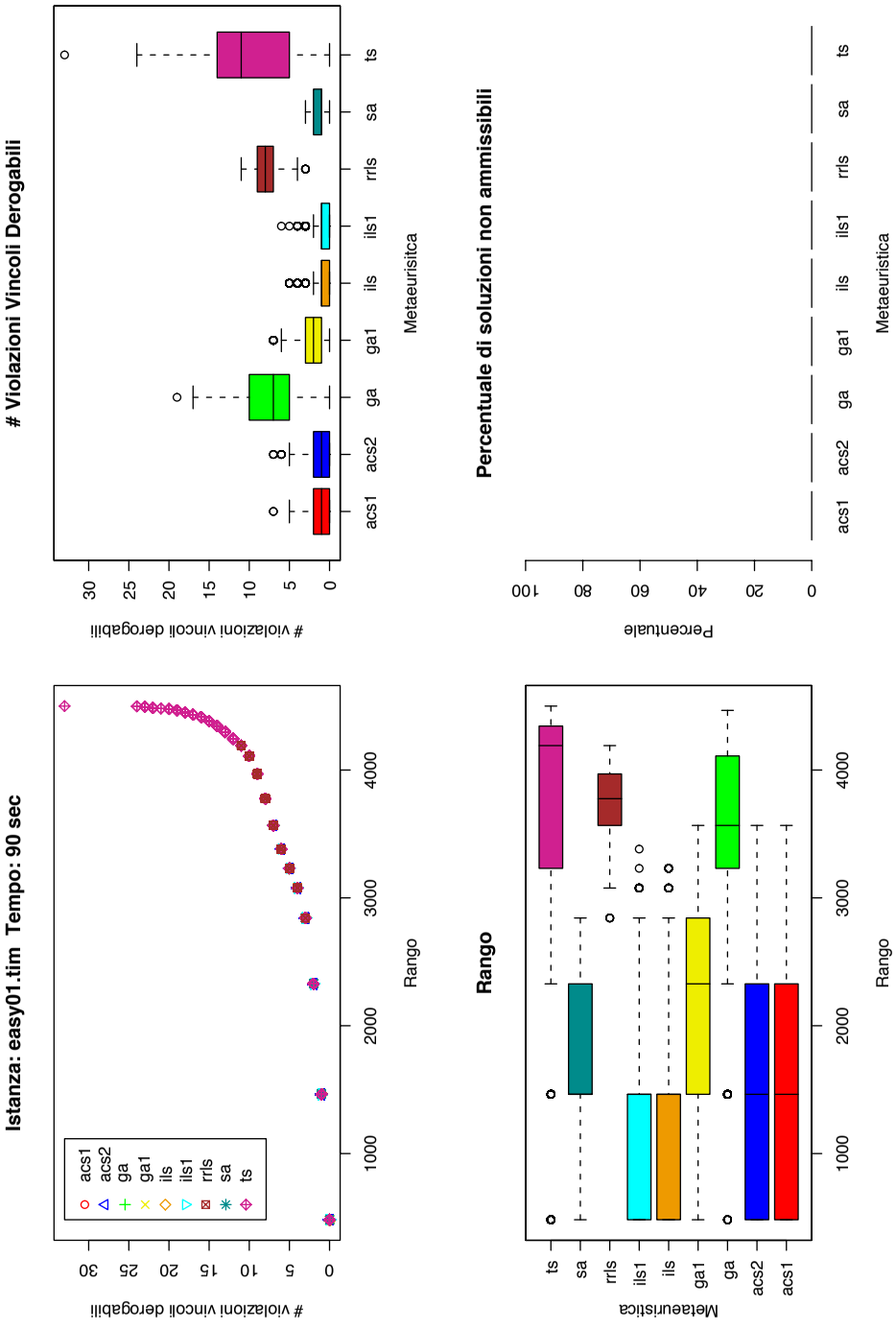

Figura 4.4: Risultati istanza `small01.tim`

Tabella 4.4: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto in tabella) non ci permettono di rifiutare l'ipotesi nulla.

	acs1	acs2	ga	ga1
acs2	0.0049			
ga	$< 2^{-16}$	$< 2^{-16}$		
ga1	$< 2^{-16}$	3.7^{-12}	$< 2^{-16}$	
ils	7.6^{-5}	1.5^{-11}	$< 2^{-16}$	$< 2^{-16}$
ils1	3^{-7}	4.4^{-15}	$< 2^{-16}$	$< 2^{-16}$
rrls	$< 2^{-16}$	$< 2^{-16}$	0.0032	$< 2^{-16}$
sa	6.9^{-9}	**0.087**	$< 2^{-16}$	5.9^{-11}
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

Tabella 4.5: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ils	ils1	rrls	sa
ils1	**0.2797**			
rrls	$< 2^{-16}$	$< 2^{-16}$		
sa	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

Conclusioni

L'obiettivo principale di questa tesi è la comparazione dei risultati prodotti da diverse metaeuristiche applicate alla risoluzione di una riduzione del problema della Compilazione degli Orari dei Corsi Universitari, la cui formulazione è stata fornita nel Paragrafo 1.3.2.

In molti dei lavori di ricerca presenti in letteratura ci si accorge di come la comparazione tra metaeuristiche non sia eseguita in modo equo. Nel Paragrafo 2.3 abbiamo analizzato i fattori che, se non tenuti nella dovuta considerazione, possono far commettere errori nei paragoni.

Per evitare di incappare in quegli errori, all'interno del *Metaheuristics Network*, la rete di addestramento alla ricerca nella quale abbiamo svolto il lavoro di tesi, presentata nell'Introduzione, è stata definita una metodologia per la progettazione e l'esecuzione degli esperimenti in modo rigoroso, in modo tale che fossero garantite le stesse condizioni sperimentali per tutte le implementazioni.

In una prima fase abbiamo confrontato i cinque algoritmi sviluppati all'interno del *Metaheuristics Network* su una serie di istanze del problema appartenenti a tre classi di difficoltà crescente. Questi confronti sono stati illustrati nel Paragrafo 4.2. In una seconda fase abbiamo esteso tali confronti alle nostre implementazioni. Questi confronti sono stati illustrati nel Paragrafo 4.3. Alla luce dei risultati ottenuti in tutti i confronti possiamo trarre

le seguenti conclusioni generali:

1. Rispettivamente ai risultati aggregati delle metaeuristiche, la difficoltà delle istanze del problema varia, a volte anche in modo significativo, tra istanze di classi diverse, e, seppur in modo minore, tra istanze appartenenti alla stessa classe (ossia, istanze generate con gli stessi parametri del generatore, ad eccezione del seme casuale. Questo comportamente era atteso, e riflette i dati del mondo reale, in cui alcuni problemi specifici (ad esempio, una particolare scelta dei corsi da seguire da parte di un gruppo di studenti) possono rendere la compilazione dell'orario molto più difficile in quel particolare anno.

2. Le prestazioni assolute di una singola metaeuristica variano, spesso in modo significativo, tra le istanze del problema. Queste variazioni sono più marcate tra istanze di categorie diverse, anche se, in una certa misura, è possibile osservarle anche fra istanze appartenenti alla stessa classe.

3. Le prestazioni relative di ogni coppia di metaeuristiche varia, spesso in modo significativo, tra le istanze del problema. Queste variazioni sono più marcate tra istanze di categorie diverse, anche se, in una certa misura, è possibile osservarle anche fra istanze appartenenti alla stessa classe.

4. Le prestazioni di una metaeuristica, rispetto al soddisfacimento dei vincoli inderogabili e dei vincoli derogabili, possono essere molto diverse.

I risultati mostrano che nessuna metaeuristica, tra quelle implementate, è migliore su tutte le istanze del problema considerato. Inoltre, anche quando le istanze sono molto simili, dal punto di vista del generatore che le produce, non

è possibile prevedere quale sarà la metaeuristica migliore, anche se emergono alcune tendenze quando ci concentriamo su istanze di una particolare classe. I risultati evidenziano quanto sia difficile trovare la migliore metaeuristica, anche per classi molto ristrette del problema.

Queste conclusioni ci portano a credere che sia molto difficile progettare una metaeuristica capace di risolvere istanze generali, anche limitatamente alla classe dei problemi forniti dal generatore. Comunque, i risultati ottenuti suggeriscono che una soluzione promettente possa essere un algoritmo ibrido composto di almeno due fasi, la prima che si occupi di raggiungere l'ammissibilità della soluzione, la seconda che si occupi di minimizzare il numero di violazioni dei vincoli derogabili.

In aggiunta, abbiamo avuto la conferma che la conoscenza di alcuni aspetti dell'istanza non garantisce che si sappia qualcosa della struttura dello spazio di ricerca. Questo suggerisce l'importanza, nel futuro, di tentare di misurare direttamente le caratteristiche dello spazio di ricerca; lo scopo è quello di riuscire a determinare la migliore configurazione dei parametri per ciascun algoritmo, basandosi sulle misure di queste caratteristiche.

Questo lavoro è in corso d'opera. Allo scopo di approfondire la comprensione di questi problemi, il *Metaheuristics Network* ha organizzato una competizione internazionale[10] basata sullo stesso generatore di istanze usato per questa tesi. Un'analisi preliminare sulle caratteristiche dello spazio di ricerca è stata presentata da Chiarandini e Stützle (2002).

[10] `http://www.idsia.ch/Files/ttcomp2002`

Sviluppi futuri

La disponibilità di nuove infrastrutture informatiche (un nuovo Beowulf) di cui si sta dotando **IRIDIA**, ci permetterà di applicare F-Race, la metodologia automatica per la configurazione delle metaeuristiche, anche sulle istanze di classe *medium* e *large*, sia per gli algoritmi implementati all'interno del *Metaheuristics Network*, che per le nostre implementazioni. Inoltre abbiamo in progetto di adattare la metodologia F-Race per l'uso come metodo di selezione della miglior metaeuristica.

Appendice A

Grafici dei confronti

In questa appendice sono riportati i grafici con le comparazioni dei risultati ottenuti dagli algoritmi sulle istanze usate per gli esperimenti del mese di Aprile 2002, descritti nel Paragrafo 4.2.

A.1 Confronti interni al *Metaheuristics Network*

Gli algoritmi implementati nella prima fase dell'attività di ricerca sul problema della Compilazione degli Orari dei Corsi Universitari, descritto nel Paragrafo 1.3.2, sono stati i seguenti:

- **IRIDIA:** l'autore della tesi ha implementato l'Algoritmo Ant Colony System, descritto nel Paragrafo 2.4.2. Nei grafici è identificato dalla sigla ACO;
- **INTELLEKTIK:** Luis Paquete[1] ha implementato un algoritmo di Ricerca Locale Iterata, di cui abbiamo parlato nel Paragrafo 2.2.1. Nei

[1] e-mail: lpaquete@intellektik.informatik.tu-darmstadt.de

grafici è identificato dalla sigla ILS;

- **INTELLEKTIK:** Marco Chiarandini[2] ha implementato un algoritmo di Simulated Annealing, di cui abbiamo parlato nel Paragrafo 2.2.2. Nei grafici è identificato dalla sigla SA;

- **ECRG:** Olivia Rossi-Doria[3] ha implementato un algoritmo di Computazione Evolutiva, di cui abbiamo parlato nel Paragrafo 2.2.4. Nei grafici è identificato dalla sigla GA;

- **IDSIA:** Monaldo Mastrolilli[4] ha implementato un algoritmo di Tabu Search, di cui abbiamo parlato nel Paragrafo 2.2.3. Nei grafici è identificato dalla sigla TS.

Di seguito riportiamo i grafici con le comparazioni dei risultati, ottenuti dagli algoritmi sopracitati, su tutte le istanze generate per gli esperimenti descritti nel Paragrafo 4.2.

[2]e-mail: machud@intellektik.informatik.tu-darmstadt.de
[3]e-mail: o.rossi-doria@napier.ac.uk
[4]e-mail: monaldo@idsia.ch

Tabella A.1: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	$< 2^{-16}$			
ILS	1.5^{-5}	$< 2^{-16}$		
SA	2^{-9}	$< 2^{-16}$	$< 2^{-16}$	
TS	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

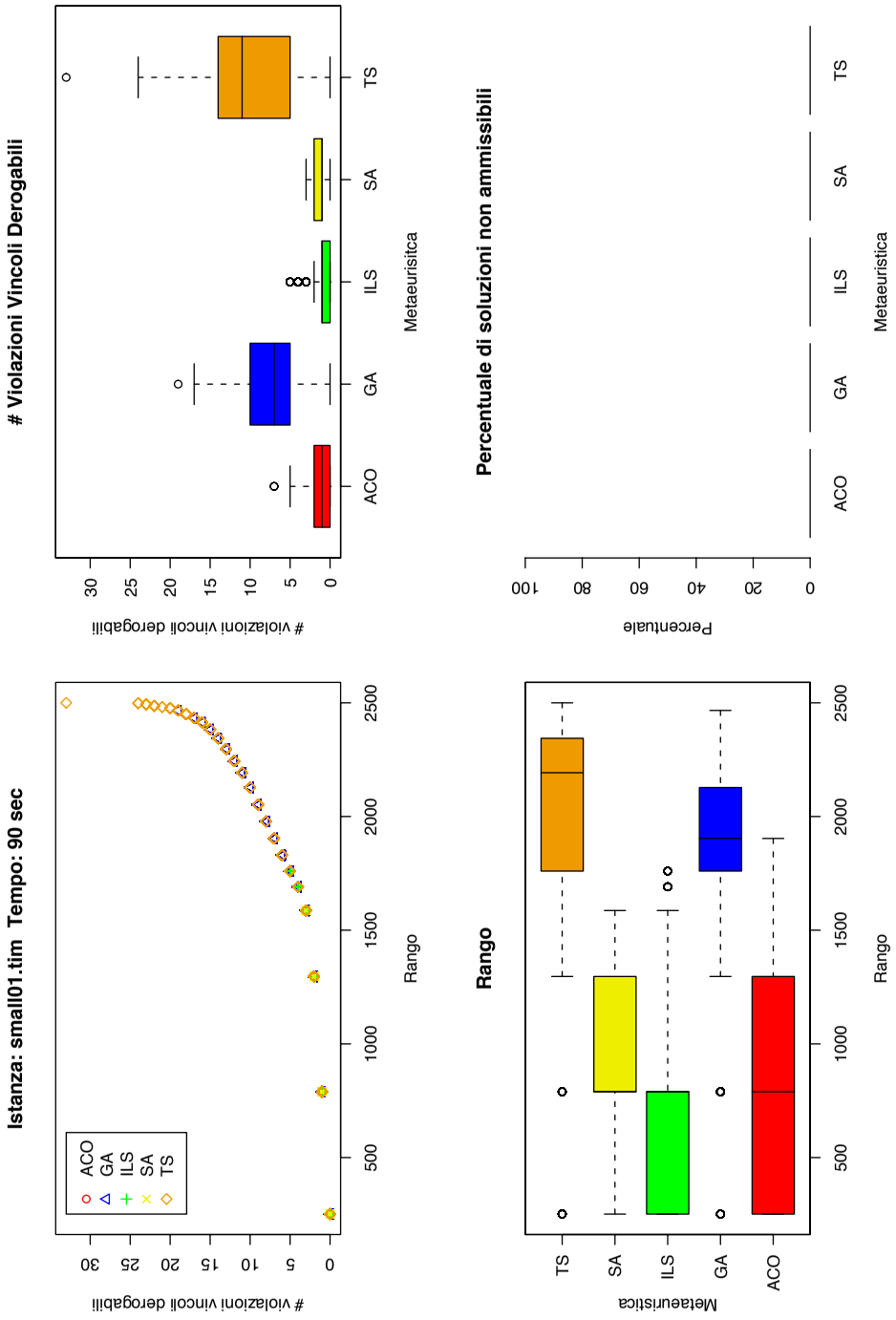

Figura A.1: Risultati istanza `small01.tim`

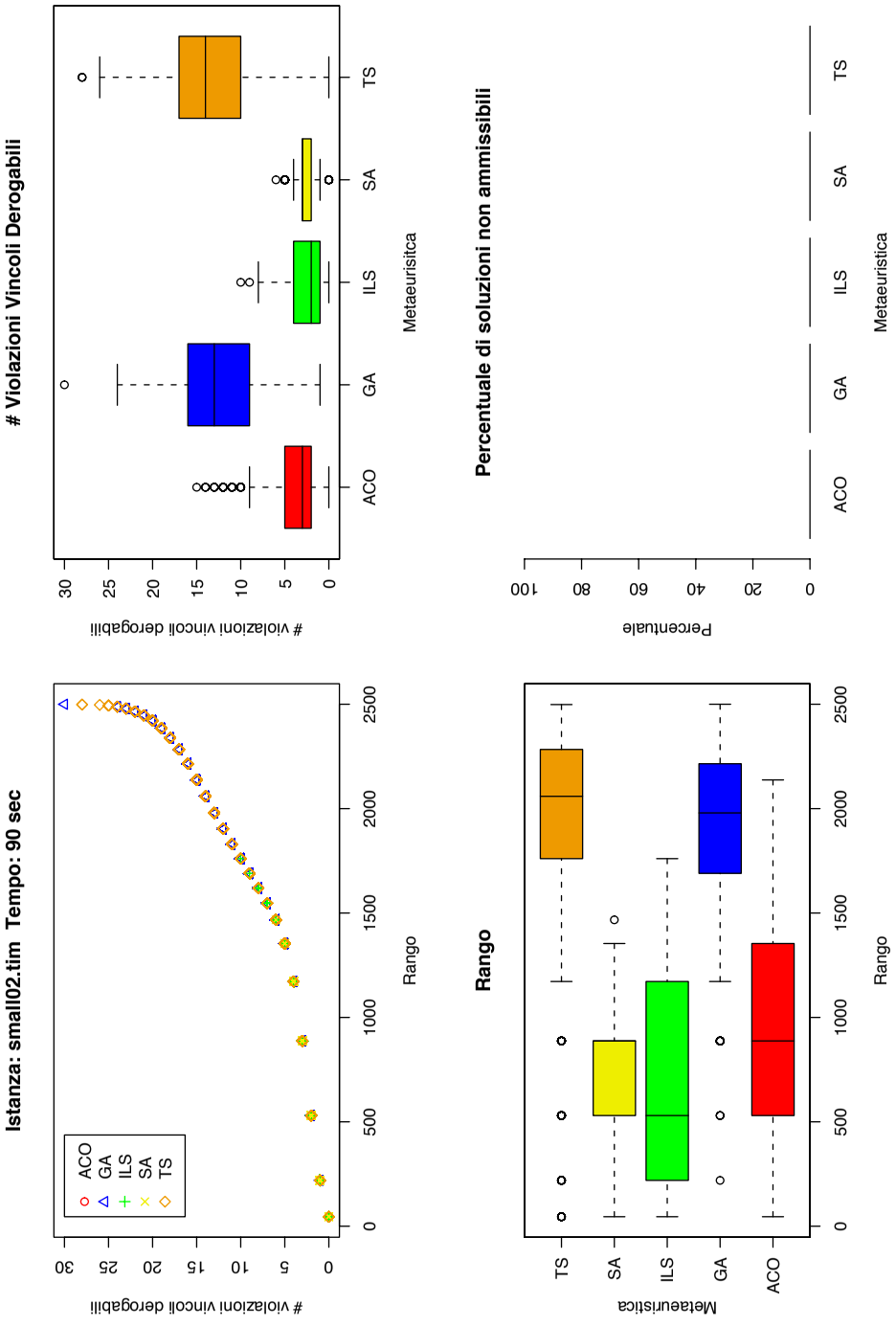

Figura A.2: Risultati istanza `small02.tim`

Tabella A.2: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small02.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	$< 2^{-16}$			
ILS	9.7^{-11}	$< 2^{-16}$		
SA	1.9^{-6}	$< 2^{-16}$	0.0039	
TS	$< 2^{-16}$	0.0372	$< 2^{-16}$	$< 2^{-16}$

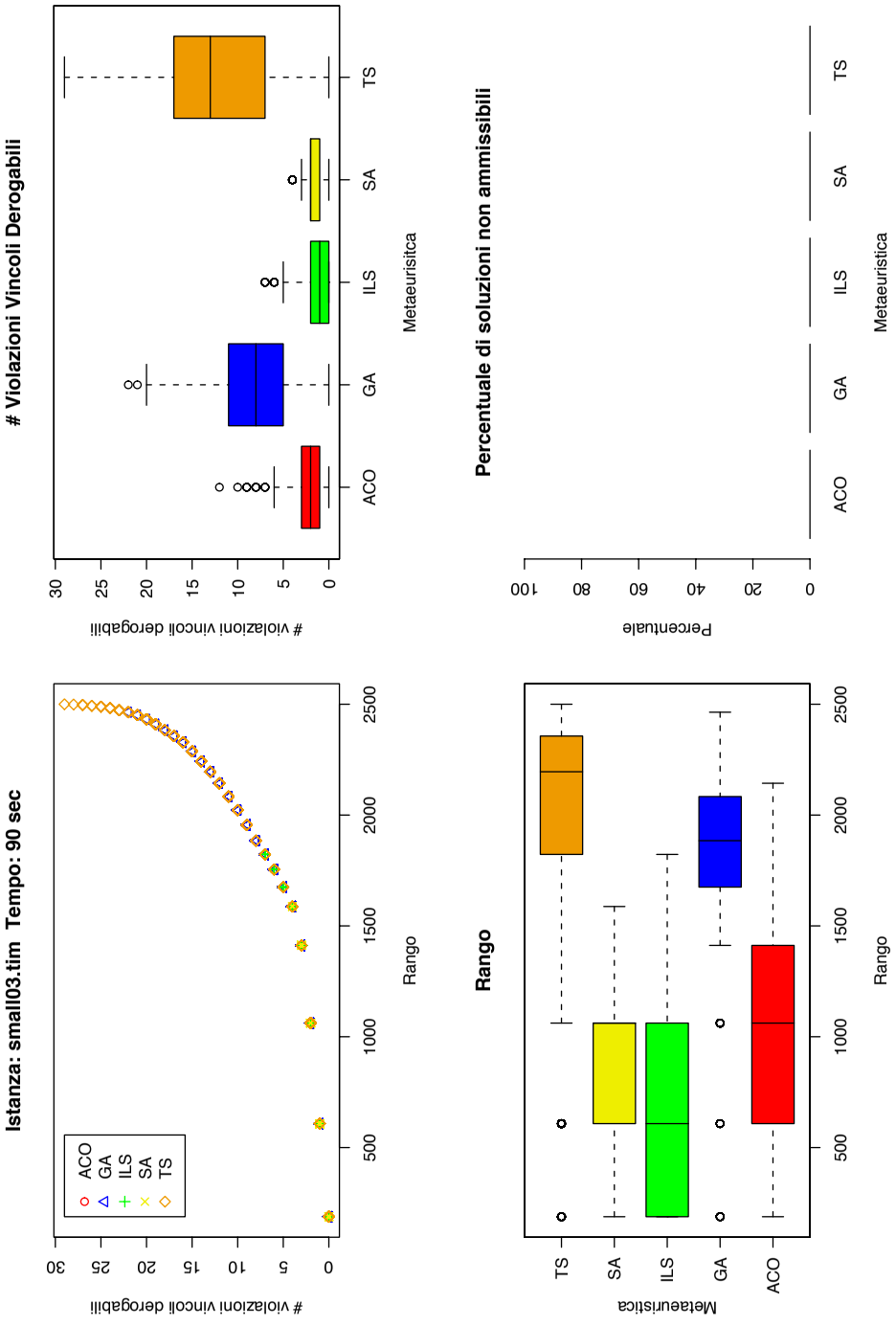

Figura A.3: Risultati istanza `small03.tim`

Tabella A.3: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small03.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	$< 2^{-16}$			
ILS	3.8^{-10}	$< 2^{-16}$		
SA	**0.43**	$< 2^{-16}$	$< 2^{-16}$	
TS	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

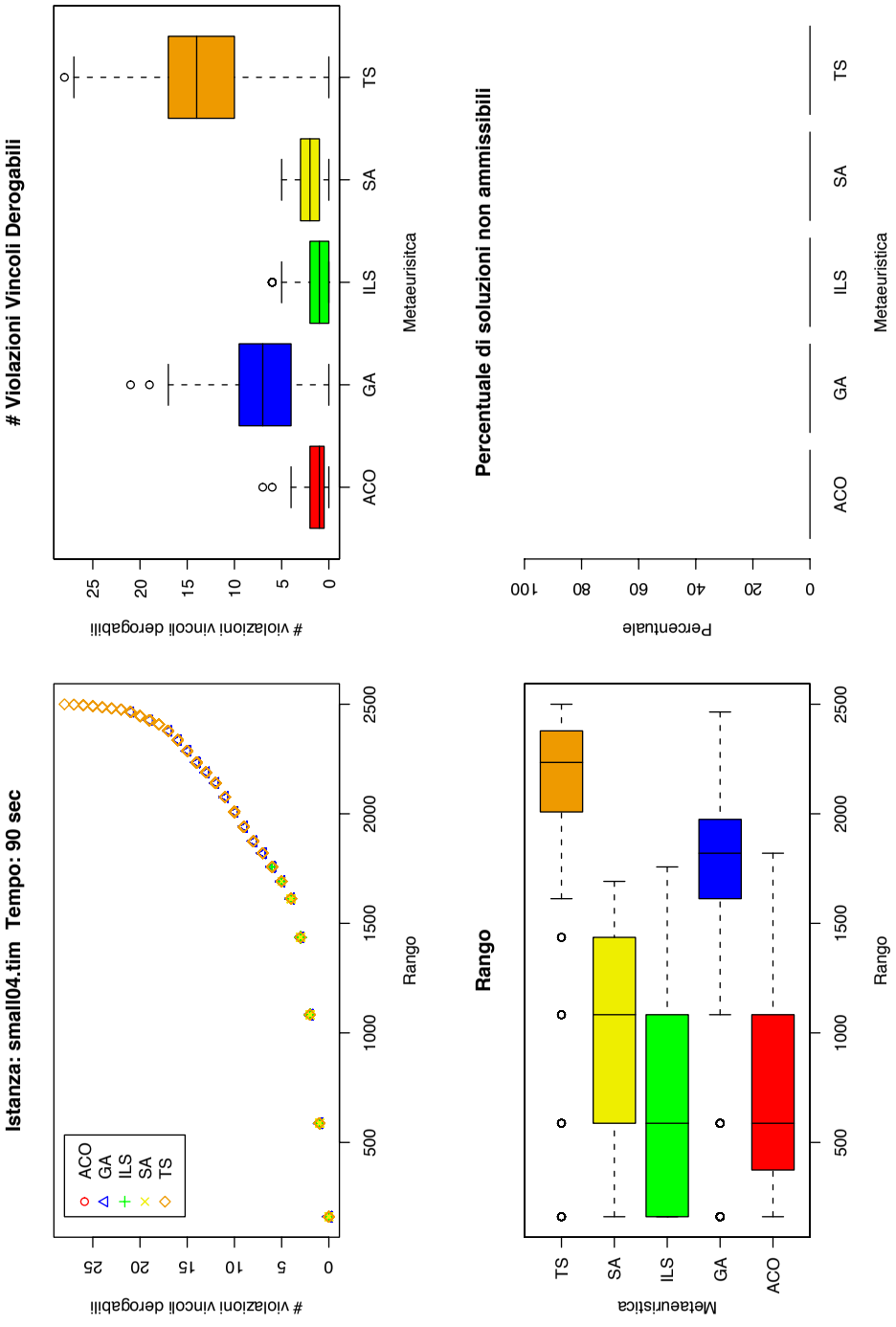

Figura A.4: Risultati istanza `small04.tim`

Tabella A.4: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small04.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	$< 2^{-16}$			
ILS	**0.11**	$< 2^{-16}$		
SA	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	
TS	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

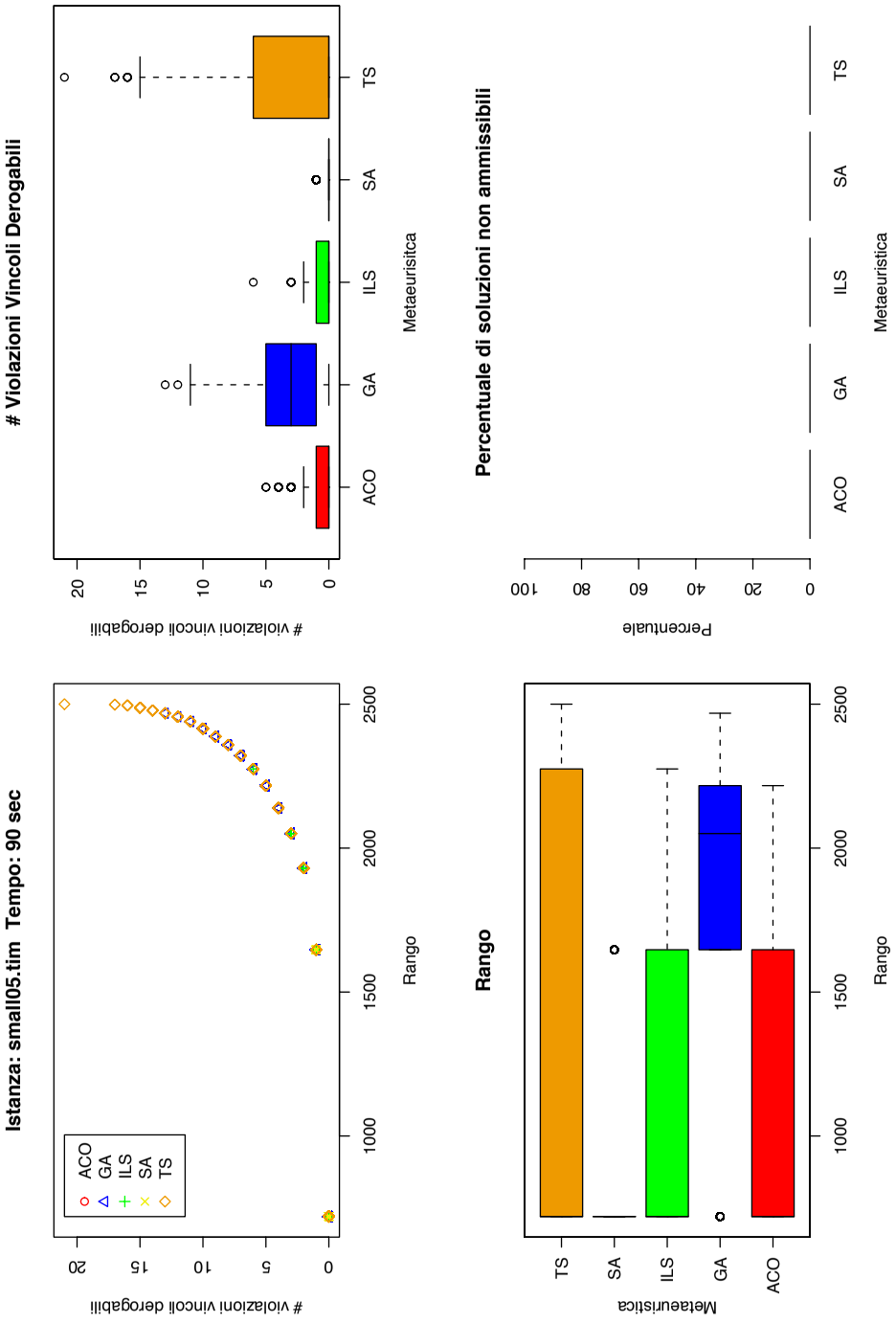

Figura A.5: Risultati istanza `small05.tim`

Tabella A.5: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small05.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	$< 2^{-16}$			
ILS	1.9^{-13}	$< 2^{-16}$		
SA	$< 2^{-16}$	$< 2^{-16}$	7^{-14}	
TS	5.2^{-6}	3.3^{-10}	$< 2^{-16}$	$< 2^{-16}$

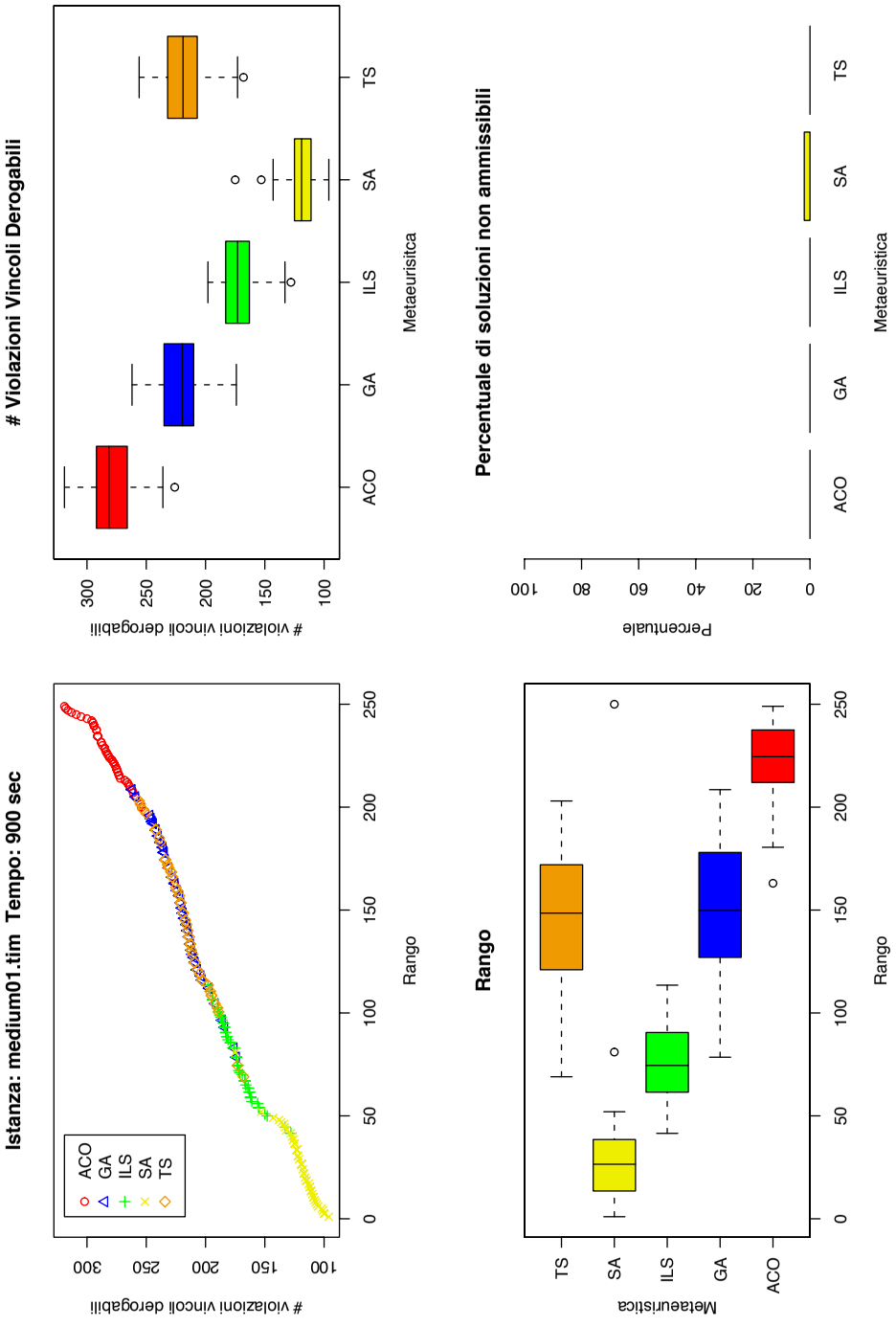

Figura A.6: Risultati istanza `medium01.tim`

Tabella A.6: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `medium01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	7.8^{-16}			
ILS	$< 2^{-16}$	1.4^{-15}		
SA	9.3^{-16}	9.3^{-16}	3.2^{-15}	
TS	5.8^{-16}	**0.54**	2^{-15}	1.1^{-15}

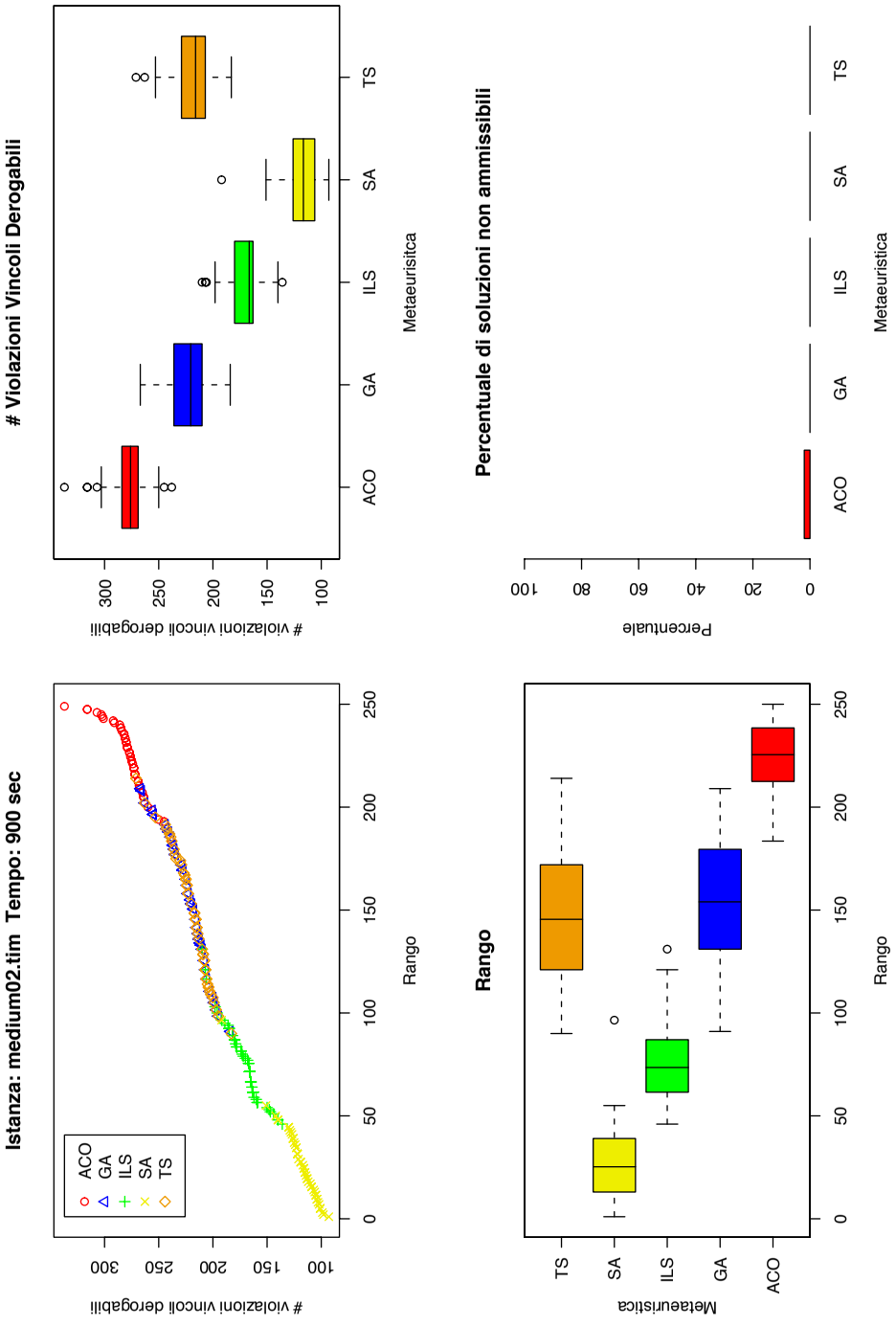

Figura A.7: Risultati istanza `medium02.tim`

Tabella A.7: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `medium02.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	3.5^{-16}			
ILS	$< 2^{-16}$	3.5^{-16}		
SA	$< 2^{-16}$	$< 2^{-16}$	5.5^{-16}	
TS	$< 2^{-16}$	**0.31**	5.5^{-16}	$< 2^{-16}$

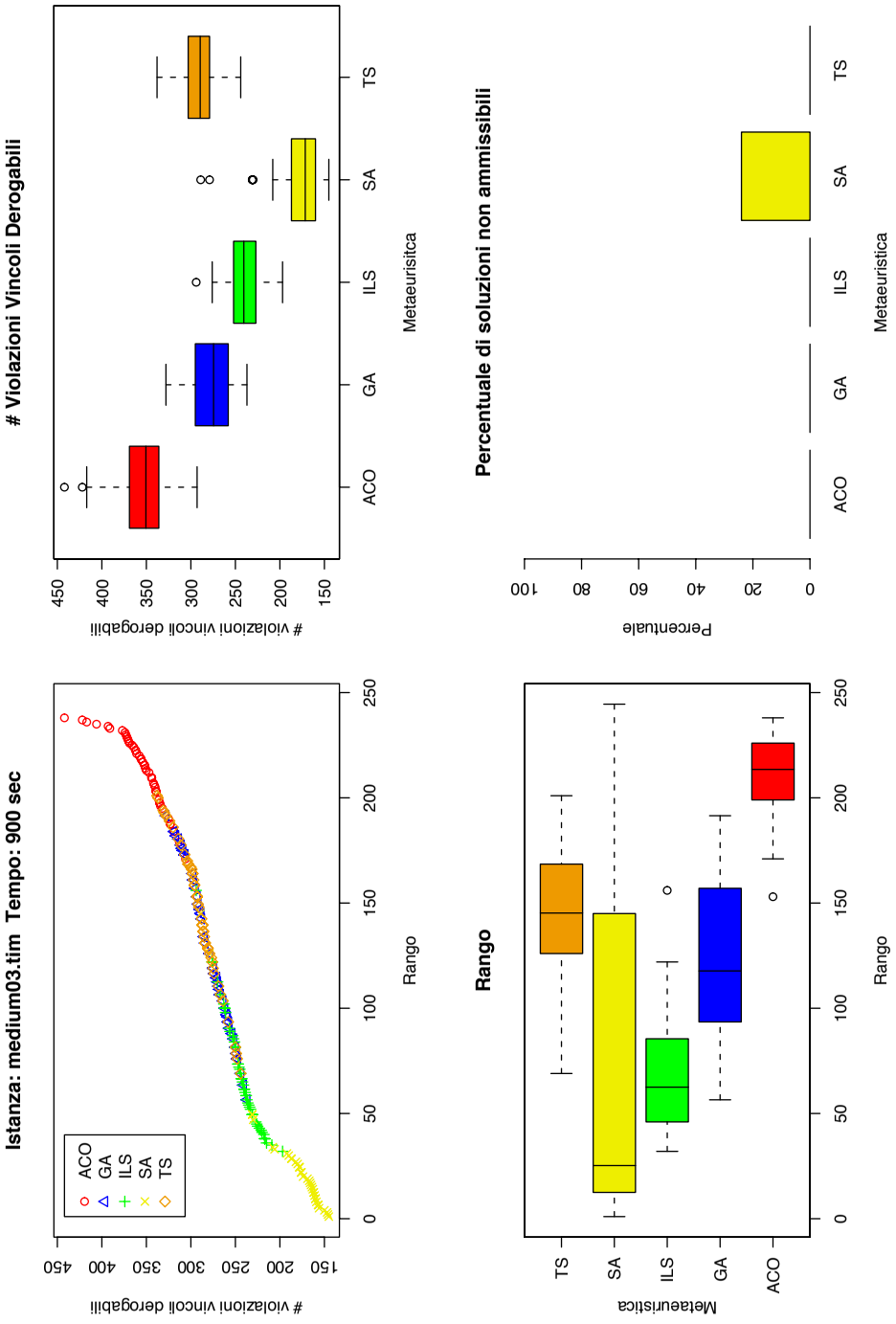

Figura A.8: Risultati istanza `medium03.tim`

Tabella A.8: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `medium03.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	4.3^{-16}			
ILS	$< 2^{-16}$	1^{-10}		
SA	3.7^{-5}	0.00014	0.00090	
TS	6.1^{-15}	0.00724	4^{-14}	9.6^{-5}

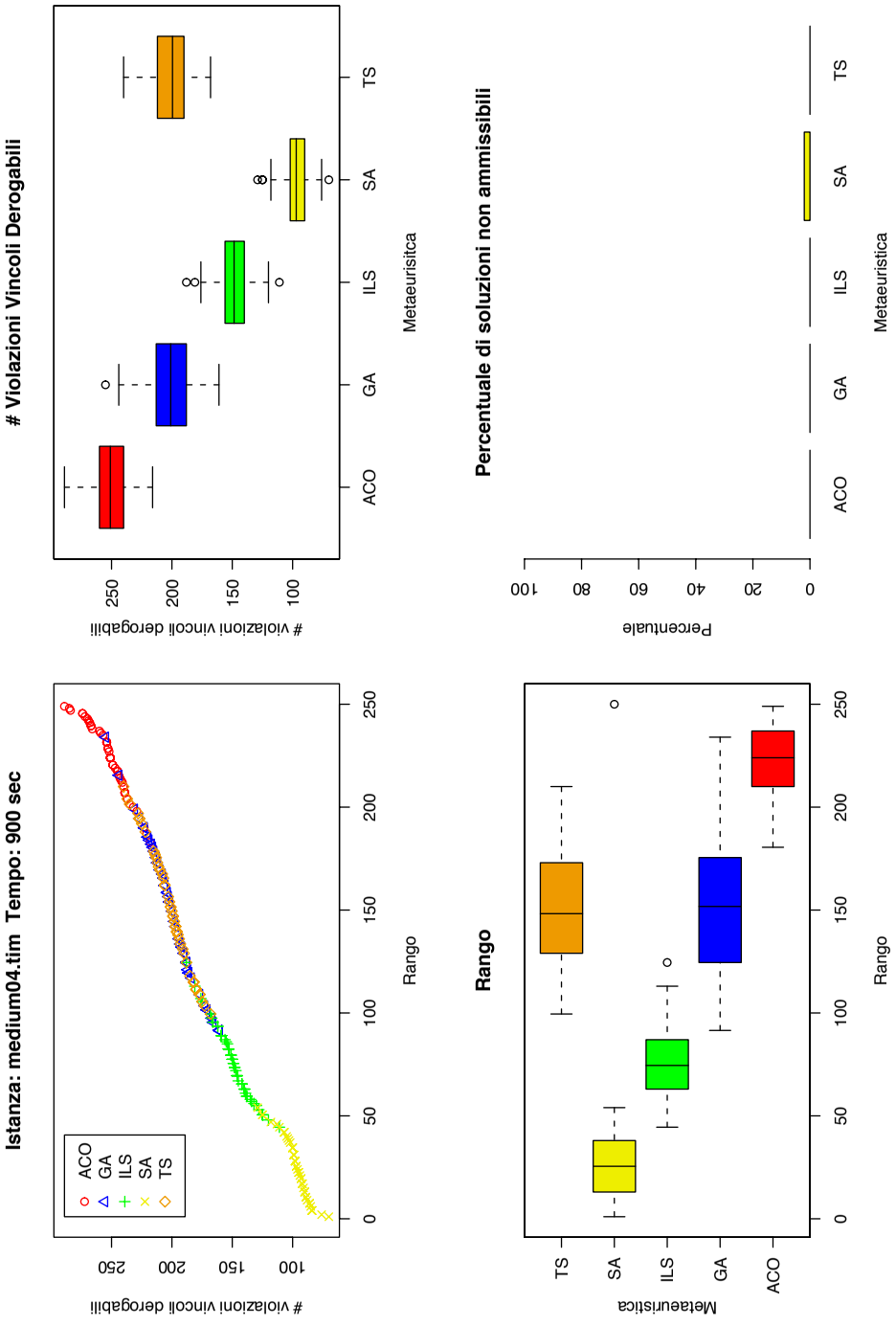

Figura A.9: Risultati istanza `medium04.tim`

Tabella A.9: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `medium04.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	1.2^{-15}			
ILS	$< 2^{-16}$	1.1^{-15}		
SA	1.1^{-15}	1.1^{-15}	1.1^{-15}	
TS	1.1^{-15}	**0.71**	$< 2^{-16}$	1.1^{-15}

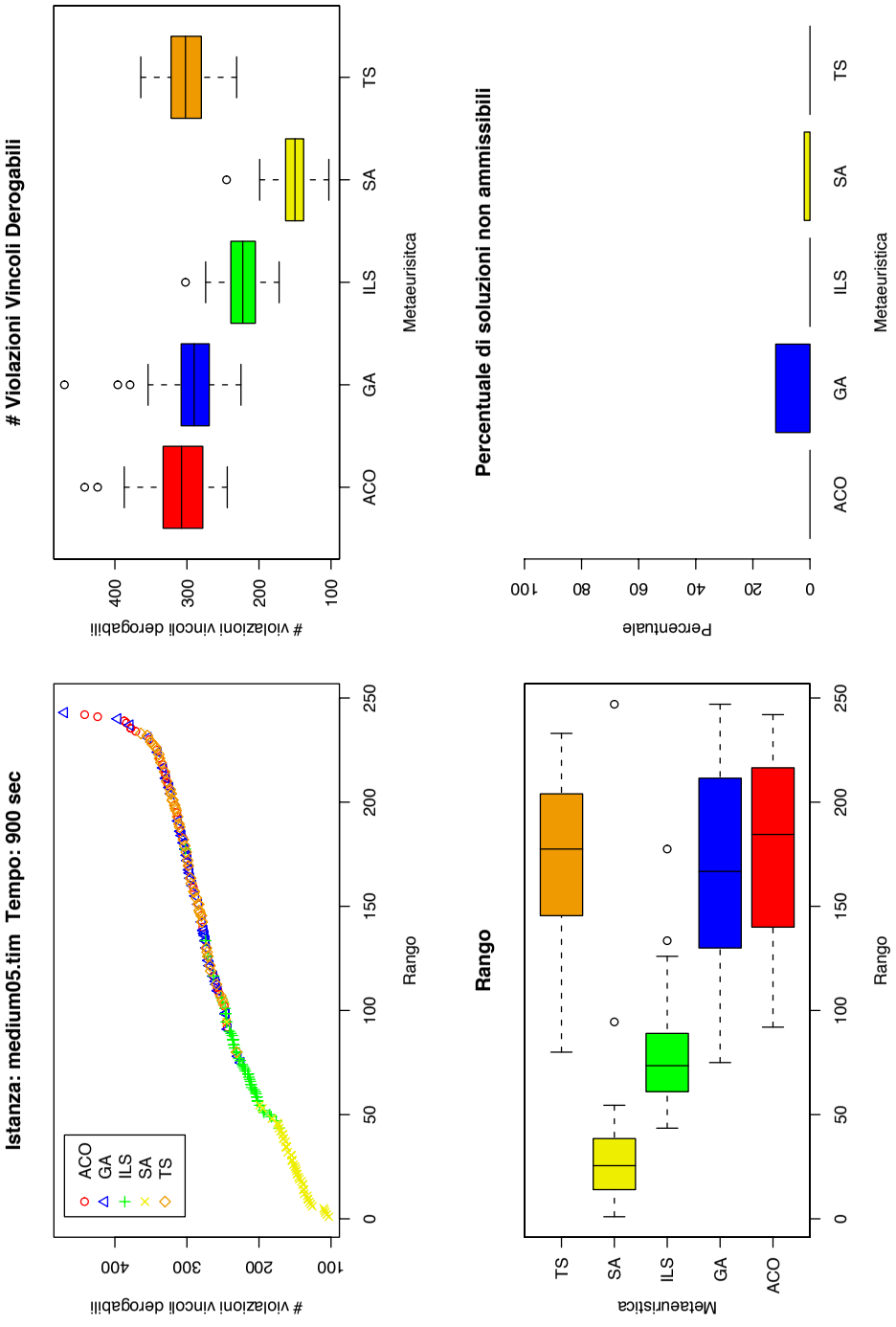

Figura A.10: Risultati istanza `medium05.tim`

Tabella A.10: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `medium05.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	**1**			
ILS	2.5^{-15}	2.3^{-14}		
SA	1.3^{-15}	1.3^{-15}	1.7^{-14}	
TS	**1**	**1**	2.7^{-15}	1.8^{-15}

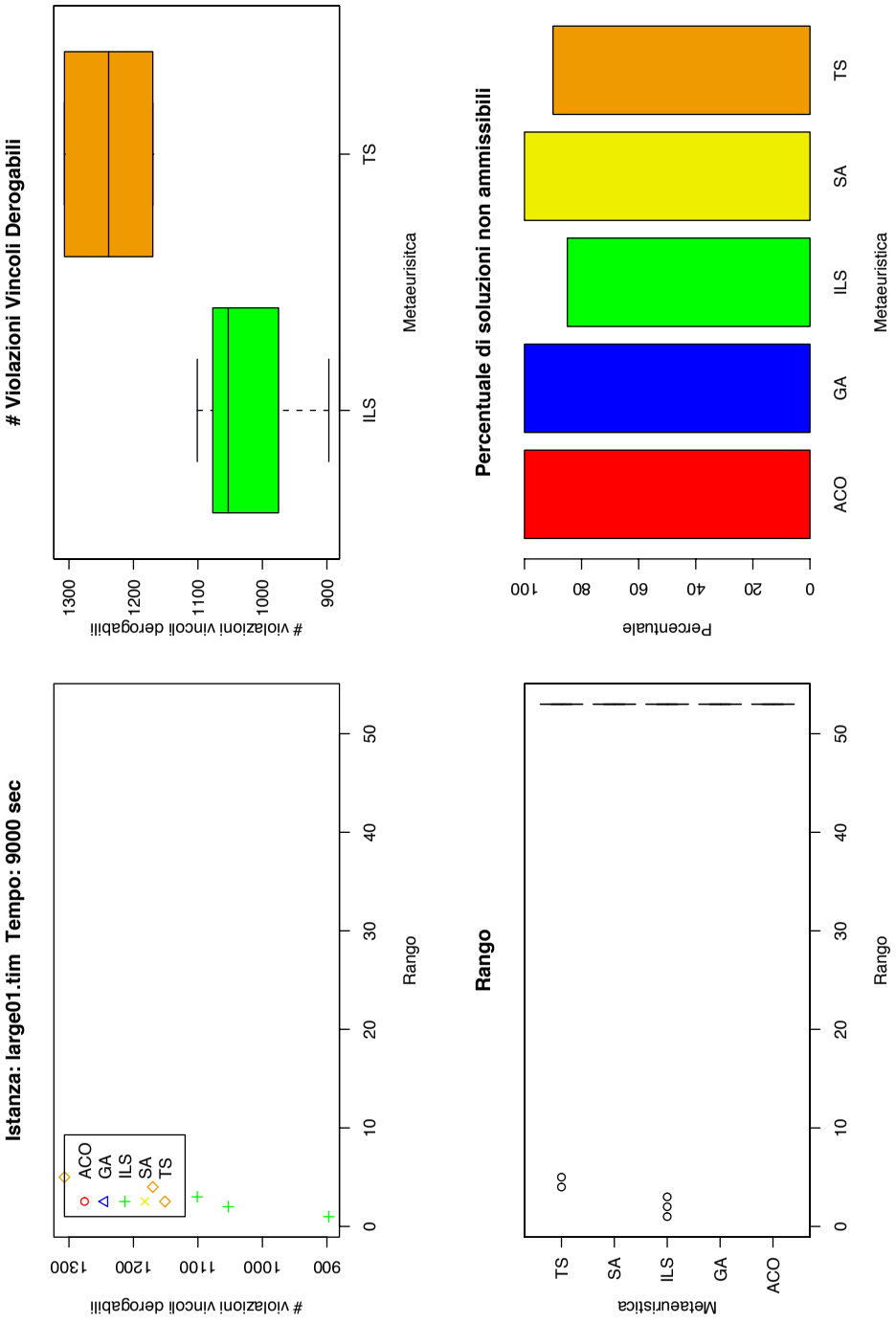

Figura A.11: Risultati istanza `large01.tim`

Tabella A.11: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `large01.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	-			
ILS	**0.8**	**0.8**		
SA	-	-	**0.8**	
TS	**1**	**1**	**1**	**1**

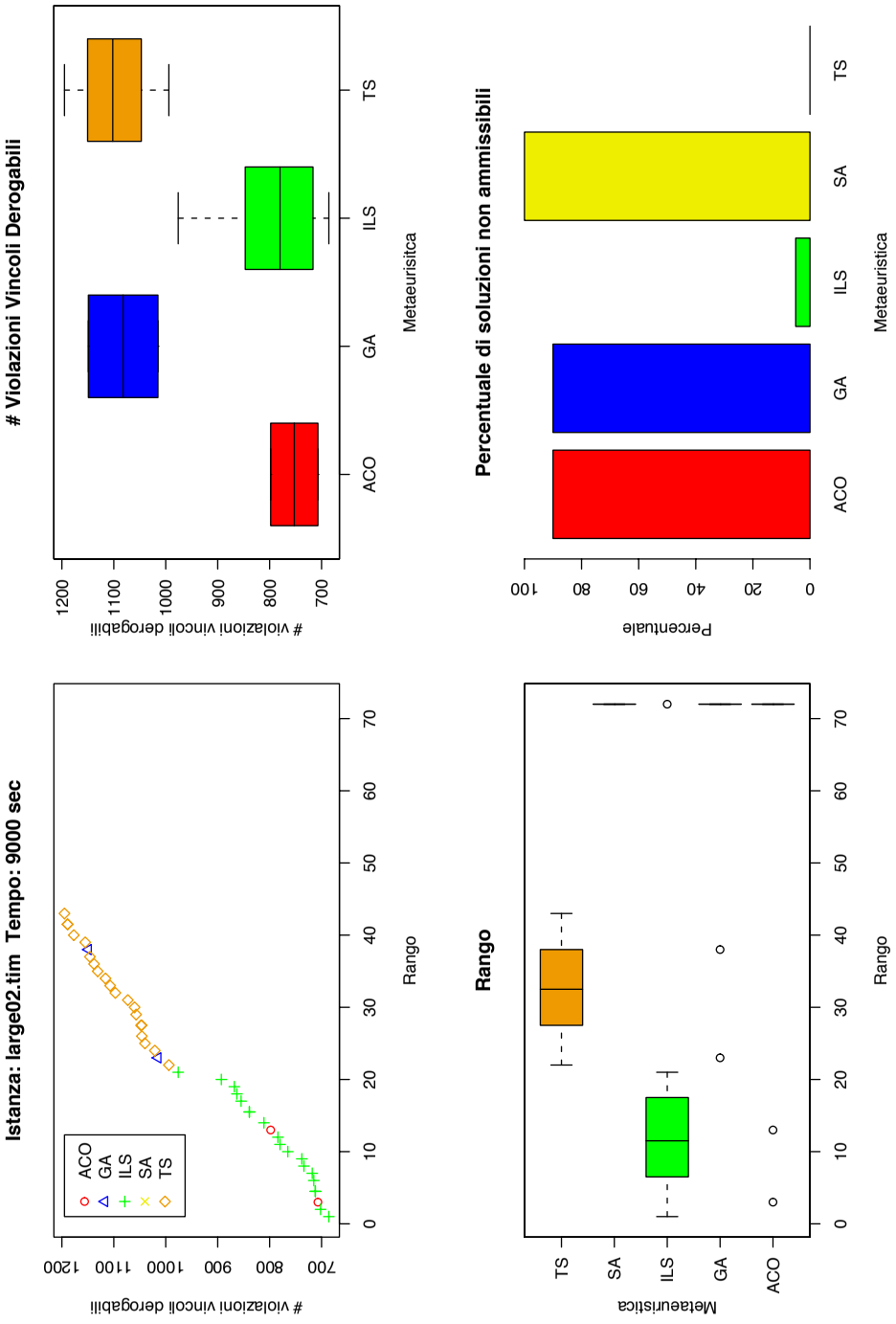

Figura A.12: Risultati istanza `large02.tim`

Tabella A.12: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `large02.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ACO	GA	ILS	SA
GA	**0.94**			
ILS	1.4^{-5}	5.4^{-7}		
SA	**0.49**	**0.49**	2.7^{-7}	
TS	2.4^{-5}	4.5^{-6}	7.2^{-6}	8^{-8}

A.2 Confronti estesi alle nostre implementazioni

Dopo il mese di Aprile 2002, l'autore della tesi ha implementato due ulteriori algoritmi: l'Algoritmo Memetico, descritto nel Paragrafo 2.4.4, e un algoritmo di Ricerca Locale Iterata, descritto nel Paragrafo 2.4.3. Inoltre si è proceduto alla ricerca della migliore configurazione dei parametri per gli algoritmi Ant Colony System e Algoritmo Memetico usando il metodo automatico F-Race, come descritto nel Paragrafo 3.3. Sono stati eseguiti, quindi, gli stessi esperimenti del mese di Aprile 2002, per poter mettere a confronto le nuove implementazioni, e la versione di Ant Colony System con la nuova configurazione dei parametri. Gli algoritmi sono identificati all'interno dei grafici nel seguente modo:

- **acs1** identifica l'algoritmo Ant Colony System, implementato dall'autore della tesi, in cui i valori dei parametri sono stati scelti dopo una serie di esperimenti manuali (si veda la Tabella 2.1);
- **acs2** identifica l'algoritmo Ant Colony System, implementato dall'autore della tesi, in cui i valori dei parametri sono stati selezionati con il metodo automatico F-Race (si veda la Tabella 3.3);
- **ga** identifica l'Algoritmo Genetico, implementato da Olivia Rossi-Doria presso il nodo ECRG;
- **ga1** identifica l'Algoritmo Memetico, implementato dall'autore della tesi, in cui i valori dei parametri sono stati selezionati con il metodo automatico F-Race (si veda la Tabella 3.6);

- **ils** identifica l'algoritmo Ricerca Locale Iterata, implementato da Luis Paquete presso il nodo INTELLEKTIK;
- **ils1** identifica l'algoritmo Ricerca Locale Iterata, implementato dall'autore della tesi;
- **rrls** identifica l'algoritmo *Random Restart Local Search*, usato come *benchmark*.
- **sa** identifica l'algoritmo Simulated Annealing, implementato da Marco Chiarandini presso il nodo INTELLEKTIK;
- **ts** identifica l'algoritmo Tabu Search, implementato da Monaldo Mastrolilli presso il nodo IDSIA.

Un criterio visivo per l'identificazione degli algoritmi implementati dall'autore della tesi è la presenza di un numero nella sigla identificativa dell'algoritmo.

Tabella A.13: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small01.tim`.

	acs1	acs2	ga	ga1
acs2	0.0049			
ga	$< 2^{-16}$	$< 2^{-16}$		
ga1	$< 2^{-16}$	3.7^{-12}	$< 2^{-16}$	
ils	7.6^{-5}	1.5^{-11}	$< 2^{-16}$	$< 2^{-16}$
ils1	3^{-7}	4.4^{-15}	$< 2^{-16}$	$< 2^{-16}$
rrls	$< 2^{-16}$	$< 2^{-16}$	0.0032	$< 2^{-16}$
sa	6.9^{-9}	**0.087**	$< 2^{-16}$	5.9^{-11}
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

Tabella A.14: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small01.tim`.

	ils	ils1	rrls	sa
ils1	**0.2797**			
rrls	$< 2^{-16}$	$< 2^{-16}$		
sa	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

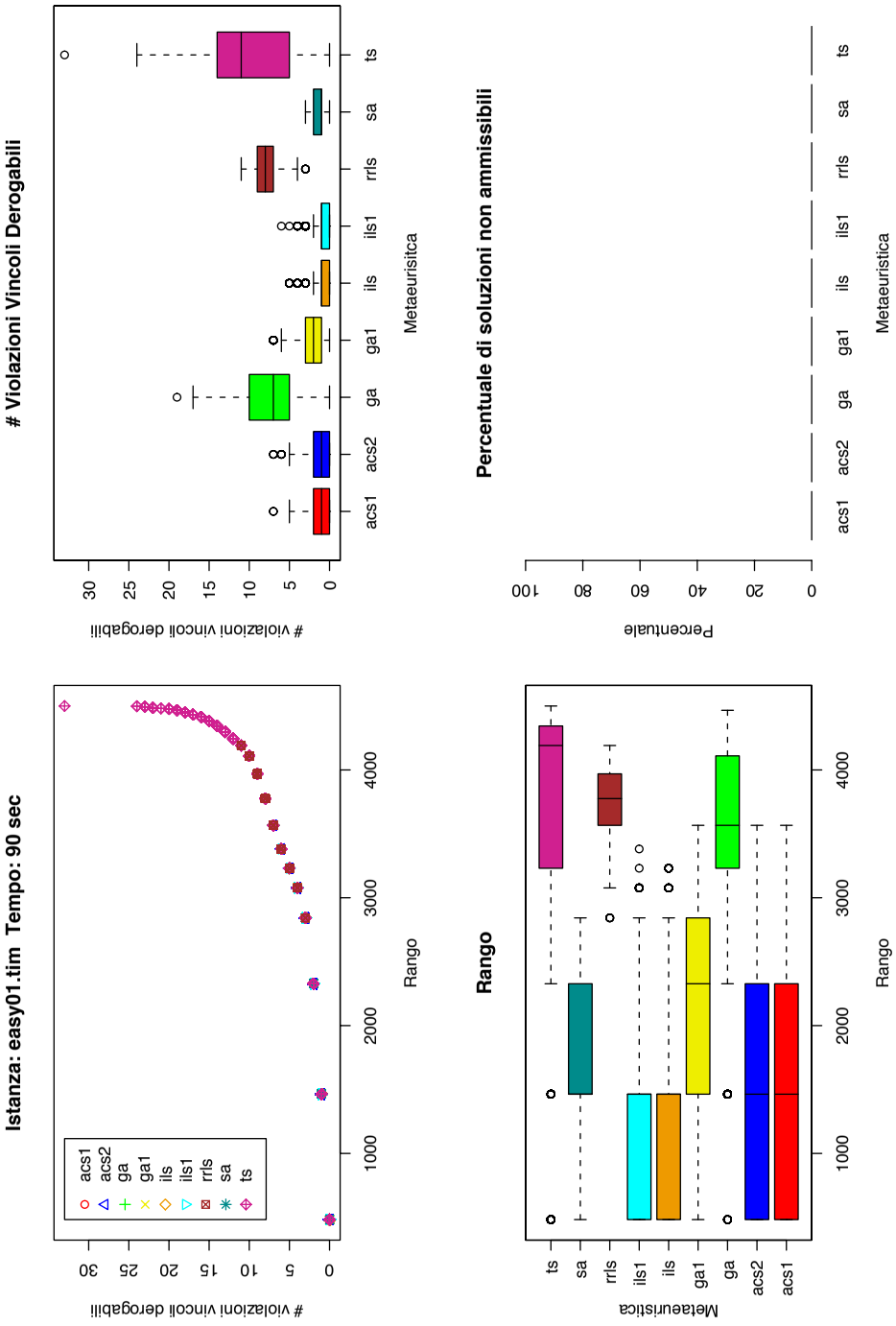

Figura A.13: Risultati istanza `small01.tim`

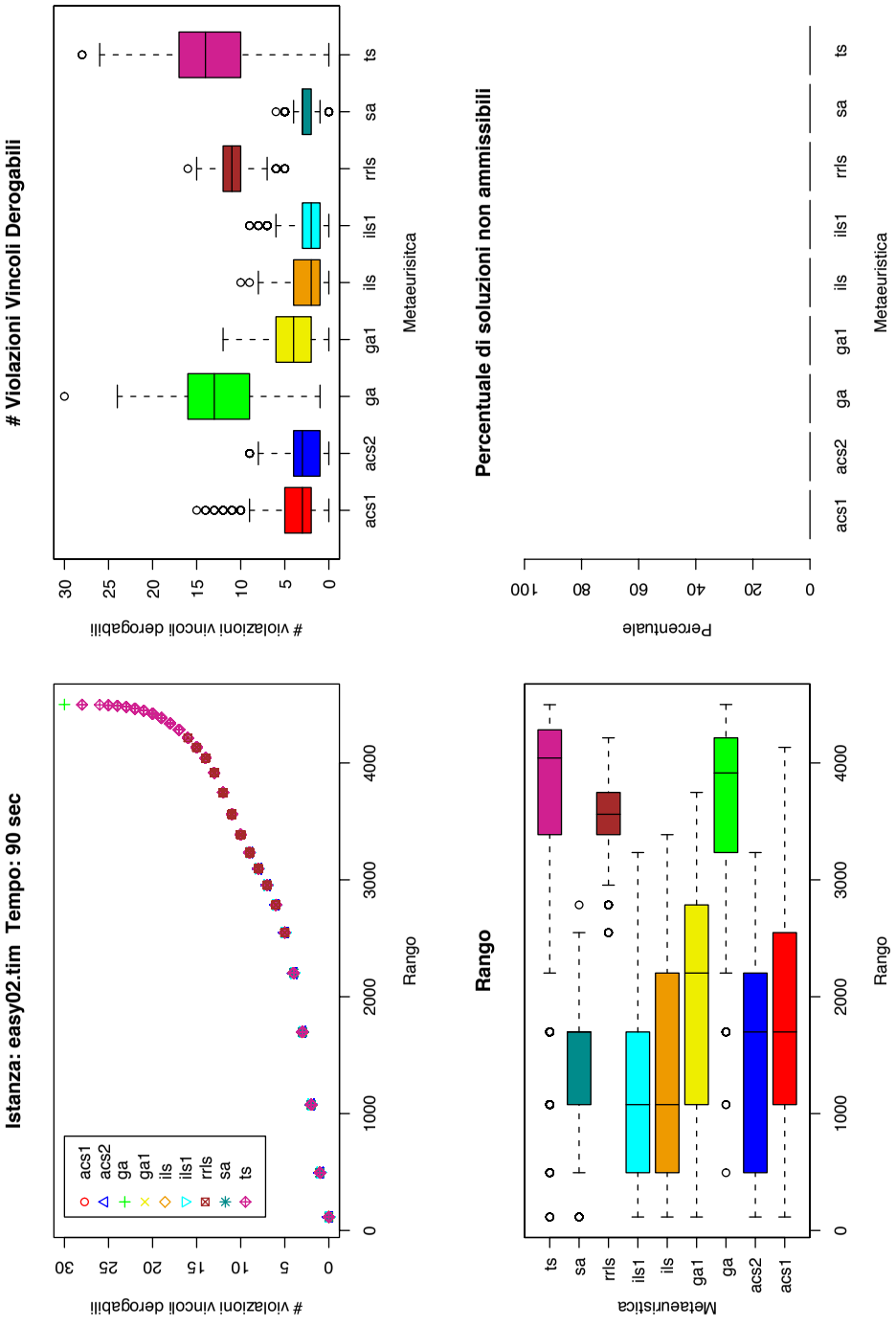

Figura A.14: Risultati istanza `small02.tim`

Tabella A.15: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small02.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	acs1	acs2	ga	ga1
acs2	7^{-6}			
ga	$< 2^{-16}$	$< 2^{-16}$		
ga1	0.0286	2.9^{-14}	$< 2^{-16}$	
ils	2.7^{-10}	**0.2312**	$< 2^{-16}$	$< 2^{-16}$
ils1	1.9^{-15}	0.0034	$< 2^{-16}$	$< 2^{-16}$
rrls	$< 2^{-16}$	$< 2^{-16}$	2.5^{-11}	$< 2^{-16}$
sa	5.6^{-6}	**0.6085**	$< 2^{-16}$	$< 2^{-16}$
ts	$< 2^{-16}$	$< 2^{-16}$	**0.1488**	$< 2^{-16}$

Tabella A.16: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small02.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ils	ils1	rrls	sa
ils1	**0.2312**			
rrls	$< 2^{-16}$	$< 2^{-16}$		
sa	0.0117	9.4^{-7}	$< 2^{-16}$	
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

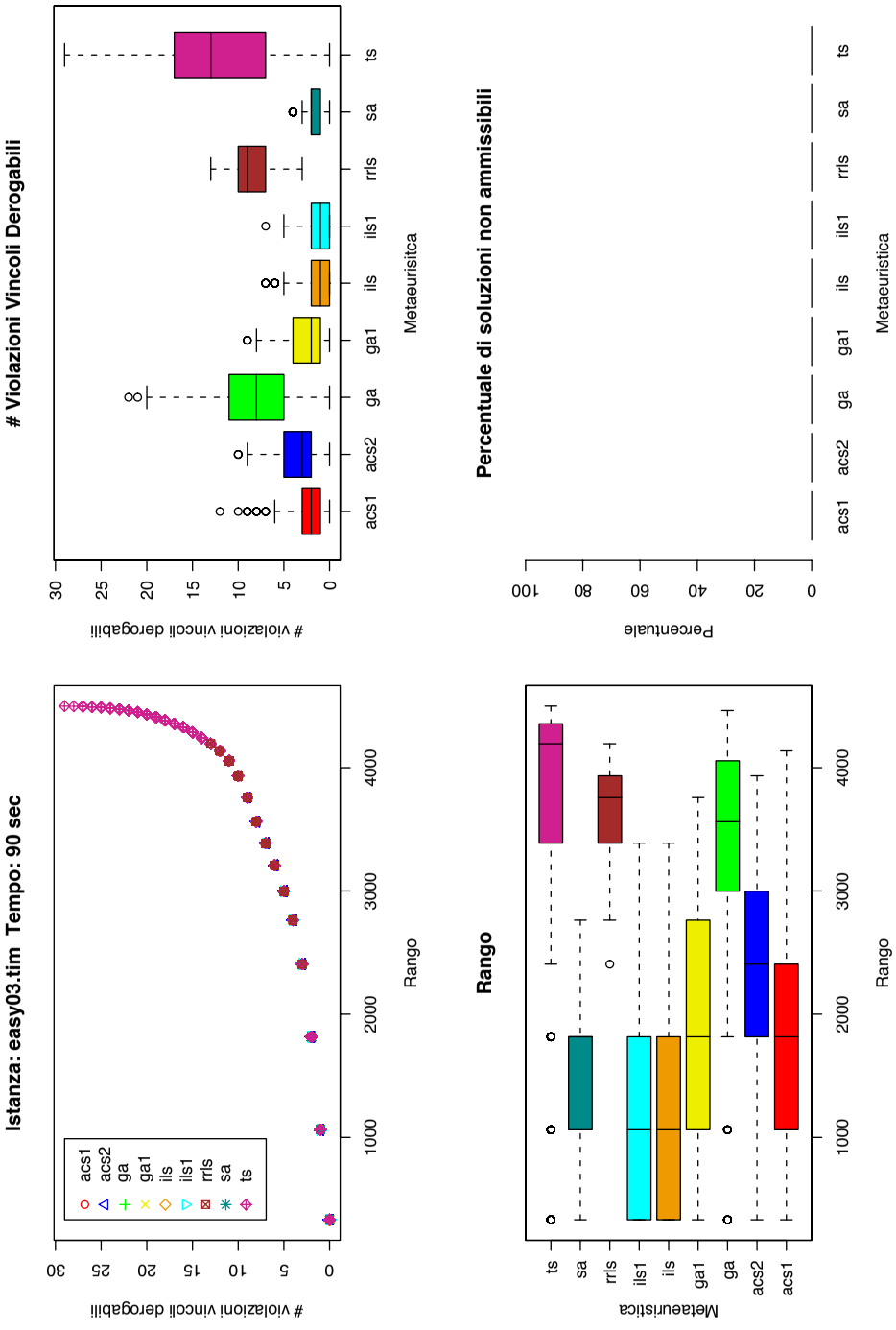

Figura A.15: Risultati istanza `small03.tim`

Tabella A.17: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small03.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	acs1	acs2	ga	ga1
acs2	$< 2^{-16}$			
ga	$< 2^{-16}$	$< 2^{-16}$		
ga1	7.3^{-9}	2.3^{-12}	$< 2^{-16}$	
ils	9.4^{-10}	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$
ils1	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$
rrls	$< 2^{-16}$	$< 2^{-16}$	**0.14**	$< 2^{-16}$
sa	**0.43**	$< 2^{-16}$	$< 2^{-16}$	7.1^{-11}
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

Tabella A.18: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small03.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ils	ils1	rrls	sa
ils1	**0.19**			
rrls	$< 2^{-16}$	$< 2^{-16}$		
sa	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

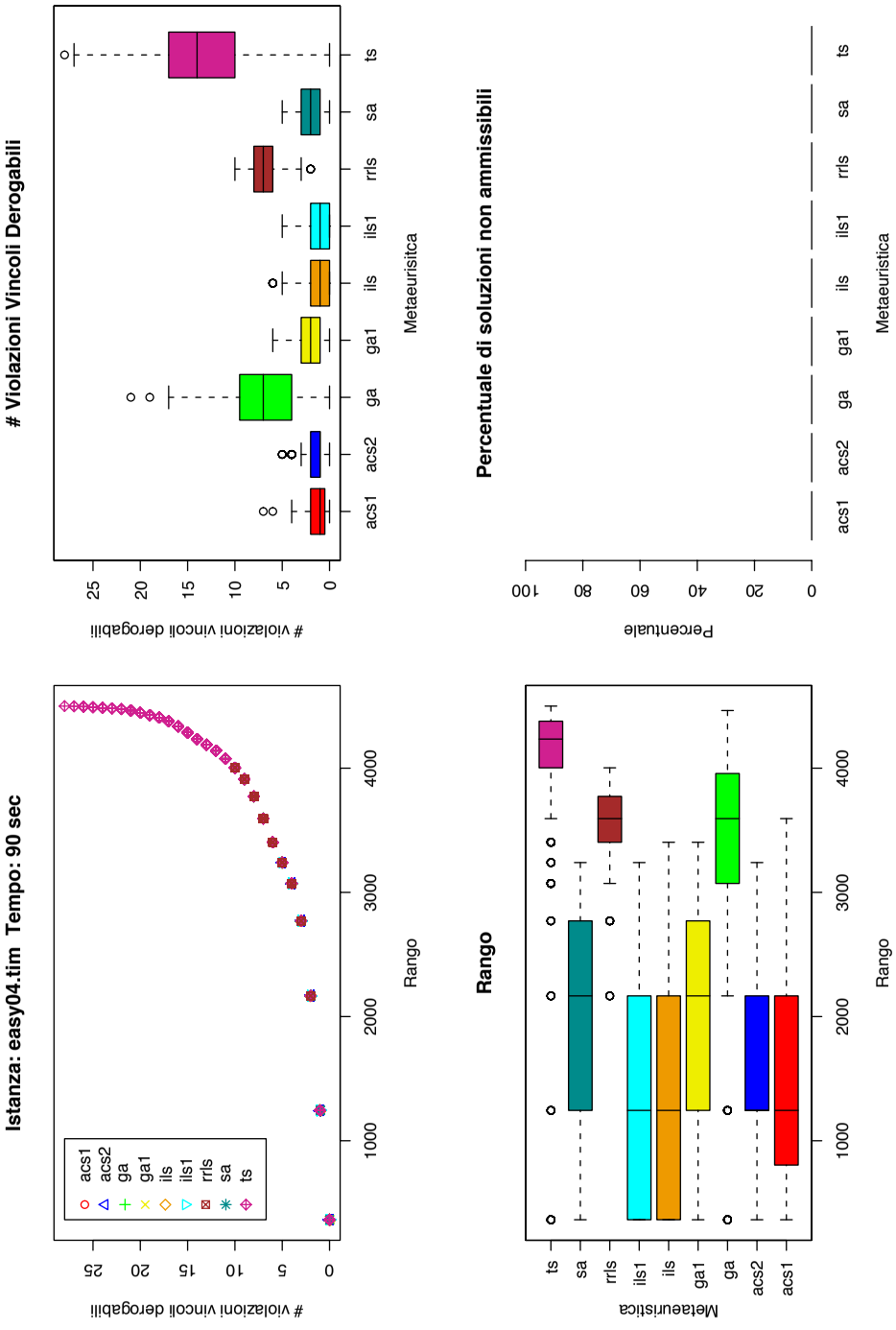

Figura A.16: Risultati istanza `small04.tim`

Tabella A.19: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small04.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	acs1	acs2	ga	ga1
acs2	**0.77047**			
ga	$< 2^{-16}$	$< 2^{-16}$		
ga1	$< 2^{-16}$	1.4^{-12}	$< 2^{-16}$	
ils	**0.4458**	**1**	$< 2^{-16}$	9.2^{-9}
ils1	0.01594	0.00058	$< 2^{-16}$	$< 2^{-16}$
rrls	$< 2^{-16}$	$< 2^{-16}$	**1**	$< 2^{-16}$
sa	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	**0.05161**
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

Tabella A.20: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small04.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in grassetto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ils	ils1	rrls	sa
ils1	0.00027			
rrls	$< 2^{-16}$	$< 2^{-16}$		
sa	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

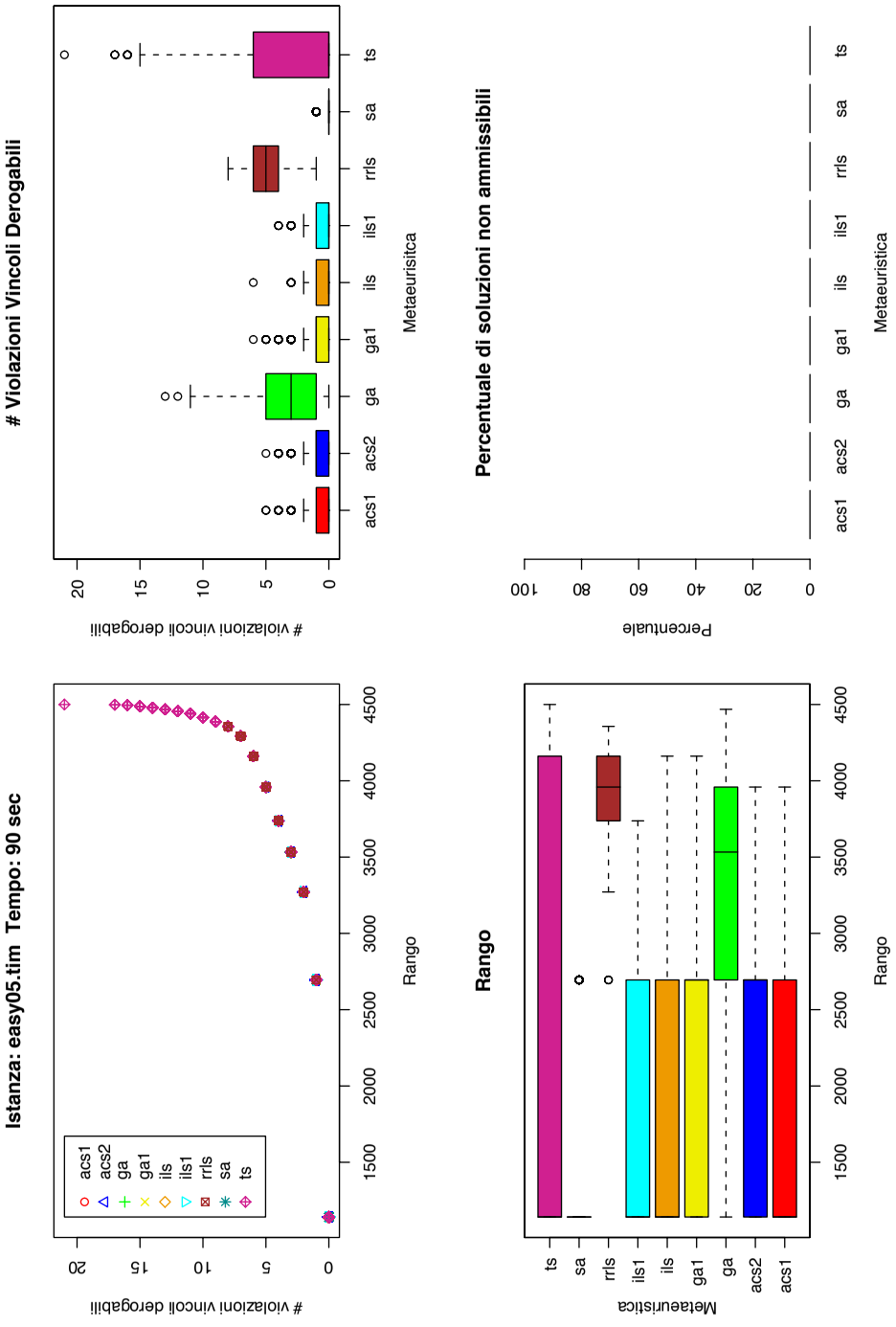

Figura A.17: Risultati istanza `small05.tim`

Tabella A.21: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small05.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	acs1	acs2	ga	ga1
acs2	**0.6377**			
ga	$< 2^{-16}$	$< 2^{-16}$		
ga1	**0.06516**	**0.46246**	$< 2^{-16}$	
ils	6.2^{-13}	8.4^{-16}	$< 2^{-16}$	$< 2^{-16}$
ils1	1.3^{-10}	2.8^{-13}	$< 2^{-16}$	$< 2^{-16}$
rrls	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$
sa	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$
ts	3.7^{-5}	0.00037	1.3^{-9}	0.00726

Tabella A.22: *p-value* dell'ipotesi nulla che le distribuzioni delle soluzioni siano uguali per l'istanza `small05.tim`. Il livello di significatività con cui si rigetta l'ipotesi nulla è 0.95. Valori del *p-value* minori di 0.05 sono sufficienti per rifiutare l'ipotesi nulla in favore dell'ipotesi alternativa, mentre valori del *p-value* maggiori di 0.05 (in neretto nella tabella) non ci permettono di rifiutare l'ipotesi nulla.

	ils	ils1	rrls	sa
ils1	**0.63775**			
rrls	$< 2^{-16}$	$< 2^{-16}$		
sa	2.1^{-13}	3.4^{-15}	$< 2^{-16}$	
ts	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$	$< 2^{-16}$

Appendice B

Metodo F-Race

In questa appendice sono riportati gli andamenti completi del metodo F-Race, descritto nel Paragrafo 3.2.2, per la scelta della configurazione migliore degli algoritmi Ant Colony System (ACS), illustrato nel Paragrafo 2.4.2 e Algoritmo Memetico (AM), illustrato nel Paragrafo 2.4.4.

La scelta dei parametri per la determinazione degli insiemi Θ_0 è stata descritta nel Paragrafo 3.3.1 e nel Paragrafo 3.3.2, rispettivamente per Ant Colony System e per Algoritmo Memetico.

B.1 Andamento di F-Race su Ant Colony System

Tabella B.1: Andamento completo di F-Race su ACS. Alla k-esima iterazione dell'algoritmo, $|\Theta_{k-1}|$ indica il numero di configurazioni candidate ancora in gara.

k	$\lvert\Theta_{k-1}\rvert$	Miglior configurazione	Numero di esperimenti effettuati
1	648	**5**	648

Tabella B.1: (Continua)

k	$\lvert\Theta_{k-1}\rvert$	Miglior configurazione	Numero di esperimenti effettuati
2	648	**493**	1296
3	648	**147**	1944
4	648	147	2592
5	648	**596**	3240
6	**144**	596	3888
7	144	**361**	4032
8	144	**416**	4176
9	144	**596**	4320
10	144	**297**	4464
11	**68**	**181**	4608
12	68	181	4676
13	68	181	4744
14	68	181	4812
15	68	**416**	4880
16	68	416	4948
17	68	416	5016
18	68	**278**	5084
19	68	278	5152
20	68	**596**	5220
21	68	596	5288
22	68	**278**	5356
23	68	278	5424
24	68	278	5492
25	68	278	5560

Tabella B.1: (Continua)

k	$\lvert\Theta_{k-1}\rvert$	Miglior configurazione	Numero di esperimenti effettuati
26	68	278	5628
27	68	278	5696
28	68	278	5764
29	68	278	5832
30	68	278	5900
31	68	278	5968
32	**52**	**608**	6036
33	52	608	6088
34	52	608	6140
35	52	608	6192
36	52	**214**	6244
37	52	214	6296
38	52	214	6348
39	52	214	6400
40	52	**206**	6452

B.2 Andamento di F-Race su Algoritmo Memetico

Tabella B.2: Andamento completo di F-Race su AM. Alla k-esima iterazione dell'algoritmo, $|\Theta_{k-1}|$ indica il numero di configurazioni candidate ancora in gara.

k	$\|\Theta_{k-1}\|$	Miglior configurazione	Numero di esperimenti effettuati
1	675	**14**	675
2	675	**101**	1350
3	675	**440**	2025
4	675	440	2700
5	675	440	3375
6	**119**	**22**	4050
7	119	22	4169
8	119	**440**	4288
9	119	**15**	4407
10	119	**440**	4526
11	119	**39**	4645
12	119	39	4764
13	119	39	4883
14	**63**	39	5002
15	63	39	5065
16	63	**5**	5128
17	63	5	5191
18	63	5	5254
19	63	**39**	5317

Tabella B.2: (Continua)

k	$\lvert\Theta_{k-1}\rvert$	Miglior configurazione	Numero di esperimenti effettuati
20	63	**5**	5380
21	63	5	5443
22	63	5	5506
23	63	5	5569
24	63	5	5632
25	63	5	5695
26	63	5	5758
27	63	5	5821
28	63	5	5884
29	63	5	5947
30	63	5	6010
31	63	**15**	6073
32	63	15	6136
33	63	15	6199
34	63	**5**	6262
35	63	5	6325
36	63	5	6388
37	63	5	6451
38	63	**15**	6514
39	63	15	6577
40	63	15	6640
41	63	15	6703

Appendice C

Il CD-ROM allegato

Il CD-ROM allegato alla tesi contiene tutti i codici sorgenti C++ degli algoritmi implementati, gli script di shell (BASH) per l'esecuzione degli esperimenti e i dati prodotti. Il codice sorgente del metodo F-Race, e gli script per l'analisi statistica dei dati sono scritti in linguaggio R[1].

acs/ Sorgenti C++ dell'algoritmo Ant Colony System, descritto nel Paragrafo 2.4.2.

ga/ Sorgenti C++ dell'Algoritmo Memetico, descritto nel Paragrafo 2.4.4.

ils/ Sorgenti C++ dell'algoritmo Ricerca Locale Iterata, descritto nel Paragrafo 2.4.3.

meta-ga/ Sorgenti C++ del Meta-Algoritmo Genetico, descritto nel Paragrafo 2.4.1.

rrls/ Sorgenti C++ dell'algoritmo Random Restart Local Search, usato come benchmark per le nostre implementazioni.

[1] `http://www.r-project.org/`

MHN_Starter-kit/ Starter Kit C++ fornito dal Metaheuristics Network, descritto nel Paragrafo 2.3.1.

Comet/ Sorgenti in R e risultati del metodo F-Race, descritto nel Paragrafo 3.2.

instances/MHN-test/ Istanze del problema usate nei confronti interni al Metaheuristics Network e nei confronti estesi alle nostre implementazioni.

instances/training/ Istanze del problema usate nella configurazione delle metaeuristiche con F-Race.

statistic/data/ Grafici dei confronti estesi, descritti nel Paragrafo 4.3. Una descrizione di come leggere i grafici è data nel Paragrafo 4.1.

statistic/data_MHN/ Grafici dei confronti interni al Metaheuristics Network, descritti nel Paragrafo 4.2. Una descrizione di come leggere i grafici è data nel Paragrafo 4.1.

statistic/output/ Risultati degli esperimenti delle nostre implementazioni.

statistic/output_MHN/ Risultati degli esperimenti delle implementazioni del Metaheuristics Network.

statistic/script/ Script di shell (BASH) con cui abbiamo lanciato gli esperimenti delle nostre implementazioni.

statistic/script_MHN/ Script di shell (BASH) con cui sono stati lanciati gli esperimenti delle implementazioni interne al Metaheuristics Network.

Bibliografia

Ausiello G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela e M. Protasi (1999). *Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties*. Springer Verlag, Berlino, Germania.

Bäck T., D. B. Fogel e Z. Michalewicz, (A cura di) (1997). *Handbook of Evolutionary Computation*. Oxford University Press, New York, NY.

Bardadym V. A. (1996). Computer-aided school and university timetabling: The new wave. In *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*. A cura di E. K. Burke e P. Ross, volume 1153 di *Lecture Notes in Computer Science*, pp. 22–45, Heidelberg, Germania. Springer Verlag.

Billingsley P. (1986). *Probability and Measure*. John Wiley & Sons, New York, NY, seconda edizione.

Birattari M., T. G. Stützle, L. Paquete e K. Varrentrapp (2002). A racing algorithm for configuring metaheuristics. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. A cura di W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Ba-

lakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke e N. Jonoska, pp. 11–18, New York, NY. Morgan Kaufmann Publishers.

Blum C. e A. Roli (2001). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. Relazione Tecnica TR/IRIDIA/2001-13, IRIDIA, Université Libre de Bruxelles, Bruxelles, Belgio. Sottomesso a ACM Computing Surveys.

Blum C., A. Roli e M. Dorigo (2001). HC-ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of MIC'2001 - Meta-heuristics International Conference*, volume 2, pp. 399–403, Porto, Portogallo.

Blum C., S. Correia, O. Rossi-Doria, M. Snoek, M. Dorigo e B. Paechter (2002). A GA evolving instructions for a timetable builder. In *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*. A cura di E. K. Burke e P. De Causmaecker, pp. 120–123, KaHo St.-Lieven, Gent, Belgio.

Burke E. K. e M. W. Carter, (A cura di) (1997). *Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling*, University of Toronto, Toronto, Canada.

Burke E. K. e M. W. Carter, (A cura di) (1998). *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT'97, Toronto, Canada, August 20-22, 1997, Selected Papers*, volume 1408 di *Lecture Notes in Computer Science*, Heidelberg, Germania. Springer Verlag.

Burke E. K. e P. De Causmaecker, (A cura di) (2002). *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*, KaHo St.-Lieven, Gent, Belgio.

Burke E. K. e W. Erben, (A cura di) (2000). *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling*, Costanza, Germania.

Burke E. K. e W. Erben, (A cura di) (2001). *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Konstanz, Germany, August 16-18, 2000, Selected Papers*, volume 2079 di *Lecture Notes in Computer Science*, Heidelberg, Germania. Springer Verlag.

Burke E. K. e P. Ross, (A cura di) (1996). *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*, volume 1153 di *Lecture Notes in Computer Science*, Heidelberg, Germania. Springer Verlag.

Burke E. K., D. Corne, B. Paechter e P. Ross, (A cura di) (1995). *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling*, Edimburgo, Regno Unito.

Calegari P., G. Coray, A. Hertz, D. Kobler e P. Kuonen (1999). A taxonomy of evolutionary algorithms in combinatorial optimization. *Journal of Heuristics*, **5**, 145–158.

Cerný V. (1985). A thermodynamical approach to the traveling salesman problem. *Journal of Optimization Theory and Applications*, **45**(1), 41–51.

Chiarandini M. e T. G. Stützle (2002). An experimental analysis of metaheuristics for course timetabling. Relazione Tecnica AIDA-02-05, FG Intellektik, FB Informatik, TU Darmstadt, Germania.

Conover W. J. (1999). *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, terza edizione.

Cooper T. B. e J. H. Kingston (1996). The complexity of timetable construction problems. In *Practice and Theory of Automated Timetabling, First International Conference, Edinburgh, U.K., August 29 - September 1, 1995, Selected Papers*. A cura di E. K. Burke e P. Ross, volume 1153 di *Lecture Notes in Computer Science*, pp. 283–295, Heidelberg, Germania. Springer Verlag.

Cramer N. (1985). A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*. A cura di J. J. Grefenstette, pp. 183–187, Pittsburgh, PA.

de Werra D. (1995). An introduction to timetabling. *European Journal of Operational Research*, **19**, 151–162.

de Werra D. (1997). The combinatorics of timetabling. *European Journal of Operational Research*, **96**, 504–513.

Dean A. e D. Voss (1999). *Design and Analysis of Experiments*. Springer Verlag, New York, NY.

Deneubourg J.-L., S. Aron, S. Goss e J.-M. Pasteels (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behaviour*, **3**, 159–168.

Dorigo M. (1992). *Ottimizzazione, apprendimento automatico, ed algoritmi basati su metafora naturale*. Tesi di Dottorato di Ricerca, Politecnico di Milano, Milano, Italia.

Dorigo M. e G. Di Caro (1999). The Ant Colony Optimization meta-heuristic. In *New Ideas in Optimization*. A cura di D. Corne, M. Dorigo e F. Glover, pp. 11–32. McGraw-Hill, Londra, Regno Unito.

Dorigo M. e L. M. Gambardella (1997). Ant Colony System: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computation*, **1**(1), 53–66.

Dorigo M. e T. G. Stützle (2002). The ant colony optimization metaheuristic: Algorithms, applications and advances. In *Handbook of Metaheuristics*. A cura di F. Glover e G. Kochenberger, volume 57 di *International Series in Operations Research & Management Science*, pp. 251–285. Kluwer Academic Publishers, Norwell, MA.

Dorigo M., V. Maniezzo e A. Colorni (1996). Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, **26**(1), 29–41.

Dorigo M., G. Di Caro e L. M. Gambardella (1999). Ant algorithms for discrete optimization. *Artificial Life*, **5**(2), 137–172.

Fogel L. J., A. J. Owens e M. J. Walsh (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, NY.

Garey M. R. e D. S. Johnson (1979). *Computers and Intractability / A Guide to the Theory of $\mathcal{NP}$-Completeness*. W.H. Freeman & Company, San Francisco, CA.

Glover F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, **13**, 533–549.

Glover F. e M. Laguna (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, seconda edizione.

Holland J. H. (1975). *Adaptation in natural artificial systems*. University of Michigan Press, Ann Arbor, MI.

Holm S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, **6**, 65–70.

Kirkpatrick S., C. D. Gelatt e M. P. Vecchi (1983). Optimization by simulated annealing. *Science*, **220**(4598), 671–680.

Knowles J. e O. Rossi-Doria (2001). Representation, neighbourhood and local search for the timetabling problem. Documentazione interna del Metaheuristics Network. Metaheuristic Network First Milestone, 16 Novembre 2001.

Lourenço H. R., O. Martin e T. G. Stützle (2002). Iterated local search In *Handbook of Metaheuristics*. A cura di F. Glover e G. Kochenberger, volume 57 di *International Series in Operations Research & Management Science*, pp. 321–353. Kluwer Academic Publishers, Norwell, MA.

Maron O. e A. Moore (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. In *Advances in Neural Information Processing Systems*. A cura di G. T. . J. A. Jack D. Cowan, volume 6, pp. 59–66, San Francisco, CA. Morgan Kaufmann.

Metropolis N., A. Rosenbluth, M. Rosenbluth, A. Teller e E. Teller (1953).

Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**, 1087–1092.

Moscato P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Relazione Tecnica Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, CA.

Moscato P. (1999). Memetic algorithms: A short introduction In *New Ideas in Optimization*. A cura di D. Corne, M. Dorigo e F. Glover, pp. 219–234. McGraw-Hill, Londra, Regno Unito.

Mühlenbein H. e H.-M. Voigt (1995). Gene pool recombination in genetic algorithms. In *Proceedings of the Metaheuristics Conference*. A cura di I. H. Osman e J. P. Kelly, Norwell, MA. Kluwer Academic Publishers.

Osman I. e G. Laporte (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, **63**, 513–623.

Press W. H., S. A. Teukolsky, W. T. Vetterling e B. P. Flannery (1993). *Numerical Recipes in C*. Cambridge University Press, seconda edizione.

Rechenberg I. (1973). *Evolution strategy: Optimization of technical systems by means of biological evolution*. Fromman-Holzboog, Stuttgart, Germania.

Rossi-Doria O. e B. Paechter (2002). Timetabling problem (updated). Documentazione interna del Metaheuristics Network.

Rossi-Doria O., M. Sampels, M. Chiarandini, J. Knowles, M. Manfrin, M. Mastrolilli, L. Paquet e B. Paechter (2002a). A comparison of the performance of differente metaheuristics on the timetabling problem. In

Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling. A cura di E. K. Burke e P. De Causmaecker, pp. 115–119, KaHo St.-Lieven, Gent, Belgio.

Rossi-Doria O., C. Blum, J. Knowles, M. Sampels, K. Socha e B. Paechter (2002b). A local search for the timetabling problem. In *Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling*. A cura di E. K. Burke e P. De Causmaecker, pp. 124–127, KaHo St.-Lieven, Gent, Belgio.

Schaerf A. (1995). A survey of automated timetabling. Relazione Tecnica CS-R9567, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, Paesi Bassi.

Socha K., J. Knowles e M. Sampels (2002). A $\mathcal{MAX}$-$\mathcal{MIN}$ ant system for the university course timetabling problem In *Ant Algorithms, Third International Workshop, ANTS 2002*. A cura di M. Dorigo, G. Di Caro e M. Sampels, volume 2463 di *Lecture Notes in Computer Science*, pp. 1–13, Berlino, Germania. Springer Verlag.

Socha K., M. Sampels e M. Manfrin (2003). Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In *Proceedings of 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization (EvoCOP'2003)*, volume 2611 di *Lecture Notes in Computer Science*, Berlino, Germania. Springer Verlag. Accettato per la pubblicazione.

Stützle T. G. (1998). *Local Search Algorithms for Combinatorial Problems - Analysis, Algorithms and New Applications*. Tesi di Dottorato di Ricerca,

Fachbereich Informatik - Technische Universität Darmstadt. Pubblicato nel 1999 - Infix, Sankt Augustin, Germania - volume 220 di DISKI.

Stützle T. G. e M. den Besten (2000). Guidelines for the production of software. Documentazione interna del Metaheuristics Network.

Stützle T. G. e M. Dorigo (2002). A short convergence proof for a class of ACO algorithms. *IEEE Transactions on Evolutionary Computation*, **6**(4), 358–365.

Stützle T. G. e H. H. Hoos (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. *Future Generation Computer System*, **16**(8), 889–914.

Voss S., S. Martello, I. Osman e C. Roucairol (1999). *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers.

Willemen R. J. (2002). *School timetable construction: algorithms and complexity*. Tesi di Dottorato di Ricerca, Technische Universiteit Eindhoven, Eindhoven, Paesi Bassi.