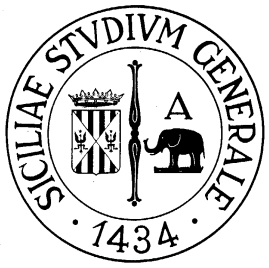

Università degli Studi di Catania

Dipartimento di Matematica e Informatica

Corso di Laurea Magistrale in Informatica

Deployed Sudoku Solver

Risolutore per sudoku distribuito con Docker e Kubernetes

Relazione Progetto di Sistemi Cloud

Simone Cartalemi

1000050366

Docenti del corso

Prof. Pappalardo Giuseppe, Prof. Fornaia Andrea

Anno Accademico 2023/2024

Indice

Introduzione

Il problema del Sudoku

Il Sudoku, nato in Giappone, è diventato un passatempo amato in tutto il mondo per la sua combinazione di logica, deduzione e strategia. Questo enigma numerico è una forma di rompicapo che si basa su una griglia 9×9 divisa in nove sottogriglie 3×3. L'obiettivo è riempire la griglia in modo che ogni riga, colonna e sottogriglia contenga tutti i numeri da 1 a 9 senza ripetizioni, tenendo in considerazione dei numeri già inseriti in partenza che ne determinano la difficoltà. Nonostante la sua apparente semplicità, il Sudoku presenta un'enorme complessità computazionale dovuta al vasto spazio di ricerca delle possibili combinazioni. Per risolvere un Sudoku, è possibile seguire diverse strategie e tecniche.

Dal punto di vista matematico, il Sudoku può essere affrontato utilizzando diverse strategie e tecniche. Una delle prime osservazioni è che la somma di tutti i numeri da 1 a 9 è 45, quindi ogni riga, colonna e sottogriglia deve sommare 45. Questa proprietà fornisce una restrizione importante quando si risolve il Sudoku, ma non è l'unica.

Anche la teoria dei grafi si è interessata del problema: si possono studiare le proprietà strutturali del grafo del Sudoku e utilizzare concetti come il teorema di *NP-completezza* per dimostrare che risolvere il Sudoku è un problema computazionalmente difficile.

Il concetto chiave del Sudoku sta proprio nella sua complessità che risiede nella sua natura combinatoria: aumenta con il numero di celle vuote e la presenza di più soluzioni possibili. Un'istanza può avere zero, una o più soluzioni. Se il puzzle è ben progettato, avrà una sola soluzione. In alcuni casi, è necessario utilizzare strategie di risoluzione sofisticate per risolvere il puzzle, che pongono sfide stimolanti

agli appassionati di enigmi di ogni livello.

Approccio Metaeuristico mediante Tabu Search

In questo contesto, gli algoritmi metaeuristici emergono come un'opzione promettente per la risoluzione delle istanze del Sudoku. Gli algoritmi metaeuristici sono tecniche di ottimizzazione che guidano la ricerca di soluzioni in spazi di ricerca complessi, utilizzando strategie ispirate ai processi naturali o ad analogie con fenomeni osservabili.

Queste tecniche si adattano bene alla natura del problema del Sudoku in quanto sono in grado di esplorare in modo efficiente lo spazio delle soluzioni, fornendo soluzioni di qualità accettabile in tempi ragionevoli. Nell'ambito della presente relazione finale, si propone la distribuzione e l'utilizzo del **TabuSearch** mediante un'applicazione gestita da container e orchestratore.

L'obiettivo di questa applicazione è quello di rendere accessibile il risolutore di Sudoku basato su Tabu Search, mediante l'utilizzo di Docker, per incapsulare il suo ambiente di esecuzione. Questi container sono gestiti e orchestrati tramite Kubernetes, che permette la distribuzione scalabile e la gestione del carico di lavoro attraverso diversi nodi in un cluster.

Capitolo 1

Architettura dell'applicazione

1.1 Infrastruttura

L'applicazione non è altro che l'implementazione del noto design pattern *Master Worker Architecture* (anche nota come *Master-Slave Architecture*), un modello computazionale per sistemi distribuiti in cui un nodo funge da "Master" e i restanti nodi da "Worker". Il nodo principale controlla le operazioni dei nodi lavoratori che gestiscono i calcoli e memorizzano i risultati: il *Master* è il responsabile del coordinamento di tutti i nodi, che riceve le richieste di istanze da risolvere; i *Worker* si occupano di svolgere questo compito e restituire la risposta al *Master*. Questa soluzione agevola l'elaborazione parallela, consentendo calcoli e gestione dei dati più rapidi, rendendola un'architettura ampiamente utilizzata nei sistemi di database, nel cloud computing, nelle reti di telecomunicazioni e nei sistemi di controllo distribuito.

Questo paradigma è particolarmente potente quando viene implementato utilizzando container e orchestrata tramite la combinazione di Docker e Kubernetes. I container, essendo unità leggere e portabili, possono essere avviati, replicati e fermati rapidamente in base alla domanda e Kubernetes, con le sue capacità di orchestrazione, permette di gestire dinamicamente il numero di container in esecuzione. Questo approccio permette una scalabilità elastica e automatica, ottimizzando l'uso delle risorse del sistema e garantendo che essi possano operare in parallelo senza interferenze reciproche.

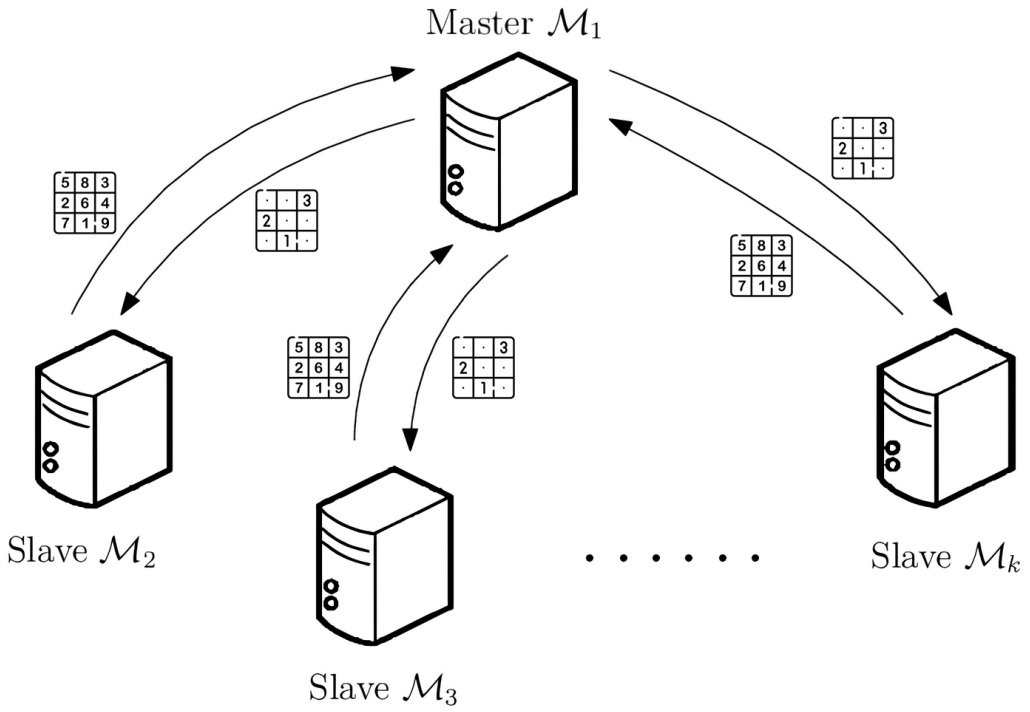

Figura 1.1: Schema generale architettura Master-Slave

1.2 Funzionamento del TabuSearch

Rispetto agli algoritmi di ottimizzazione classici e ai metodi iterativi, le metaeuristiche non garantiscono che si possa trovare una soluzione globalmente ottimale per alcune classi di problemi [1]. Nell'ottimizzazione combinatoria, ricercando un ampio insieme di soluzioni ammissibili, le metaeuristiche possono spesso trovare buone soluzioni con meno sforzo computazionale rispetto agli algoritmi di ottimizzazione, ai metodi iterativi o alle semplici euristiche. Tra quelle esistenti ci concentriamo sul Tabu Search, che si è precedentemente rivelato abbastanza buono per questo scopo.

La ricerca tabù (chiamata anche Tabu Search o TS), è una tecnica ampiamente utilizzata che si basa su principi ispirati al comportamento di ricerca degli animali. L'algoritmo prende il nome dal concetto di "tabù", che rappresenta una lista di mosse proibite o, nel nostro caso, soluzioni precedentemente esplorate. Questo metodo mira a esplorare lo spazio delle soluzioni in modo efficiente e a trovare soluzioni di alta qualità [2] [3].

L'approccio della ricerca tabù inizia con la generazione casuale di una soluzione iniziale, che viene valutata in base a una funzione di costo chiamata *Fitness*.

Successivamente, vengono esplorate soluzioni vicine attraverso una serie di mosse locali, che possono includere scambi, inversioni o altre trasformazioni consentite nel contesto del problema specifico. Uno dei punti chiave dell'algoritmo è quello di evitare la stagnazione in minimi locali o cicli infiniti, applicando regole tabù che vietano determinate mosse o soluzioni già visitate. Questo permette alla ricerca di esplorare nuove regioni dello spazio delle soluzioni e di superare ostacoli locali.

La ricerca utilizza una procedura di investigazione locale o di "vicinato" per passare in modo iterativo da una potenziale soluzione x ad una soluzione migliorata x' nel vicinato di x, fino a quando non è stato soddisfatto qualche criterio di arresto (generalmente, un limite di tentativi o una soglia di punteggio).

L'aspetto quindi fondamentale della ricerca tabù risiede nella gestione della lista tabù, che può essere implementata come una struttura dati, che tiene traccia delle mosse proibite e della funzione di ricerca del vicinato. La lista viene aggiornata dinamicamente durante il processo di ricerca, con l'aggiunta di nuove mosse tabù e la rimozione delle mosse più vecchie per evitare un accumulo eccessivo di vincoli, per questo viene anche chiamata memoria a breve termine; la funzione di ricerca del vicinato esplora le possibili soluzioni che potrebbero migliorare l'esito della fitness e aiuta a dirigere la direzione di ricerca dell'algoritmo nello spazio delle soluzioni.

1.3 Immagini dei container

Il sistema è stato configurato per massimizzare l'efficienza e minimizzare l'overhead. Esso è composto da tre immagini Docker principali, ciascuna progettata per ottimizzare specifici aspetti del processo di risoluzione e distribuzione del Sudoku.

1.3.1 GCC Alpine

Questa immagine si basa su una versione personalizzata di Alpine Linux[1], integrata con gli strumenti essenziali di *GCC*. L'immagine ufficiale di *GCC*, sebbene completa, risulta essere troppo pesante per i nostri scopi. La scelta di Alpine Linux

[1]Distribuzione Linux progettata principalmente per utenti che apprezzano la sicurezza e l'efficienza delle risorse. Per via delle sue dimensioni ridotte, è fortemente impiegato in sistemi che forniscono rapidi tempi di avvio.

permette di mantenere l'immagine finale leggera e veloce, il che è essenziale per un'efficace distribuzione scalabile. Questa immagine sarà la base per l'integrazione nel risolutore vero e proprio, consentendo una rapida compilazione del codice e una più facile manutenzione.

1.3.2 Sudoku Solver

Il cuore dell'intera applicazione è rappresentato dall'immagine *sudoku-solver*. Questo componente è implementato in linguaggio C++, noto per la sua elevata velocità rispetto ad altri linguaggi di più alto livello. L'applicazione è strutturata secondo una pipeline dedicata, ottimizzata mediante il design pattern **pooling**[4], che gestisce l'intera esecuzione e risoluzione dei Sudoku, ottimizzando estremamente la memoria e il tempo di esecuzione. La pipeline, una volta ricevuta un'istanza di Sudoku come input, procede a risolverla utilizzando l'algoritmo delineato in precedenza. Il software (disponibile su GitHub[5, 6]) è strutturato in diversi moduli e funzioni, ciascuno dei quali svolge un compito specifico. Questa suddivisione garantisce una struttura modulare, ordinata e facilmente leggibile, facilitando così la manutenzione e l'aggiornamento del codice.

1.3.3 Server

Il *sudoku-server* si basa su un'immagine leggera di Python, basata a sua volta sulla distro Alpine. Questo server si mette in ascolto delle richieste in arrivo e, per ciascuna di esse, avvia un **thread** dedicato che gestisce l'invio del Sudoku a un'istanza del risolutore (anch'essa in ascolto) e inoltra la risposta al client prima di terminare la sua esecuzione. Questo approccio assicura che ogni richiesta sia gestita in modo efficiente, riducendo al minimo il tempo di attesa per il client.

1.4 Gestione del deploy mediante Kubernetes

Kubernetes gioca un ruolo cruciale nella distribuzione e nella gestione del sistema che è il backend del servizio. Utilizzando l'immagine del risolutore costruita

come descritto sopra, Kubernetes istanzia N POD[2] contenenti ciascuno un'istanza di container che rimangono in ascolto e non raggiungibili dall'esterno. Ogni container esegue una copia indipendente del risolutore, permettendo di processare più richieste in parallelo, in modo da facilitare la scalabilità orizzontale. A titolo di esempio, supporremo di istanziare tre container. Oltre ai risolutori, viene avviato anche il POD del container del server che, contrariamente a quanto avviene per i risolutori, è raggiungibile dall'esterno. Tra i POD di server e risolutori esiste un canale di comunicazione privato, gestito automaticamente mediante Kubernetes, che si occupa di fare anche da **Load Balancer** per scalare tra i risolutori, come illustrato schematicamente nella seguente immagine 1.2.

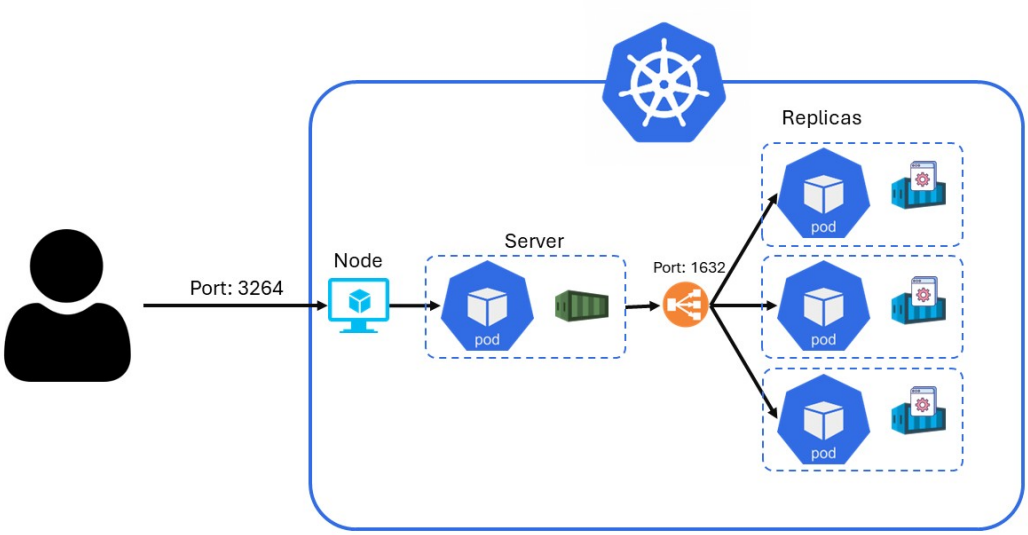

Figura 1.2: Schema di deploy dell'applicativo

[2]L'oggetto più piccolo che puoi creare in Kubernetes che rappresenta la singola istanza di una applicazione

Il modo più semplice per comprendere l'architettura è quello di esaminare la definizione della configurazione mediante il file di manifesto in formato *.yaml*, che viene diviso tra le due tipologie di container da istanziare.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: solver-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: multi-solver-deployment
  template:
    metadata:
      labels:
        app: multi-solver-deployment
    spec:
      containers:
      - name: solver
        image: sudoku-solver:v1.0
        ports:
        - containerPort: 1632
          name: solver-port
---
apiVersion: v1
kind: Service
metadata:
  name: multi-solver-service
spec:
  selector:
    app: multi-solver-deployment
  ports:
    - protocol: TCP
      port: 1632
      targetPort: 1632
      name: solver
```

La prima parte del file di manifesto definisce un **Deployment** chiamato "solver-deployment". Questo deployment specifica che devono essere create tre repliche di un POD, ognuna delle quali esegue un container con l'immagine del risolutore. Il container è configurato per ascoltare sulla porta 1632, che è denominata "solver-port". Il deployment è etichettato con "multi-solver-deployment", consentendo il facile raggruppamento e selezione di questi POD anche per una eventuale gestione manuale attraverso riga di comando.

Segue la definizione di un **Service** chiamato "multi-solver-service". Questo servizio seleziona i POD etichettati con "multi-solver-deployment" e li espone sulla porta 1632. Internamente, il traffico indirizzato a questo servizio viene inoltrato alla stessa porta dei container dei POD selezionati utilizzando il protocollo TCP, ma visibili solamente all'interno della stessa rete virtuale gestita da Kubernetes.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server-pod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: server
  template:
    metadata:
      labels:
        app: server
    spec:
      containers:
      - name: server
        image: solver-server:v1.1
        ports:
        - containerPort: 3264
          name: websocket-port
        - containerPort: 80
          name: http-port
---
apiVersion: v1
kind: Service
metadata:
  name: server-service
spec:
  selector:
    app: server
  ports:
    - protocol: TCP
      port: 3264
      targetPort: 3264
      name: websocket
    - protocol: TCP
      port: 80
      targetPort: 80
      name: http
  type: LoadBalancer
```

La seconda parte del file definisce un altro **Deployment**, denominato "server-pod", molto simile al precedente. Questo deployment, tuttavia, crea una singola replica del POD che esegue un container con l'immagine "solver-server". Questo container funge da server principale dell'applicazione.

Infine, viene definito un **Service** chiamato "server-service". Questo servizio seleziona il POD etichettato con "server" e lo espone sulla porta 3264[3]. Internamente, il traffico indirizzato a questo servizio viene inoltrato alla stessa porta del container del POD selezionato, esattamente come avveniva per il precedente, con la significativa differenza che questo servizio è di tipo "LoadBalancer", il che significa che Kubernetes configura automaticamente un bilanciamento del carico per rendere il servizio accessibile dall'esterno del cluster, fornendo un indirizzo IP esterno.

[3]La scelta delle porte 3264 e 1632 è un omaggio personale, riflettendo una preferenza simbolica

1.5 Workflow

Il flusso di lavoro del sistema è concepito per essere semplice e intuitivo. Un utente client interagisce con una pagina web che offre un'interfaccia chiara e funzionale. Questa interfaccia permette all'utente di inserire un'istanza del problema di Sudoku, che viene poi formattata e inviata al server in ascolto tramite un semplice script in JavaScript. La comunicazione tra il client e il server avviene attraverso il protocollo WebSocket sulla porta 3264. Il formato richiesto per l'invio dell'istanza del Sudoku è altrettanto semplice: gli 81 numeri della griglia (compresi tra 1 e 9, più il numero 0 per rappresentare gli spazi vuoti) devono essere racchiusi tra parentesi quadre [e] e separati da una virgola. Questo formato (meglio conosciuto come vettore JSON) semplifica la comunicazione e il controllo dell'input da parte del server.

Il server, che funge da nodo Master, avvia un thread non appena riceve una richiesta WebSocket dal client. Questo thread gestisce la connessione con uno dei nodi Worker, i risolutori, sfruttando il load balancer gestito da Kubernetes. La comunicazione tra il Master e i Worker avviene tramite socket sulla porta 1632. Il server Master, come già detto, rimane in attesa della soluzione da parte del risolutore, mantenendo la connessione aperta sia con esso che con il client. Quando il nodo Worker completa la risoluzione del Sudoku, restituisce la soluzione al Master nel medesimo formato dell'input iniziale.

Il nodo Worker riceve l'input formattato dal Master e avvia l'esecuzione del processo di risoluzione secondo l'algoritmo Tabu Search precedentemente delineato. Una volta trovata la soluzione, il risolutore invia il risultato al Master e si rimette in attesa di nuove richieste.

L'ultimo passo consiste nel restituire la soluzione dell'elaborazione al client e il thread che gestisce la connessione con esso viene quindi terminato. Il client quindi visualizzerà la soluzione del Sudoku direttamente sulla pagina web, completando così il ciclo di risoluzione.

La gestione delle richieste può avvenire in parallelo, riducendo i tempi di attesa e migliorando l'efficienza complessiva del sistema. L'architettura modulare e il flusso di lavoro ben definito garantiscono un'esperienza utente fluida e tempi di

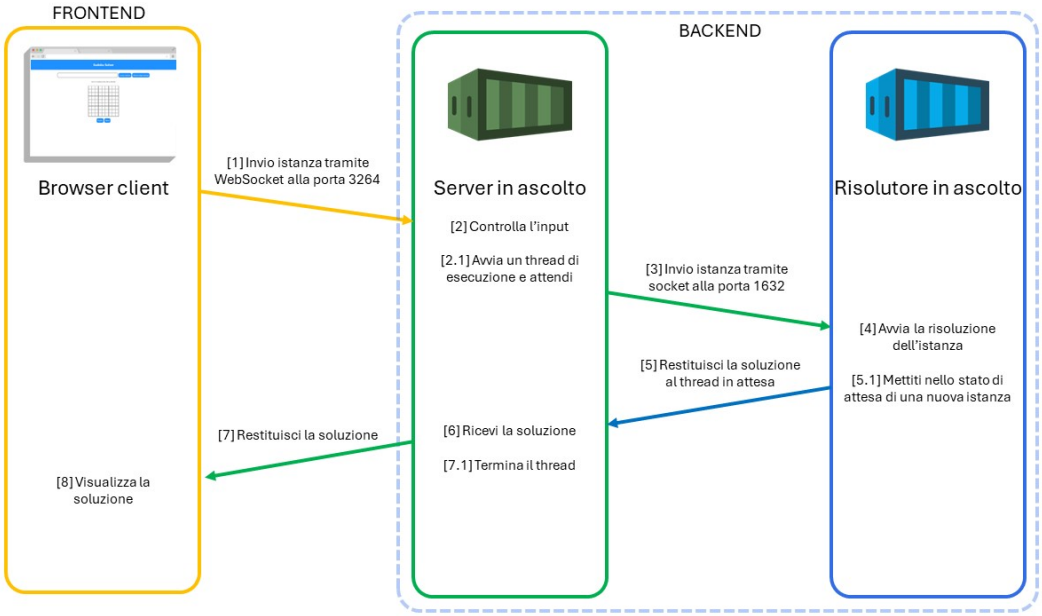

Figura 1.3: Workflow dell'applicativo

risposta rapidi, rendendo questo approccio ideale per applicazioni distribuite e ad alte prestazioni.

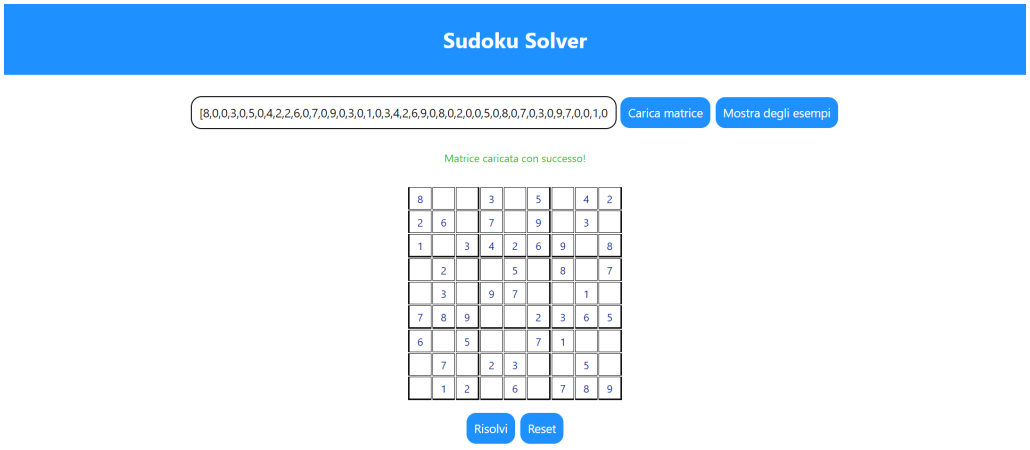

Figura 1.4: Interfaccia del browser che mostra un'istanza di Sudoku da risolvere

Ulteriori informazioni e la guida all'implementazione (testata sia in locale tramite Docker Desktop, sia su cloud con Minikube su server Ubuntu) sono disponibili nella repository [5]

Bibliografia

[1] Christian Blum e Andrea Roli. «Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison». In: *ACM Comput. Surv.* 35 (gen. 2001), pp. 268–308. DOI: 10.1145/937503.937505.

[2] Fred Glover e Manuel Laguna. *Tabu search I*. Vol. 1. Gen. 1999. ISBN: 978-0-7923-9965-0. DOI: 10.1287/ijoc.1.3.190.

[3] Wikipedia.org. *Tabu Search*. URL: https://it.wikipedia.org/wiki/Tabu_search.

[4] Wikipedia.org. *Object pool pattern*. URL: https://it.wikipedia.org/wiki/Object_pool_pattern.

[5] Simone Cartalemi. *Cloud System project-sudoku-solver - GitHub repository*. https://github.com/simone-cartalemi/cloud-system-project-sudoku-solver. 2024.

[6] Simone Cartalemi e Alessio Mezzina. *Metaheuristic Algorithms - GitHub common repository*. https://github.com/timeassassinRG/Metaheuristic-Algorithms/tree/simone-final-ai-project. Repository GitHub privata. Accessibile solo agli utenti autorizzati. 2024.