# Tecniche di ricerca operativa applicate alla generazione e risoluzione di puzzle: il gioco del domino

Nunzio Onorati s280181

**Sommario**

La seguente tesi si prefigge di dimostrare il collegamento tra tecniche di ricerca operativa, come la teoria dei grafi e la PL, con il gioco del domino. L'applicazione di queste tecniche porta alla formulazione di algoritmi efficienti, utlilizzabili per la generazione e per la risoluzione dei puzzle. Segue l'esposizione degli ambiti di applicazione di dette tecniche e l'analisi delle relative complessità. Infine si mostra un implementazione software open-source del modello usato per il gioco.

# Indice

# 1 Introduzione

I puzzle, come in generale i giochi, sono percepiti giustamente come mezzi di intrattenimento. Tuttavia rappresentano una forma educativa al problem solving e allo sviluppo del pensiero logico. In particolare, il gioco del domino risale, nelle sue prime apparizioni, al dodicesimo secolo in Cina[19] e ha raggiunto l'Europa solo nel 1800. Durante la sua diffusione sono state create numerose varianti del gioco classico. Le varianti conosciute hanno in comune:

- uso di set da gioco di lunghezza diversa
- uso di regole per il posizionamento delle tessere particolari
- numero di giocatori differente
- gioco tutti contro tutti o gioco a squadre

Tra le varie modalità quella che prevede un singolo giocatore viene anche chiamata puzzle di domino. L'obbiettivo di gioco nel caso di un puzzle è quello di trovare l'esatto e unico posizionamento delle tessere date all'interno di uno schema che presenta posizioni vuote e tessere non riposizionabili.

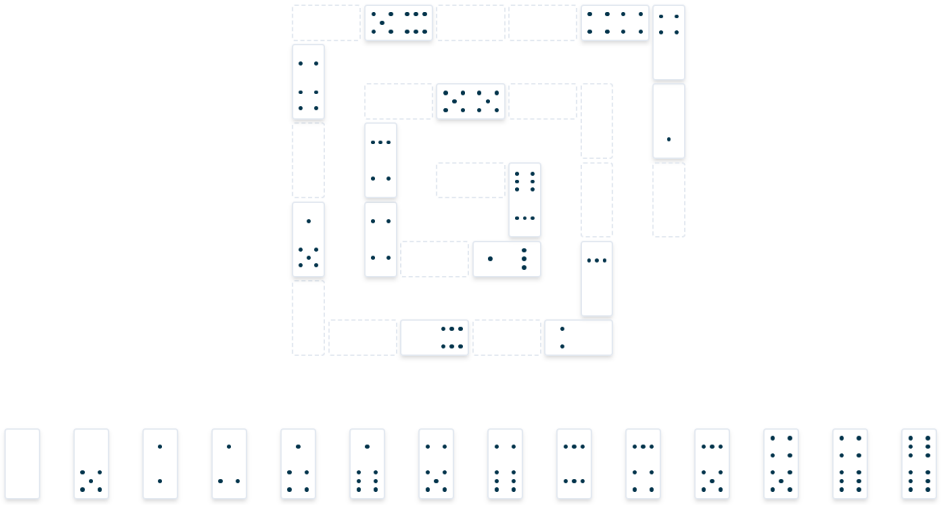

Figura 1: Un puzzle di domino da completare

Ogni set da gioco è composto da tessere. Ogni tessera ha due facce. Ogni faccia ha da 0 a N punti. Tradizionalmente i set, presenti in commercio, possono avere tessere con facce contenenti 6, 9 o 12 punti. La denominazione di un singolo set deriva dal numero più alto di punti contenuto in una tessera. Quindi esistono set da gioco chiamati rispettivamente doppio-6 (figura 3), doppio-9 o doppio-12.
Per completare un puzzle vi sono regole da rispettare:

- tutte le tessere devono essere utilizzate
- ogni tessera, ad eccezione della prima e dell'ultima, deve essere preceduta e seguita da una tessera con lo stesso numero di punti sulle facce adiacenti.

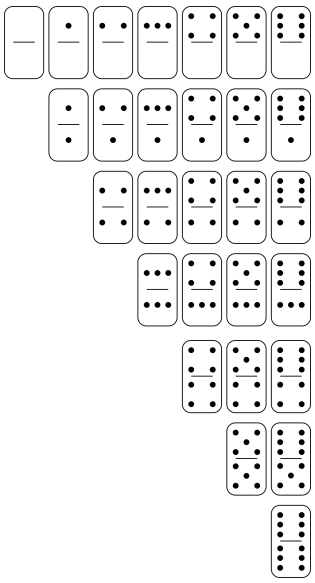

Figura 2: Un set da gioco doppio-6

La peculiarità del gioco è data dalla composizione di uno schema tale da avere una sola soluzione e che sia possibilmente non risolvibile in maniera eccessivamente semplice.

# 2 La Programmazione Lineare e gli approcci di Ricerca Operativa

Seguono nel capitolo la descrizione di metodi relativi alla Ricerca Operativa. Con il termine Ricerca Operativa ci si riferisce alla disciplina che studia un'approccio pratico e strategico ai problemi. Il termine nacque da scienziati le cui scoperte erano impiegate in ambito bellico, soprattutto nell'ottimizzazione dei costi e per cui ci si riferiva alla tipologia delle strategie come 'operative'. La materia di studio venne instituzionalizzata quando al termine delle guerre gli scienziati impiegati nello studio di queste tecniche ritornati alle loro università mantennero il riferimento alle tecniche sviluppate, innovandole e scoprendone di nuove[17]. Con il termine PL ci si riferisce ad un acronimo del termine **programmazione lineare**. La programmazione lineare è una tecnica volta a risolvere problemi di ottimizzazione. La programmazione lineare fu introdotta nel 1947 da Danzig G. B., con il testo 'Maximization of a Linear Function of Variables Subject to Linear Inequalities'[5]. A seconda del tipo di variabili del problema la programmazione lineare richiede l'utilizzo di tecniche diverse. In particolare si distinguono i problemi in cui le variabili sono continue da quelli che utilizzano variabili intere.

Ogni problema per essere risolto deve essere rappresentato tramite 3 componenti che lo descrivono:

- L'**obbiettivo** da raggiungere nella soluzione al problema;
- I **vincoli** che la soluzione deve ottemperare;
- Il tipo di **variabili** che devono essere contenute nella soluzione;

La descrizione di un problema secondo queste componenti viene detta **modellazione** del problema.

La **funzione obbiettivo** di un problema in forma classica segue la rappresentazione:

$$\min : \vec{c} \cdot \vec{x}$$

Dove $\vec{c}$ rappresenta il vettore dei "costi" delle variabili e l'obiettivo viene rappresentato in forma della minimizzazione del prodotto tra l'uso (singolo o multiplo) di una variabile per il suo costo.
I **vincoli** di un problema in forma classica seguono la rappresentazione:

$$A \cdot \vec{x} \leq \vec{b}$$

Dove $A$ rappresenta la matrice dei coefficienti relativi all'uso delle variabili per ogni vincolo, $b$ rappresenta il vettore dei limiti numeri di ogni vincolo ed i vincoli stessi sono espressi come prodotti matriciali $A \cdot x \leq b$.
Le **variabili** di un problema in forma classica seguono la rappresentazione:

$$\vec{x} \geq 0$$

## 2.1 La programmazione lineare continua

### 2.1.1 L'algoritmo del simplesso

L'approccio di risoluzione più conosciuto nell'ambito della Programmazione Lineare continua è dato dall'**algoritmo del simplesso**. L'algoritmo in questione è considerato uno dei dieci algoritmi che hanno influenzato maggiormente la tecnologia moderna nell'ultimo secolo e venne presentato per la prima volta con un report interno di **George Dantzig** per un progetto inerente le attività statistico operative dell'areonautica statunitense nel 1947. L'algoritmo del simplesso opera sul problema rappresentandolo geometricamente tramite lo spazio delle sue soluzioni. Invece di operare su tutte le possibili soluzioni valuta solamente le cosiddette "soluzioni di base" e raggiunge una soluzione ottima globale ottenendo soluzioni ottime locali progressivamente miglioranti. Nella rappresentazione dello spazio delle soluzioni pone le variabili del problema in forma classica come degli assi nello spazio (multi-dimensionale), i vincoli del problema corrispondono a degli iper-piani e le soluzioni ottime localmente/globalmente corrispondono agli spigoli nati dall'intersezione degli iper-piani. La forma geometrica convessa, nata dall'intersezione degli iperpiani, viene chiamata in letteratura **politopo**[18]. Un problema, risolvibile, rappresentato in forma classica ha uno spazio delle soluzioni che corrisponde ad un politopo avente come numero di vertici il numero di variabili/dimensioni del piano. Il politopo, avente un numero di vertici pari al numero di dimensioni dello spazio, viene chiamato **simplesso**. Il nome dell'algoritmo viene quindi dalla rappresentazione geometrica dello spazio delle soluzioni 'fattibili' di un problema in forma classica.

Gli step necessari alla risoluzione di un problema di ottimizzazione rappresentato in forma classica, a mezzo dell'algoritmo del simples-

so[20] sono:

1. L'aggiornamento di due matrici disgiunte B e D la cui unione corrisponda alla matrice A, tramite un operazione chiamata **pivoting** delle variabili.[1];
2. Il calcolo di una soluzione: $\left\{\vec{x}\middle| \vec{x_B} = B^{-1} \cdot \vec{b^T} \wedge \vec{x_D} = \vec{0}\right\}$;
3. Il calcolo dei costi ridotti: $\vec{r} = \vec{c} - \vec{c_B^T} \cdot B^{-1} \cdot \vec{a}$ ;
4. La verifica di ottimalità della soluzione corrente: $\vec{r} \geq \vec{0}$;
5. L'aggiornamento della soluzione ottima alla soluzione corrente se la verifica ha successo altrimenti si ritorna al punto 1[2];

[1]Inizialmente si selezionano gli indici delle colonne arbitrariamente (tecniche più accurate influenzano la velocità di convergenza delle soluzioni ma non impattano le soluzioni stesse)

[2]Il criterio di pivoting successivamente alla prima iterazione per migliorare la velocità di convergenza dell'algoritmo può essere la sostituzione di una variabile di $x_B$ a costo ridotto negativo con una variabile di $x_D$

## 2.2 La programmazione lineare intera

La programmazione lineare intera si occupa di problemi in cui le soluzioni sono descrivibili tramite variabili contenenti numeri interi. Per la risoluzione di un problema di ottimizzazione con variabili intere si adoperano metodi esatti e metodi euristici. I **metodi esatti** sono metodi che restituiscono dato un problema di ottimizzazione, una soluzione ottima globalmente.

I **metodi euristici** invece sono metodi che forniscono una soluzione ad un problema di ottimizzazione che approssima il valore obbiettivo di una soluzione ottima senza garanzia di ottimalità ma richiede molto meno tempo per la risoluzione.

Esiste inoltre tra i metodi euristici una categoria di metodi chiamata **meta-euristiche** che hanno come logica, quella di trovare una soluzione approssimata all'ottimo globale ma seguono 'strategie' in grado di esaminare lo spazio delle soluzioni di un problema in maniera efficiente di modo da non esaminare solamente un sottoinsieme di soluzioni che si somigliano molto tra loro. Recentemente si è riscontrato un utilizzo di queste tecniche in maniera più intensiva. Si nota come le tecniche di quest'ultima categoria siano frequentemente ispirate a meccanismi comunemente trovati in natura.

## 2.3 Metodi esatti

### 2.3.1 Branch and bound

Il metodo di **Branch and bound** è sicuramente una delle tecniche più conosciute tra i metodi esatti usati nella programmazione lineare intera. Il Branch and Bound venne introdotto nel 1960 da Land, A. H. e Doig A. G., nell'aritcolo 'An automatic Method of Solving Discrete Programming Problems'[7]. Il metodo consiste nella rappresentazione, sotto forma di albero composto da un nodo biforcato ricorsivamente, della soluzione ottima globalmente di un problema e nella **ramificazione** di quest'ultimo in soluzioni a sotto-problemi dello stesso. La costruzione delle ramificazioni di una soluzione viene chiamata fase di branch dell'algoritmo.
Il criterio secondo il quale una soluzione genera il livello sottostante contenente le ramificazioni è quello del **rilassamento** del problema. Il rilassamento può avvenire modificando il problema di partenza in diversi modi:

- L'eliminazione di uno o più vincoli del problema di partenza;
- La modifica dell'obbiettivo del problema;
- La sostituzione dell'obbiettivo originale tramite una combinazione lineare dei vincoli con l'obbiettivo stesso, detto rilassamento lagrangiano;
- La sostituzione di uno o più vincoli con una combinazione lineare degli stessi, detto rilassamento surrogato;

A seguire una fase di ramificazione nell'algoritmo di branch and bound, è appunto una fase di bound o **limitazione**. La limitazione consiste nel calcolo del miglior valore possibile dell'obbiettivo di un problema generato tramite ramificazione. Lo scopo di questa operazione è il ridurre il numero di iterazioni dell'algoritmo definendo quali problemi

non vanno ramificati ulteriormente in quanto non producono e le loro ramificazioni non produrranno valori migliori di quello attuale. La limitazione avviene tramite il calcolo di un **limite** per ogni sottoproblema, corrispondente ad un limite superiore nei problemi in cui l'obbiettivo è una massimizzazione o ad un limite inferiore in caso contrario.

### 2.3.2 Branch and cut

La tecnica del branch and cut[14] fu introdotta nel 1991 in relazione al tentativo di ottimizzare i metodi di risoluzione del travel salesman problem. La tecnica del branch and cut è costituita da un implementazione di base del branch and bound seguita dall'utilizzo di tagli (da questi il nome). I tagli consistono nell'aggiunta di vincoli che restringono lo spazio delle soluzioni. Un taglio viene definito tramite il concetto di copertura delle variabili, infatti per la computazione di un taglio, data la funzione obbiettivo ed un vincolo si individua un sottoinsieme delle variabili del problema che ecceda il vincolo se le variabili binarie in questione sono settate ad 1 e si impone un vincolo aggiuntivo al problema imponendo che non tutte le variabili che farebbero eccedere il vincolo devono essere scelte. Questa tecnica seguita dalle normali fasi di branch and bound permette di ottimizzare l'algoritmo stesso eliminando le fasi di valutazione di soluzioni che risulterebbero per definizione non risolventi il problema.

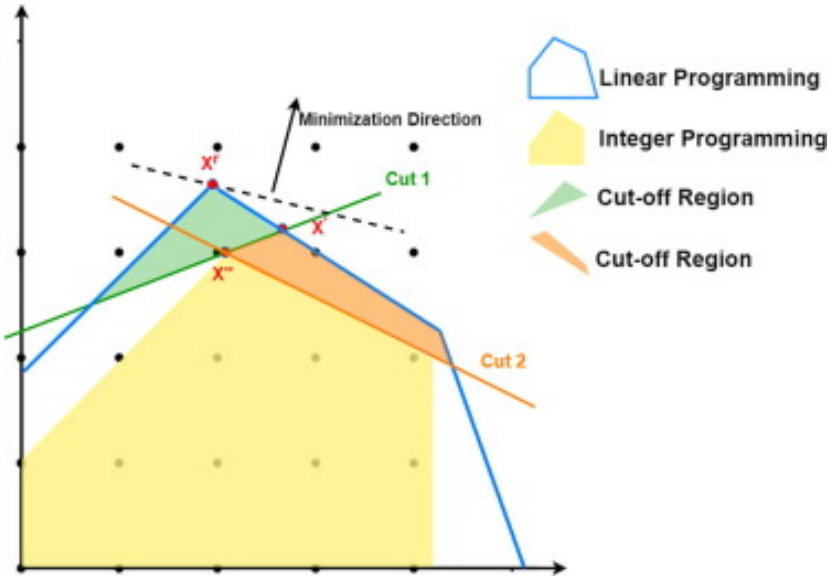


Figura 3: Una rappresentazione dello spazio delle soluzioni di un problema generico all'applicazione dei tagli

### 2.3.3 Programmazione dinamica

Metodo generico per la risoluzione di problemi inventato da Richard Bellman e esposto per la prima volta nel 1957 con l'articolo 'Dynamic Programming'[2]. Consiste di un processo in 3 fasi nel quale si operano:

1. La scomposizione del problema in problemi più piccoli.

2. La risoluzione dei problemi generati al punto precedente se scomponibili anch'essi si ritorna al punto 1.

3. La ricomposizione della soluzione finale tramite la definizione di una funzione che ricombini le soluzioni dei sottoproblemi risolti precedentemente.

## 2.4 Metodi euristici

### 2.4.1 Beam search

L'algoritmo di beam search[16] è un algoritmo euristico che opera seguendo un approccio euristico. Procede similmente al metodo del branch and bound rappresentando il modello del problema e opera rilassamenti successivi al fine di esplorare diverse soluzioni. Tuttavia questo metodo cerca di ridurre il numero di soluzioni da esplorare tenendo in considerazione per le ramificazioni successive solamente i k nodi migliori (rispetto al valore limite dell'obbiettivo).

### 2.4.2 Local search

L'algoritmo di local search[11] è un algoritmo euristico che permette di ridurre il numero di soluzioni da esaminare concentrando la ricerca della soluzione migliore ad un intorno di una soluzione iniziale. Ad influire sull'efficienza di questo algoritmo sono la scelta della soluzione iniziale, la grandezza del numero di vicini da esaminare ed il numero di iterazioni per le quali ripetere l'operazione di valutazione delle soluzioni vicine. Un'altra ottimizzazione possibile è passare all'iterazione successiva una volta trovata la prima soluzione migliore di quella attuale.

## 2.5 Metodi meta-euristici

### 2.5.1 Iterated local search

L'algoritmo di ricerca locale iterata[15] è un algoritmo che mira a ridurre il numero di iterazioni rispetto ai metodi esatti, basandosi sul concetto di accettabilità del risultato e perturbazione del problema. Concettualmente l'algoritmo seguendo lo stesso approccio dell'algoritmo di local search valuta le soluzioni del problema al suo stato corrente fino a quando un criterio specificato in partenza non determina che la soluzione è accettabile, nel caso in cui, invece, il problema non possiede soluzioni che soddisfino l'accettabilità effettua una perturbazione, ovvero itera nuovamente ma a partire da una modifica della soluzione con valore obbiettivo migliore, trovata nell'iterazione precedente. Affinchè l'algoritmo non si stabilizzi nella ricerca su un gruppo di soluzioni simili tra loro la scelta del tipo di perturbazione da effettuare influisce largamente sull'ottimalità della soluzione trovata. Un'altra tipologia di problema in cui è possibile incorrere è data dalla scelta di tecniche di perturbazione 'troppo forti' ossia che a partire da una soluzione già non ottima si allontanano eccessivamente dall'ottimo globale.

### 2.5.2 Variable neighborhood search

L'algoritmo VNS (Variable Neighbourhood Search)[8] ha l'obiettivo di migliorare progressivamente la qualità delle soluzioni esplorando vicinati di ampiezza variabile attorno a una soluzione corrente. Per vicinato si intende linsieme delle soluzioni ottenibili applicando una determinata trasformazione alla soluzione attuale; vicinati diversi sono definiti da trasformazioni differenti, ciascuna caratterizzata da un diverso grado di modifica, che produce soluzioni progressivamente più distanti rispetto a quella di partenza. A ogni iterazione, lalgoritmo applica una fase detta shaking, in cui viene generata in modo casuale una nuova soluzione allinterno del vicinato corrente; questa serve a perturbare la soluzione attuale e avviare lesplorazione in una diversa regione dello spazio delle soluzioni. Successivamente, a partire dalla soluzione perturbata, viene eseguita una ricerca locale per individuare un minimo locale. Se la ricerca locale non porta a un miglioramento, lalgoritmo passa a un vicinato successivo, più ampio o strutturalmente differente, per aumentare la diversificazione della ricerca e tentare di uscire da minimi locali. Se viene trovata una soluzione migliorativa, questa diventa la nuova soluzione corrente e si riparte dal primo vicinato. Lalgoritmo termina quando viene soddisfatto un criterio di arresto, che può consistere in un numero massimo di iterazioni, un limite di tempo, o nellassenza di miglioramenti per un numero predefinito di passaggi consecutivi.

Il Variable Neighborhood Descent è una tecnica simile all'algoritmo trattato di local search questo algoritmo riduce il numero di controlli sulle soluzioni possibili limitandosi all'esaminazione delle soluzioni vicine ad una soluzione che migliora il valore obbiettivo già trovato, tuttavia a differenza del local search richiede comunque l'analisi di tutte le soluzioni vicine e migliori della soluzione dell'iterazione attuale. Questo modo di operare la ricerca delle soluzioni risulta essere ottimo nel evitare soluzioni minime localmente e quindi rendere più efficiente

la ricerca di una soluzione che si avvicini all'ottimo. Tuttavia come molte soluzioni euristiche la vicinanza all'ottimo globale è dipesa dalla tipologia del problema, nello specifico nella disposizione delle soluzioni ritenute ottime localmente rispetto all'ottimo globale.

### 2.5.3 Simulated annealing

La tecnica del simulated annealing[12] come già accennato emula il processo naturale del raffreddamento dei metalli per regolare la 'malleabilità' del algoritmo. Con questo termine si intende quanto l'algoritmo sia in grado di esaminare insiemi di soluzioni 'distanti' nello spazio delle stesse dalla soluzione attuale. L'algoritmo regola questa malleabilità tramite una funzione monotona decrescente chiamata funzione di Boltzmann. Se l'algoritmo si trova alle sue prime iterazioni esso è in grado di esaminare soluzioni con valori della funzione obbiettivo molto distanti dall'ottimo attuale mentre procedento nelle iterazioni fino alle ultime restringe le soluzioni esaminabili a quelle che migliorano o peggiorano la soluzione attuale di valori poco distanti. I parametri cruciali nel funzionamento di questo metodo sono:

- La scelta di una funzione di boltzmann adatta al numero di iterazioni massime desiderate e che fornisca un tempo di raffreddamento necessario ad esaminare un buon numero di soluzioni.
- La scelta di una temperatura iniziale, il grado di flessibilità che si vuole concedere inizialmente all'algoritmo.
- La scelta di una soluzione iniziale, rappresenta l'origine dalla quale l'algoritmo si estende alle soluzioni vicine.

Rispetto alle altre euristiche questo metodo permette di scorrelare, secondo il grado desiderato, la distanza della soluzione ottima globalmente dalle soluzioni ottime localmente.

### 2.5.4 Algoritmi genetici

Gli algoritmi genetici sono una famiglia di algoritmi basati sui concetti naturali della ricombinazione genetica, della mutazione, dell'ereditarietà e della selezione naturale. L'idea di base di questo metodo è di mappare le componenti delle soluzioni (i valori specifici assegnati alle variabili) nell'analogo dei geni e le soluzioni nei cromosomi. All'avanzare nell'iterazione dell'algoritmo corrisponde l'avanzamento ad una generazione di nuove soluzioni 'figlie' la cui formazione avviene tramite ricombinazione dei geni delle soluzioni 'genitrici'. La scelta dei candidati genitori ad ogni generazione avviene secondo la valutazione della bontà delle soluzioni tramite una funzione chiamata fitness. Il metodo in cui i geni vengono ricombinati per la formazione delle soluzioni figlie è deciso a priori all'inizio dell'algoritmo e allo stesso modo la probabilità che avvengano mutazioni in una nuova generazione. Tra le tecniche più comuni per la ricombinazione dei geni abbiamo il 'crossover'. Con l'arrivo di una nuova generazione le soluzioni presenti in precedenza vengono sostituite parzialmente o totalmente secondo il valore minimo di fitness desiderato. La terminazione dell'algoritmo può avvenire in due casi:

- Il numero di iterazioni massime, numero di generazioni prese in esame, è stato raggiunto.
- La popolazione delle soluzioni nella generazione corrente contiene una soluzione il cui valore di fitness è quello desiderato.

I parametri che influenzano quindi la bontà delle soluzioni restituite e la velocità di convergenza dell'algoritmo sono:

- Il numero di generazioni/iterazioni massimo che l'algoritmo può eseguire.
- Il numero di soluzioni che 'sopravvivono' tra una generazione e l'altra.

- La funzione di fitness che valuta la bontà delle soluzioni di una generazione.
- La metodologia di ricombinazione dei geni delle soluzioni scelte per generare le soluzioni della generazione successiva.
- La probabilità di mutazione dei geni durante la ricombinazione.

## 2.6 Teoria dei grafi

La teoria dei grafi nasce come disciplina matematica a seguito dello studio del problema dei sette ponti di Königsberg. Il problema fu formulato nel 1800 dai cittadini della città stessa, che si trovava topologicamente connessa da sette ponti in quanto attraversata da un fiume, e consisteva nella ricerca di un percorso unico che permettesse di visitare la città in un unico percorso che infine ritornasse al punto di partenza. Per risolvere il problema **Leonard Euler** utilizzò per la prima volta una rappresentazione sotto forma di grafo e perciò è a lui attribuita il primo utilizzo di questa tecnica di rappresentazione. La teoria fu elaborata nel 1736 e poi pubblicata nel 1741. Euler concluse che non esisteva un percorso ciclico che tornasse al punto iniziale senza ritornare su una strada già percorsa utilizzando quella che oggi definiamo teoria dei grafi. La teoria dei grafi è una disciplina scientifica che si occupa dello studio e della rappresentazione di problemi che richiedono delle relazioni tra entità. Alla base della teoria dei grafi vi è una rappresentazione delle entità tramite elementi chiamati **nodi** o **vertici** e delle relazioni tra esse tramite linee dette **spigoli** o frecce dette **archi**. La tipologia delle relazioni in un grafo determina ulteriori distinzioni tra i grafi stessi. Se la relazione tra due nodi è quindi rappresentata tramite spigoli si dice non orientata se tramite archi, orientata.

Quando una relazione che occorre tra i nodi di uno stesso grafo avviene con variazioni di coppia in coppia di un valore si rappresenta sul grafo un **peso** per disinguerle. Un peso consiste in un numero che sovrasta la relazione indicandone una capacità, intensità o misura di altro genere. Uno spigolo o un arco può essere quindi pesato o non pesato per distinguere la relazione che occorre tra due nodi distinti di uno stesso grafo. Due nodi che posseggono uno spigolo o un arco che li congiunge sono **connessi**.

Un grafo in base alle relazioni tra i suoi nodi può essere detto **orientato** se ogni relazione tra due nodi è rappresentata con un arco e non orientato altrimenti. Un grafo **parzialmente orientato** è un grafo in cui la relazione tra i nodi a volte è rappresentata tramite spigoli e altre volte tramite archi. Altre definizioni di base relative agli elementi interni ad un grafo sono:

- Coppie di spigoli che terminano in uno stesso nodo sono dette adiacenti, lo stesso vale per gli archi distinti che hanno rispettivamente nodo sorgente e nodo destinazione in comune;
- Insiemi di spigoli adiacenti formano un **cammino**;
- Un cammino che termina nel nodo in cui è iniziato è detto **ciclo**;

Particolari tipologie di cammini e cicli sono quelli hamiltoniani e euleriani i quali sono rispettivamente:

- **hamiltoniani** se attraversano tutti i nodi del grafo una ed una sola volta;
- **euleriani** se attraversano tutti gli spigoli/archi del grafo una ed una sola volta;

Un grafo può essere pesato o non pesato se tutti i suoi spigoli/archi posseggono un peso o nessuno di essi ne possiede uno.

### 2.6.1 La topologia di un grafo

La topologia di un grafo descrive il modo in cui i nodi sono collegati tra loro. Un grafo viene detto **connesso** quando per ogni coppia di nodi esiste un percorso che li collega attraverso una sequenza di spigoli. Al contrario, si parla di **sconnesso** se almeno un nodo non è raggiungibile dagli altri.

Nel caso in cui ogni coppia di nodi presenti un collegamento diretto, il grafo si definisce **completo**.

Un grafo è considerato **planare** quando può essere rappresentato nel piano senza che i suoi spigoli si intersechino, a meno che non si incontrino in un nodo comune. Se non è possibile trovare una rappresentazione nella quale gli archi giacenti sullo stesso piano non si intersecano, il grafo è detto **non planare**.

Un grafo si definisce **cordale** quando ogni ciclo formato da quattro o più nodi contiene almeno un arco che collega due nodi non consecutivi del ciclo. Questo arco interno prende il nome di corda.

Infine, si definisce **ordine di eliminazione perfetto** un ordinamento dei nodi in cui, per ogni nodo considerato, tutti i suoi vicini successivi nellordine risultano collegati tra loro. O descrivendo l'ordinamento secondo il significato di eliminazione si considera un nodo nell'ordinamento eliminato e durante leliminazione di ciascun nodo, i vicini ancora presenti devono essere connessi tra loro da spigoli. Questa proprietà caratterizza i grafi cordali.

Le definizioni relative alla topologia dei grafi verranno riutilizzate insieme a quelle del capitolo sulla colorazione (2.6.2) nei capitoli dedicati al completamento degli orientamenti (4.5)

### 2.6.2 La colorazione di un grafo

La colorazione dei vertici in un grafo consiste nell'assegnare un colore a ciascun vertice in modo tale che due vertici adiacenti non condividano lo stesso colore. Il numero minimo di colori necessari per ottenere una colorazione valida è detto *numero cromatico* del grafo. Un risultato fondamentale in questo ambito è il **teorema dei quattro colori**, secondo cui ogni grafo planare può essere colorato utilizzando al massimo quattro colori, in modo che nessuna coppia di vertici adiacenti abbia lo stesso colore. La colorazione dei vertici trova applicazioni in diversi contesti:

- **Schedulazione**: i problemi in cui devono essere pianificate delle attività, ogni attività è rappresentata un vertice e un arco tra due vertici indica conflitto. La colorazione rappresenta l'assegnazione di risorse o intervalli di tempo distinti.
- **Sudoku**: il gioco logico può essere rappresentato come un grafo in cui ogni cella è un vertice, e le regole del gioco (righe, colonne e riquadri) definiscono le adiacenze. Una colorazione valida corrisponde a una soluzione corretta del puzzle.

### 2.6.3 L'algoritmo di hierholzer

Citando nuovamente **Leonard Euler** e usando la terminologia appena illustrata, è possibile individuare un percorso euleriano su di un grafo se e solamente se tutti i nodi hanno grado pari o solamente due di essi hanno grado dispari. E' importante notare che se il grafo sul quale viene applicato l'algoritmo ha nodi tutti con grado pari il percorso euleriano trovato corrisponde sempre ad un **ciclo euleriano**. Questa definizione in seguito alla risoluzione del problema dei sette ponti di Königsberg (risultata nell'impossibilità dello stesso), portò nel 1873 **Carl Hierholzer** [10] allo sviluppo di un algoritmo polinomiale esposto nell'articolo 'Sulla possibilità di percorrere un tracciato senza ripetizioni e senza interruzioni' per la ricerca di un ciclo euleriano sui grafi aventi grado pari per ogni nodo.

L'**algoritmo di hierholzer**[3] consiste nei seguenti passaggi:

1. Si sceglie arbitrariamente un qualsiasi vertice iniziale v.
2. Si segue un percorso di archi da quel vertice fino a tornare a v.
3. Fino a quando esiste un vertice u che appartiene al tour corrente ma che ha archi adiacenti non facenti parte del tour, si inizia un altro percorso da u, seguendo gli archi non utilizzati fino a tornare a u, e si unisce il tour cosi' formato in questo modo al tour precedente. Poiché assumiamo che il grafo originale sia **connesso**, ripetendo il passaggio precedente si esauriranno tutti gli archi del grafo.

L'algoritmo di hierholzer ha **complessità** $O(|E|)$ dove E è l'insieme degli spigoli nel grafo.

# 3 Il gioco del domino

## 3.1 Introduzione al gioco

In questo capitolo si spiega come, sia nella generazione degli schemi sia nella loro risoluzione sia possibile e per di più vantaggioso, utilizzare algoritmi e tecniche comuni nella ricerca operativa. Partendo dall'idea di dover rappresentare il set da gioco da utilizzare in una forma matematica, noto come ogni tessera di un set da gioco possa essere vista come uno spigolo in un grafo che abbia per nodi il numero di punti che appiaono nel set da gioco. Per esempio il grafo seguente rappresenta un set da gioco doppio sei:

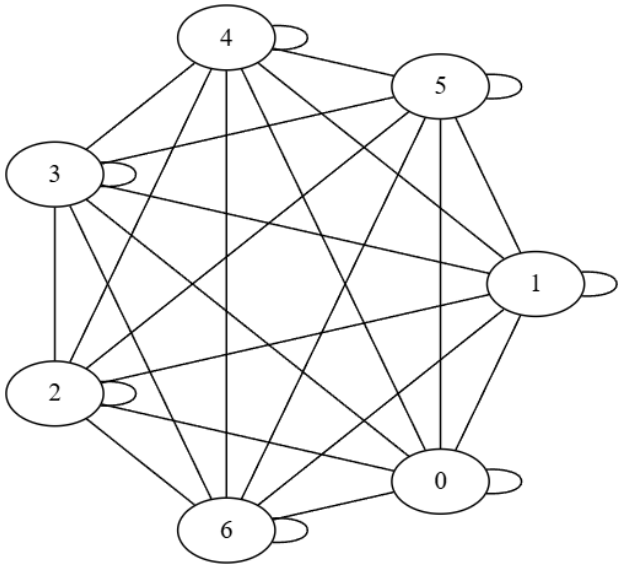


Figura 4: Un grafo corrispondente ad un set doppio-6

Così come dei set da gioco doppio tre e doppio quattro corrispondono rispettivamente a dei grafi così composti:

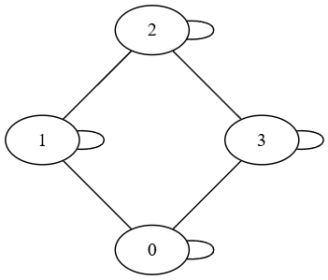


Figura 5: Un grafo corrispondente ad un set doppio-3

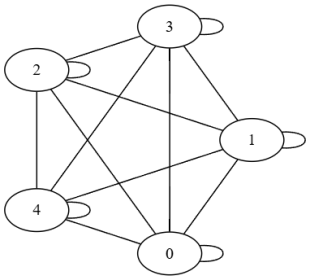


Figura 6: Un grafo corrispondente ad un set doppio-4

Il concetto attorno al quale la teoria del gioco è stata sviluppata, è la rappresentazione di un **set da gioco/tileset** (figura 7) come un grafo regolare a N+1 vertici (da 0 ad N) nel quale le tessere da gioco corrispondono ad ogni spigolo esistente tra i nodi del grafo.

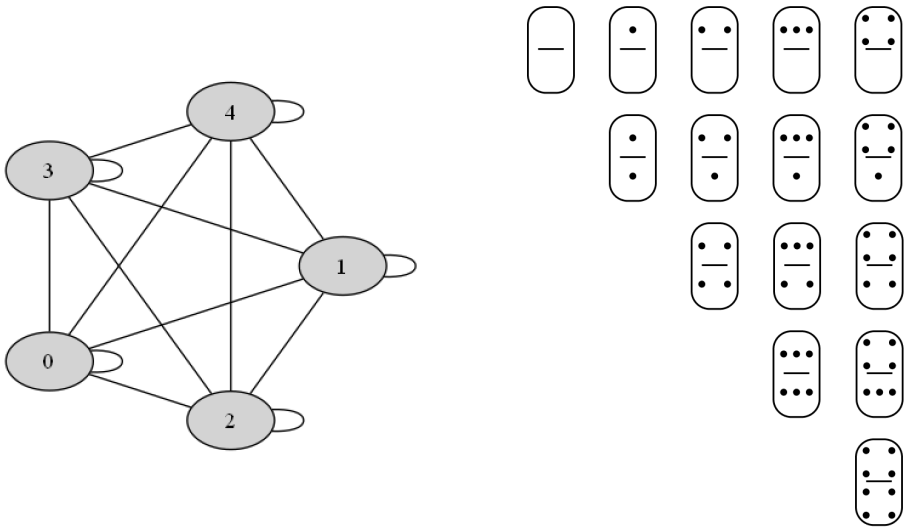


Figura 7: Rappresentazione grafica di un grafo e il corrispondente tileset per N pari a 4

Una **sequenza di tessere** (figura 8) corrisponde invece ad un insieme ordinato di tessere che può essere disposto in maniera sequenziale solo se due tessere hanno delle facce 'uguali' e corrisponde nella teoria dei grafi ad un cammino/ciclo euleriano sul grafo del set da gioco.

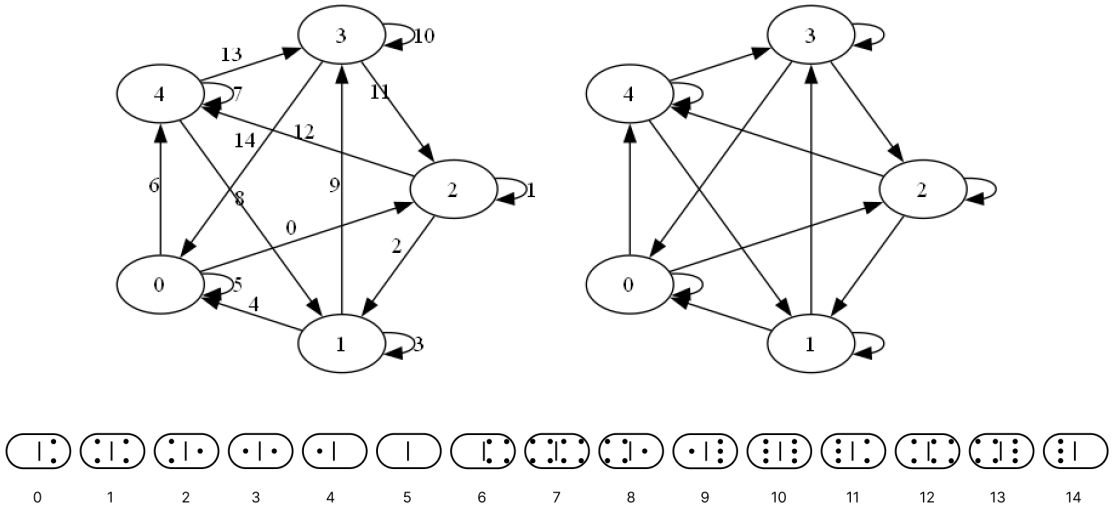


Figura 8: Rappresentazione grafica di un grafo orientato e la corrispondente sequenza per N pari a 4

Occorre infine per definire il gioco individuare il concetto di **puzzle** ovvero una sequenza in cui sono state rimosse alcune tessere e corrisponde ad un orientamento parziale di un sottografo/grafo del grafo rappresentante il set da gioco.

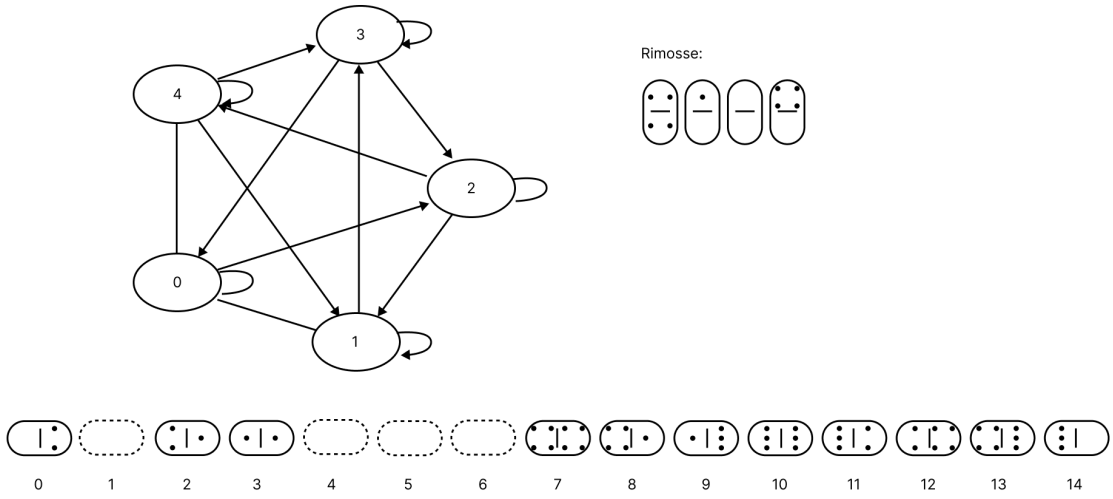


Figura 9: Rappresentazione grafica di un grafo parzialmente orientato e il corrispondente puzzle per N pari a 4

Affinchè si distinguano le due possibilità, avere una sequenza corrispondente ad un ciclo euleriano/hamiltoniano occorre fornire la prima delle proprietà del gioco residente nel teorema di eulero. Ovvero è possibile tracciare un ciclo euleriano su di un grafo se e solo se ogni vertice in esso ha grado pari (il numero di spigoli connessi ad ogni vertice è pari oppure in un grafo orientato la differenza tra archi entranti e uscenti è 0). E' quindi conseguentemente necessario distinguere puzzle creati con set da gioco ad N pari o dispari assumendo che tutte le sequenze di tessere derivate da un set da gioco che ha N dispari potranno rappresentare effettivamente una sequenza equivalente ad un ciclo euleriano sul grafo del set da gioco mentre set da gioco con N pari permetteranno di generare sequenze in cui si ha un ciclo euleriano e multipli cicli hamiltoniani oppure nel caso di grafi planari un ciclo euleriano che corrisponde esattamente ad un solo ciclo hamiltoniano.

## 3.2 Considerazioni relative all'uso di tessere doppie

Nel gioco del domino tradizionalmente i set da gioco contengono due tipologie di tessere, quelle raffiguranti su entrambe le facce lo stesso numero chiamate tessere doppie e tessere raffiguranti due numeri distinti. In funzione del gioco un puzzle può essere generato e risolto sia con la presenza di tessere doppie che senza, nelle teorie esposte nei prossimi capitoli per spiegare il funzionamento del gioco e le tecniche ad esso afferenti si assume usando i set da gioco presenti in commercio di utilizzare anche questa tipologia di tessere. Nella rappresentazione dei set da gioco sotto forma di grafi queste tessere corrispondono a degli auto anelli ovvero degli spigoli da un nodo verso se stesso, sebbene questi possano sembrare ridondanti servono ad avere una rappresentazione realistica delle risorse realmente disponibili nel gioco. Come verrà spiegato in seguito il paragone tra il grafo ed il set da gioco non si ferma qui, bensì lo schema iniziale di una partita di domino puzzle corrisponde ad un grafo parzialmente orientato in cui si conosce l'orientamento e la posizione di alcuni archi all'interno del ciclo euleriano che giace sul grafo e la soluzione di un puzzle corrisponde ad un orientamento completo di tutti gli spigoli di uno schema in modo che essi formino un ciclo euleriano. Ne consegue che dal punto di vista della rappresentazione del set da gioco l'orientamento di una tessera che va da un nodo a se stesso non abbia realmente un orientamento ma va considerata in quanto tessera presente nel gioco originale.

## 3.3 La generazione di sequenze con set da gioco dispari

Se si prova a costruire il grafo corrispondente ad un set da gioco del domino che arriva fino ad un numero di punti dispari ci si accorge come non sia possibile giocare con tutte le tessere in quanto per comporre un unica sequenza che include tutte le tessere il grado di ogni nodo deve essere pari.
Sebbene questa affermazione possa non essere di immediata comprensione, è effettivamente vera quando si pensa all'analogia tra il grafo rappresentante il set da gioco e l'obbiettivo della formazione di uno schema di gioco.
Una sequenza di tessere adiacenti tra loro corrisponde infatti su di un grafo ad un ciclo euleriano. Un ciclo euleriano è un cammino su di un grafo, comprendente tutti gli spigoli/archi dello stesso, che parte da un nodo e termina in esso, il concetto è stato introdotto nel paragrafo 2.6. L'esistenza di un ciclo euleriano su di un grafo orientato dipende dal grado dei nodi del grafo. Così come stabilito da Eulero nella sua dimostrazione dell'irrisolvibilità nel problema dei ponti di Königsberg. Non è possibile trovare un ciclo euleriano su di un grafo in cui un nodo abbia grado dispari.
Analogamente non è possibile generare una sequenza di tessere adiacenti, usando tutte le tessere di un set da gioco, se prima non ci si assicura che nel set da gioco per ogni numero di punti il numero di tessere su cui il numero di punti appare sia pari. La soluzione adoperata in questi casi per lo sviluppo del gioco è quella di adoperare il maggior numero di tessere possibili del set da gioco rimuovendo quelle che collegano vertici il cui valore assoluto della differenza tra i valori dei vertici equivale metà del valore massimo di punti.
Avremo quindi per set da gioco a numeri dispari che il grafo corrispondente prima e dopo la rimozione sia il seguente:

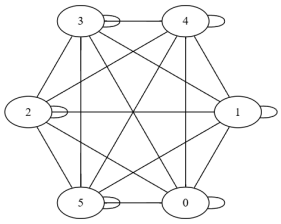


Figura 10: Un grafo corrispondente ad un set da gioco doppio-5 dispari prima della rimozione

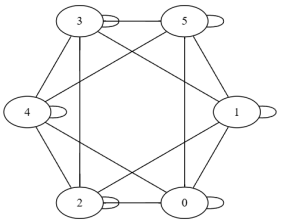


Figura 11: Un grafo corrispondente ad un set da gioco doppio-5 dispari dopo la rimozione

L'utilizzo di un sottoinsieme delle tessere del set da gioco al fine di poter individuare un ciclo euleriano corrisponderà per un set da gioco doppio-5 all'utilizzo delle tessere colorate nell'immagine seguente:

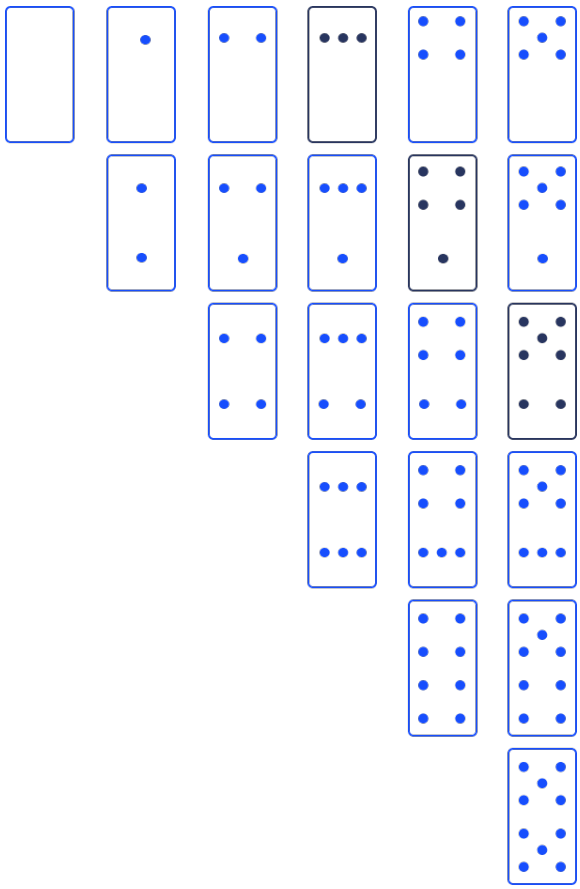

Figura 12: Un set da gioco doppio-5 dispari (che include le tessere doppie) dopo la rimozione

Un esempio di sequenza generata a partire da questo set da gioco ‘ridotto‘:

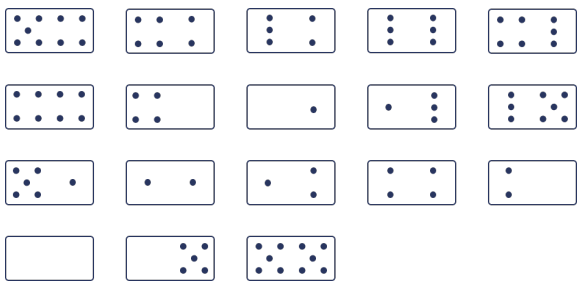

Figura 13: Una sequenza generata usando un set da gioco doppio-5 dispari (che include le tessere doppie) dopo la rimozione

## 3.4 Letteratura scientifica legata al gioco del domino

L'impiego della teoria dei grafi e delle tecniche di ricerca operativa al gioco del domino è storicamente documentato nella letteratura scientifica. Si riscontra l'interesse generale degli accademici verso lo studio dei comportamenti, delle strategie competitive e degli equilibri della teoria dei giochi attraverso diverse pubblicazioni riguardanti gli aspetti computazionali dei giochi e le loro somiglianze con problemi storici della ricerca operativa.

Tra le definizioni più illuminanti riguardo alle relazioni tra i giochi e le teorie computazionali/ strategie di ottimizzazione abbiamo la pubblicazione del testo 'Games, Puzzles, and Computation' da parte di **Hearn R. e Demaine E.D.**[9] nella quale l'approccio agnostico definisce dall'inizio cosa sia un gioco differenziando nei capitoli 2/3 varie tipologie di giochi secondo parametri come il numero dei giocatori, obbiettivo del gioco, tipo di interazioni tra lo stato del gioco e le 'mosse dei giocatori' e infine le posizioni dei giocatori da intendersi come tipologie di schieramenti, ovvero gioco solitario, gioco di squadra e tutti contro tutti. Nel testo si includono nella definizione di gioco tutti gli eventi, anche non a scopi ludici, che possono essere rappresentati secondo i parametri di cui sopra, ad esempio si parla di come i comportamenti di individui in ambito economico possano anch'essi essere considerati giochi, riconoscendo come già la celebre definizione fosse stata data da Von Neumann J. e Nash J. Per di più si evidenzia come i famosi equilibri di Nash secondo i quali qualsiasi sia la strategia dei giocatori se i loro obbiettivi convergono esisterà uno stato di equilibrio.

**Billaut J.-C., Della Croce F.** spiegano nell'articolo 'No-idle, no-wait: when shop scheduling meets dominoes, eulerian and hamiltonian paths'[4], come sia possibile applicare la teoria dei grafi, percorsi eule-

riani ed hamiltoniani sia alla formazione di sequenze di domino che allo scheduling di attività generiche. In particolare nell'articolo, viene paragonato lo scheduling di attività senza pause nè attese su 2 macchine e come vi sia una corrispondenza tra l'algoritmo di scheduling da loro adottato e l'algoritmo utilizzato comunemente per la generazione di puzzle di domino. Infine stabiliscono come questa tecnica risulti in un miglioramento netto rispetto alle tecniche conosciute per la risoluzione del problema fornendo un metodo esatto e con una classificazione della complessità presentata pari a O(n).

Si evidenzia come vi siano diverse publlicazioni relative al gioco del domino, come 'Playing Dominoes Is Hard, Except by Yourself' portata a termine da **Demaine E.D., Ma F., Waingarten F.**[6], sviluppata presso il Massachussets Institute of Technology. La ricerca si concentra sulla complessità computazionale di alcune classi di giochi tradizionali (includendo i puzzle) e prova come i puzzle di tipo domino appartengano ad una complessità più semplice (polinomiale nelle dimensioni dell'input), mentre le versioni del domino giocate in squadre o tutti contro tutti provano una complessità computazionale molto maggiore.

# 4 Ricerca operativa applicata al domino

## 4.1 Applicazioni della programmazione lineare intera al domino

E' possibile risolvere e costruire dei puzzle del domino tramite la PLI e la teoria dei grafi. Si può identificare come all'interno di un gioco di domino si possa scomporre le operazioni necessarie per la formulazione e la risoluzione del gioco in sottoproblemi, quali:

- La **generazione di una soluzione** di partenza;
- La **generazione di uno schema di puzzle** generico;
- La **classificazione** della complessità di uno schema di puzzle;
- La **validazione di uno schema di puzzle**[3];
- La **risoluzione di uno schema di puzzle** alla sua soluzione di partenza;

[3]La verifica che uno schema di puzzle sia risolvibile e la soluzione sia una ed una sola

## 4.2 Generazione di puzzle

Di seguito si presenta l'algoritmo per la creazione di un puzzle composto da tessere del domino, accompagnato dall'analisi della relativa complessità e da un criterio di distinzione basato sulle opzioni di generazione del puzzle.

I puzzle, come illustrato in precedenza, possono appartenere a due categorie: $N$ pari o dispari. Tale distinzione incide sulla lunghezza della sequenza, sul numero di tessere già presenti o da inserire. Inoltre, è necessario specificare la complessità desiderata, indicata in questo capitolo con la lettera $C$.

Nellimplementazione proposta si assume, per semplicità, che la complessità venga fornita sotto forma di categoria (ad esempio: *semplice*, *media*, *difficile*). Si suppone che ciascuna categoria corrisponda univocamente a un intervallo di complessità assoluta, calcolata come descritto nel capitolo 4.4.

Come primo passo, si genera un grafo rappresentante il tileset selezionato, scelto come grafo regolare con $N + 1$ nodi. Successivamente, si utilizza l'algoritmo di Hierholzer, opportunamente adattato per la generazione di sequenze, come riportato di seguito:

**Algorithm 1:** Algoritmo di Hierholzer adattato per la generazione di sequenze

**Data:** $N$
**Result:** Un grafo fortemente connesso e orientato $G(V, A)$

```
1  Si inizializza un grafo P fortemente connesso con N + 1 vertici;
2  Si inizializza uno stack vuoto per i nodi da visitare;
3  Si inizializza un vettore circuito vuoto per i nodi visitati;
4  Si inserisce un vertice arbitrario nello stack (assumendo che i
     vertici siano numerati, se ne sceglie uno tra 0 e N);
5  while lo stack non è vuoto do
6      if l'ultimo vertice nello stack ha nodi adiacenti then
7          Si rimuove lo spigolo che collega il vertice corrente a un
             vertice adiacente arbitrario da P;
8          Si aggiunge il vertice adiacente allo stack;
9      else
10         Si aggiunge l'ultimo vertice dello stack al circuito;
11         Si orienta uno spigolo secondo l'ordine degli ultimi due
             elementi nel circuito;
12 return Il grafo P
```

Una volta ottenuta una sequenza, la generazione del puzzle si riconduce alla rimozione controllata di tessere, secondo la procedura seguente:

1. Si rimuove una tessera in maniera arbitraria dalla sequenza.
2. Si applica il modello PL; se la soluzione ottenuta coincide con la sequenza originaria, il puzzle viene considerato valido.
3. Si valuta la complessità attuale del puzzle tramite l'algoritmo descritto nel capitolo 4.4; il processo si ripete fino al raggiungimento della complessità desiderata.

4. Vengono reinserite le tessere singole.

Il modello PL pensato per la risoluzione del problema è il seguente:

Funzione obiettivo:

$$\min y \tag{1}$$

Vincoli:

$$y = 0$$

$$\forall i \quad \sum_{j} x_{i,j} = 1 \tag{2}$$

$$\forall j \quad \sum_{i} x_{i,j} = 1 \tag{3}$$

$$\forall j < \text{len} \quad x_{i,j} \leq \sum_{i' \text{ adiacente a } i} x_{i',j+1} \tag{4}$$

$$\forall \text{i,j} \mid x_{i,j} \in \text{puzzle} \quad x_{i,j} = 1 \tag{5}$$

Variabili:

$$y \in \{0,1\}, \quad x_{i,j} \in \{0,1\} \tag{6}$$

La funzione obiettivo (7) è fittizia, con valore costante $y = 0$, poiché il modello non mira a ottimizzare una quantità specifica, ma piuttosto a trovare una configurazione che soddisfi tutti i vincoli del problema.

Le variabili binarie $x_{i,j}$ rappresentano la scelta di posizionare la tessera $i$-esima del set nella posizione $j$-esima della sequenza finale.

- **Vincoli di utilizzo delle tessere**(8): ogni tessera deve essere usata una sola volta, quindi per ogni $i$, la somma delle assegnazioni alle posizioni $j$ deve essere pari a 1:

$$\forall i \quad \sum_{j} x_{i,j} = 1$$

- **Vincoli di copertura delle posizioni**(9): ogni posizione della sequenza deve essere occupata da una tessera:

$$\forall j \quad \sum_{i} x_{i,j} = 1$$

- **Vincoli di adiacenza**(10): se una tessera $i$ è posizionata nella posizione $j$, allora nella posizione successiva $j+1$ deve trovarsi una tessera adiacente a $i$:

$$\forall j < \text{len} \quad x_{i,j} \leq \sum_{i' \text{ adiacente a } i} x_{i',j+1}$$

## 4.3 Considerazioni sulla validità di un puzzle

Con il termine validità di un puzzle si allude alla proprietà di uno schema di puzzle di essere completabile e avere una ed una sola soluzione. La validità di un puzzle può essere verificata attraverso la **Programmazione Lineare**: si procede risolvendolo e controllando che la soluzione ottenuta coincida con quella iniziale.
Poiché il modello utilizzato per testare la validità è lo stesso impiegato per risolvere il puzzle, risulta interessante analizzare fino a che punto sia possibile rimuovere tessere dallo schema iniziale senza compromettere la validità del puzzle stesso.
Il modello utilizzato per la validazione con la PL ha le stesse variabili $x_{i,j}$ indicate nel modello per la generazione, la funzione obbiettivo sta volta corrisponde alla minimizzazione delle variabili corrispondenti alle tessere inserite nelle posizioni vuote del puzzle contenute nella soluzione iniziale:

Funzione obiettivo:

$$\min \sum_{i',j' | \text{tessere presenti nella soluzione ma non nel puzzle}} x_{i',j'} \tag{7}$$

Vincoli:

$$\forall i \quad \sum_{j} x_{i,j} = 1 \tag{8}$$

$$\forall j \quad \sum_{i} x_{i,j} = 1 \tag{9}$$

$$\forall j < \text{len} \quad x_{i,j} \leq \sum_{i' \text{ adiacente a } i} x_{i',j+1} \tag{10}$$

$$\forall \text{i,j} \mid x_{i,j} \in \text{puzzle} \quad x_{i,j} = 1 \tag{11}$$

Variabili:

$$y \in \{0,1\}, \quad x_{i,j} \in \{0,1\} \tag{12}$$

Questa analisi fornisce un'indicazione sui limiti della **complessità assoluta** del puzzle, discussi nel dettaglio nel Capitolo 4.4.
Assumendo che i tileset da gioco prevedano l'uso di tessere doppie, per garantire lunicità della soluzione si richiede che tali tessere siano già presenti nello schema iniziale. Consideriamo ora un grafo regolare rappresentante un set da gioco, assieme a un orientamento parziale del grafo con annotazione delle posizioni in cui ogni arco appare all'interno del puzzle.
La validità del puzzle è garantita se:

> Tutti i cicli hamiltoniani disgiunti presenti nel grafo con orientamento completato contengono almeno una tessera per ciclo inserita nello schema.

In altri termini, nel parallelismo con un grafo regolare parzialmente orientato, è sufficiente che almeno un arco appartenente a ciascun ciclo hamiltoniano disgiunto sia stato orientato. Nel caso di un **grafo planare** euleriano (cioè ogni nodo ha grado pari), è sufficiente che anche uno solo dei suoi spigoli sia orientato affinché sia possibile completarne lorientamento (e quindi risolvere il puzzle). Per un **grafo euleriano non planare**, è comunque possibile scomporlo in grafi planari che:

- non condividono spigoli tra loro, e
- possono essere resi validi orientando almeno uno spigolo ciascuno.

Da qui si ricava la seguente **regola di validità**:

> Dato un grafo sottostante con $n$ vertici (cioè un set da gioco doppio-$n$) e tutti i nodi di grado pari, il numero minimo di tessere da avere nello schema è:
>
> $$\left\lfloor \frac{n}{2} \right\rfloor$$

più le tessere doppie, che non sono comprese nei grafi scomposti.

Premesso che i **grafi planari** sono quelli rappresentabili senza che gli archi si intersechino tra loro, le seguenti figure mostrano esempi di grafi **non planari** e la loro scomposizione in **cicli hamiltoniani** allinterno di sottografi planari distinti e disgiunti:

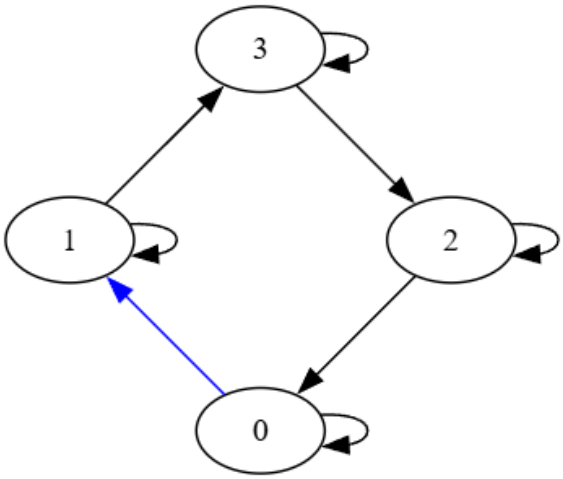


Figura 14: Un grafo planare parzialmente orientato rappresentante un set da gioco doppio-3. In blu: lorientamento di una tessera implica la determinazione dellintero grafo.

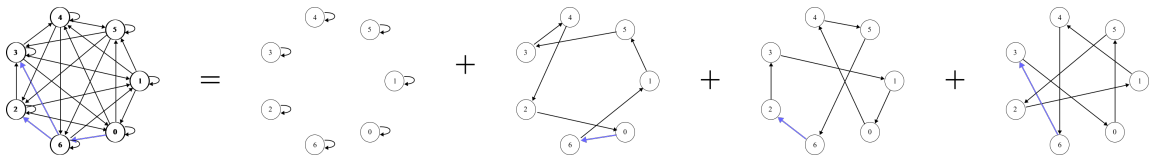


Figura 15: Un grafo non planare parzialmente orientato rappresentante un set da gioco doppio-6. È mostrata la sua scomposizione in cicli hamiltoniani. In blu: alcuni archi che permettono di determinare un orientamento univoco per tutto il grafo.

## 4.4 Considerazioni relative alla complessità del gioco

La complessità di un puzzle è l'altra caratteristica da tenere in considerazione per la costruzione di un puzzle. Dato un puzzle che deve essere comunque valido per poter essere risolto, la complessità per un utente ha indubbiamente una proporzionalità diretta con il numero di tessere rimosse in una sequenza il quale corrisponde per un puzzle valido ad un numero che varia tra 1 e la differenza della lunghezza della sequenza e il numero minimo di tessere del set per la validità $\lfloor n/2 \rfloor$.

Data quindi un'indicazione sul numero di tessere rimosse massimo e minimo, occorre descrivere come la rimozione di una tessera non sia fine a se stessa ma collegata alla presenza di tessere simili, dalla quale una maggiore complessità pratica scaturisce. Riguardo a quest'ultima osservazione bisogna ricordare che nella sequenza le tessere 'simili' si trovano in coppie adiacenti in posizioni diverse della sequenza. Quindi mettendo insieme i due parametri, il numero di tessere rimosse e la relazione tra le tessere rimosse, stabiliamo come la difficoltà di un puzzle per il giocatore sia numericamente quantificabile in un'unica misura. Definiamo come misura C la complessità massima di un puzzle costruito a partire da un set da gioco doppio-N.

Data quindi come definizione C la complessità massima,
essa è direttamente proporzionale al numero di tessere rimosse e al possibile reinserimento di una tessera in piu' di una posizione. Mentre la misura del numero di tessere rimosse appare abbastanza intuitiva, la misura del possibile reinserimento di una tessera in piu' posizioni è a priori sconosciuta e ne possiamo ipotizzare solamente valori massimi e minimi. Infatti il numero di posizioni disponibili in cui inserire una tessera e' al massimo pari al numero di tessere rimosse ed al minimo 1. In piu' la sparsità delle posizioni libere nello schema gioca un ruolo

importante nel possibile reinserimento di una tessera.
Questa proprietà è facilmente osservabile se si osservano due casi limite di due puzzles generati usando lo stesso set da gioco, aventi lo stesso numero di tessere rimosse, ma aventi l'uno le tessere rimosse sempre intervallate da una tessera invece presente e l'altro avente tutte le tessere rimosse contigue. Nel primo puzzle la difficoltà nel reinserimento delle tessere è nulla in quanto le tessere 'vicine' indicano chiaramente il numero di punti che le due facce della tessera mancante deve avere nell'altro invece si individuano in partenza diverse posizioni disponibili per ogni tessera ma una sola sequenza risolutiva che posiziona tutte le tessere.

Figura 16: Un puzzle costruito con un set da gioco doppio 3 avente 4 tessere mancanti in maniera intervallata

Figura 17: Un puzzle costruito con un set da gioco doppio 3 avente 4 tessere mancanti in maniera contigua

Definendo un buco come una sequenza contigua di tessere mancanti e l'insieme dei buchi in un puzzle H. Dati L lunghezza dello schema e $H_{MAX}$ la lunghezza del buco massimo possibile:

$$L = \begin{cases} \frac{(N+1)(N+2)}{2} & \text{se } N \text{ è pari} \\ \frac{(N+1)^2}{2} & \text{se } N \text{ è dispari} \end{cases} \tag{13}$$

$$H_{MAX} = L - \lfloor N/2 \rfloor \tag{14}$$

Possiamo formulare la complessità di un puzzle come:

$$C = \frac{1}{(|H|)^{0.1}} \sum_{i=1}^{|H|} \left( \frac{|H_i|}{H_{MAX}} \right)^2 \tag{15}$$

La formula precedente è stata scelta per descrivere approssimativamente gli aspetti appena descritti, indicativamente l'esponente di ogni termine della somma è stato scelto a 2 per esasperare l'importanza della lunghezza dei buchi e 0.1 nel fattore 'peso' di ogni addendo per ridurre l'importanza del numero di buchi presenti. In questo modo la formula effettivamente provvede a calcolare un numero sempre compreso tra 0 e 1 che indica il livello di difficoltà del puzzle da completare.

Una volta sviluppate le classificazioni con la formula al fine di poter assegnare un metodo di selezione dei puzzle più intuitivo, sono stati raggruppati in scala esponenziale i puzzle in base alla complessità 'assoluta ricavata':

1. I puzzle che ricadono nei valori di complessità assoluta da 0 a 0.5 sono classificati come puzzle 'facili'.

2. I puzzle che ricadono tra un valore di 0.5 e 0.83 vengono classificati come puzzle di 'media' difficoltà.

3. Infine i puzzle che ricadono nel range superiore a 0.83 vengono classificati come puzzle 'difficili'.

Di seguito si riportano alcuni esempi di schemi generati secondo questa classificazione, dove la lunghezza del set da gioco dipende dal parametro N (numero massimo di punti sulle tessere) e la difficoltà dal parametro C (corrispondente alle categorie facile, mediamente complesso, difficile):

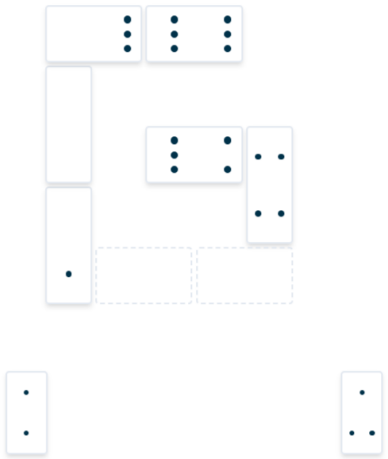

Figura 18: Esempio di puzzle con n=3 e c=1

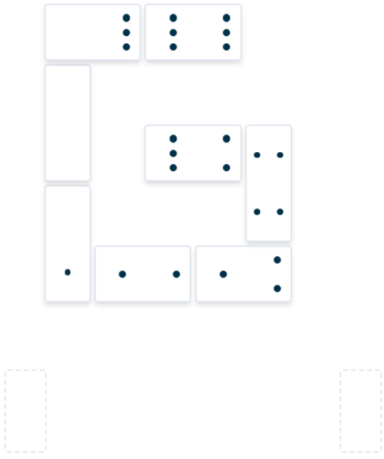

Figura 19: Soluzione del puzzle con n=3 e c=1

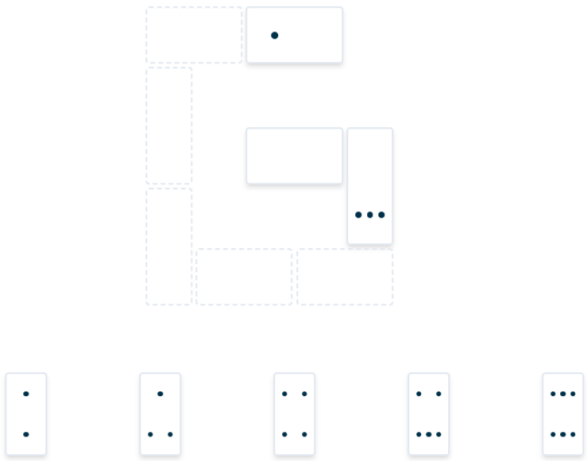

Figura 20: Esempio di puzzle con n=3 e c=2

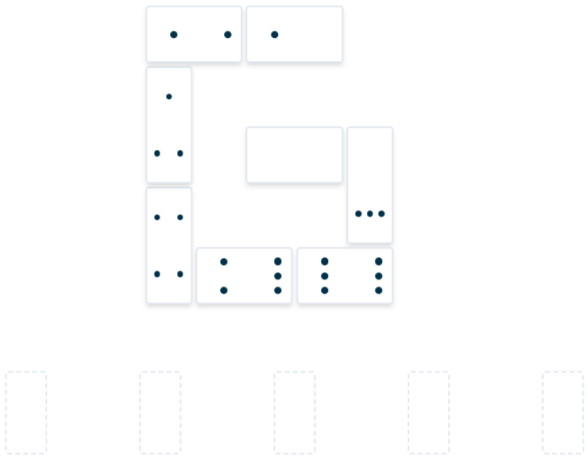

Figura 21: Soluzione del puzzle con n=3 e c=2

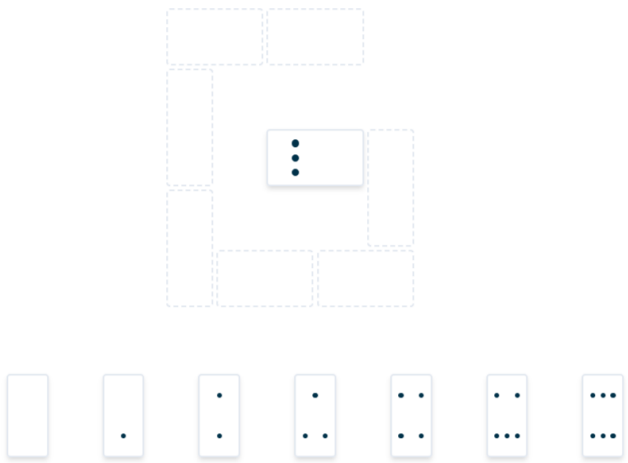

Figura 22: Esempio di puzzle con n=3 e c=3

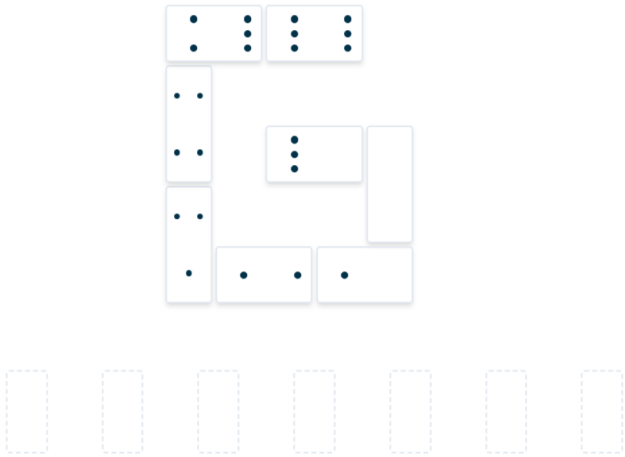

Figura 23: Soluzione del puzzle con n=3 e c=3

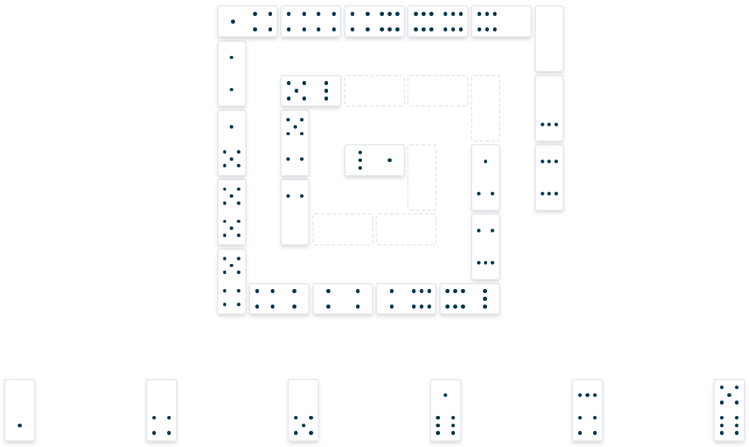

Figura 24: Esempio di puzzle con n=6 e c=1

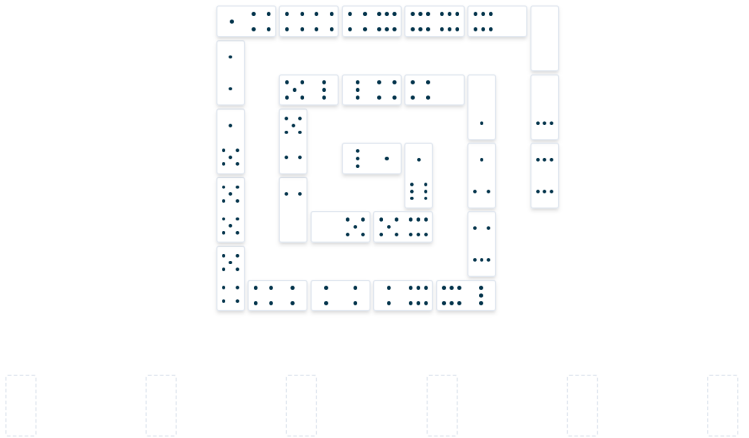

Figura 25: Soluzione del puzzle con n=6 e c=1

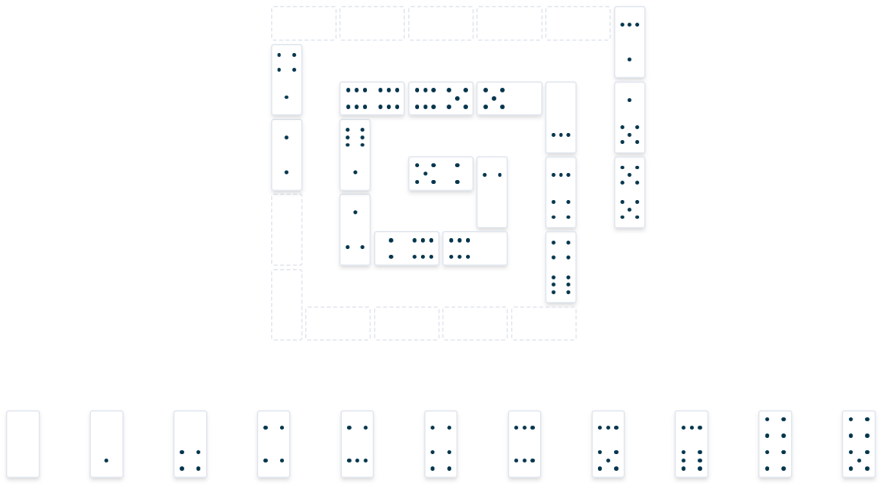

Figura 26: Esempio di puzzle con n=6 e c=2

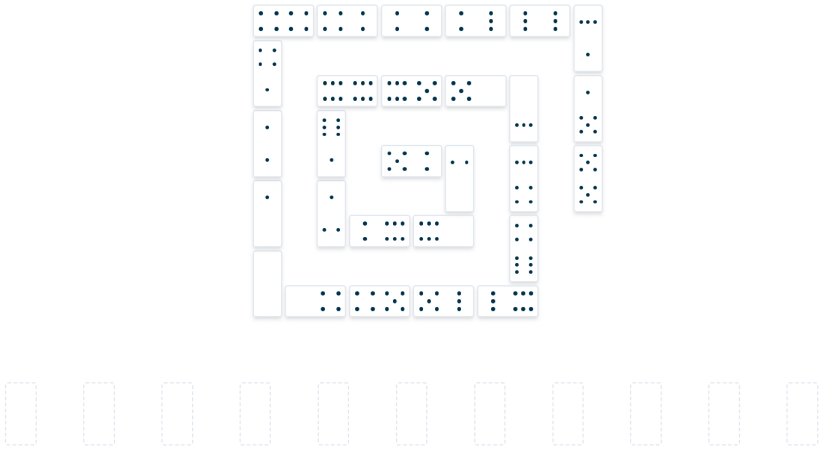

Figura 27: Soluzione del puzzle con n=6 e c=2

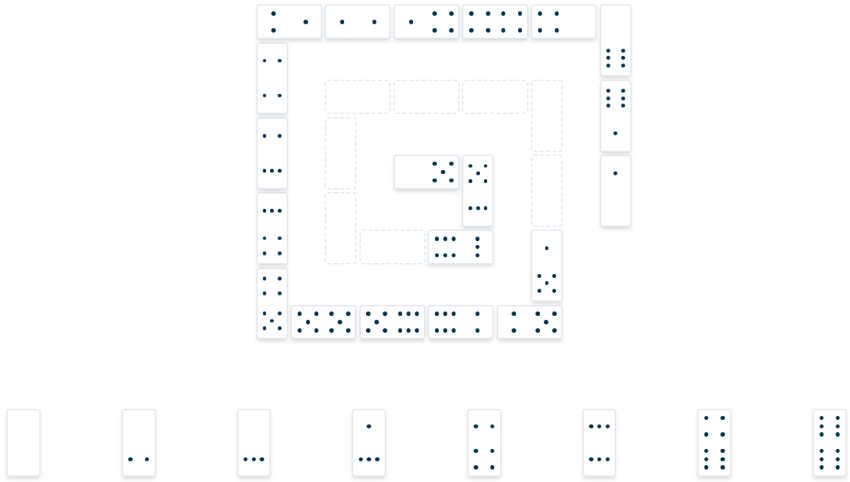

Figura 28: Esempio di puzzle con n=6 e c=3

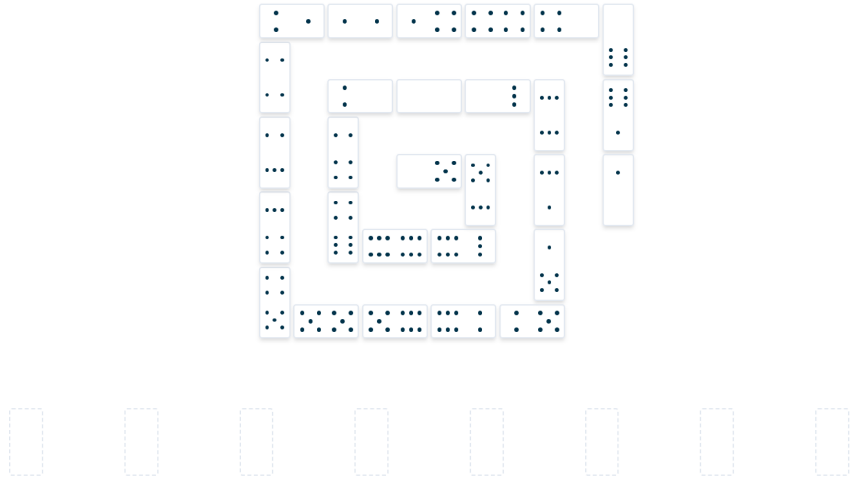

Figura 29: Soluzione del puzzle con n=6 e c=3

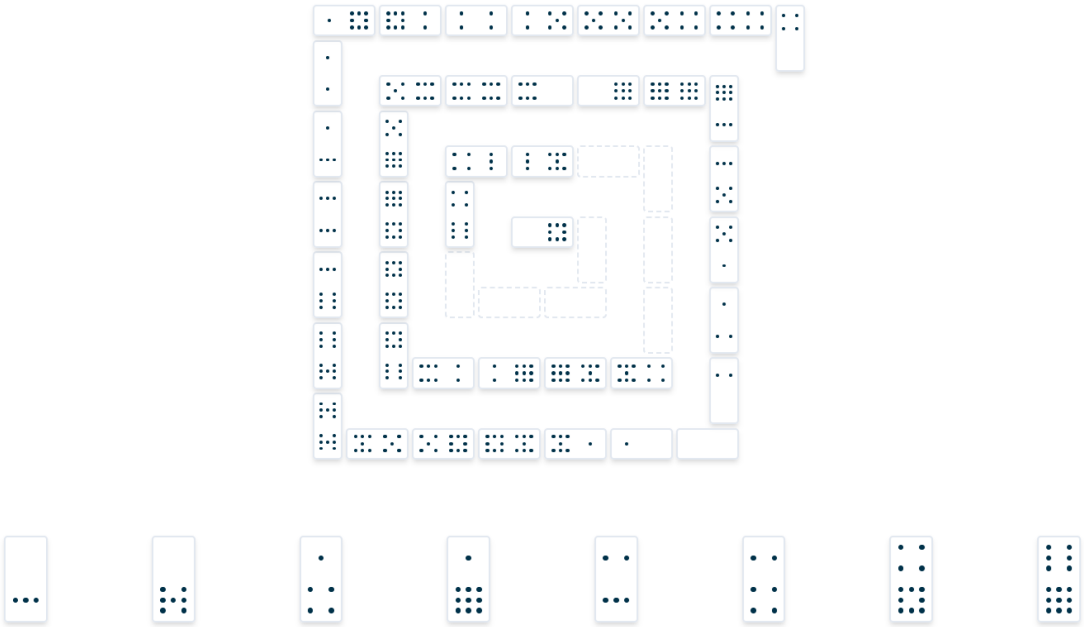

Figura 30: Esempio di puzzle con n=9 e c=1

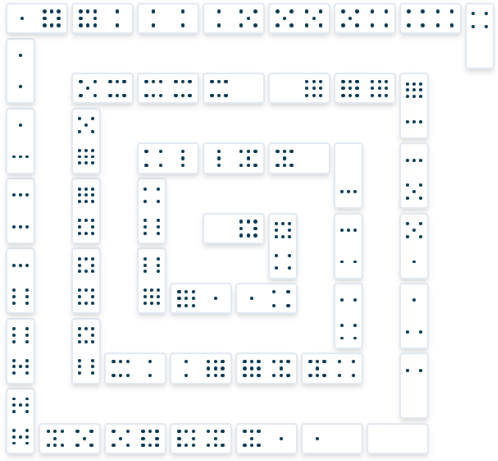

Figura 31: Soluzione del puzzle con n=9 e c=1

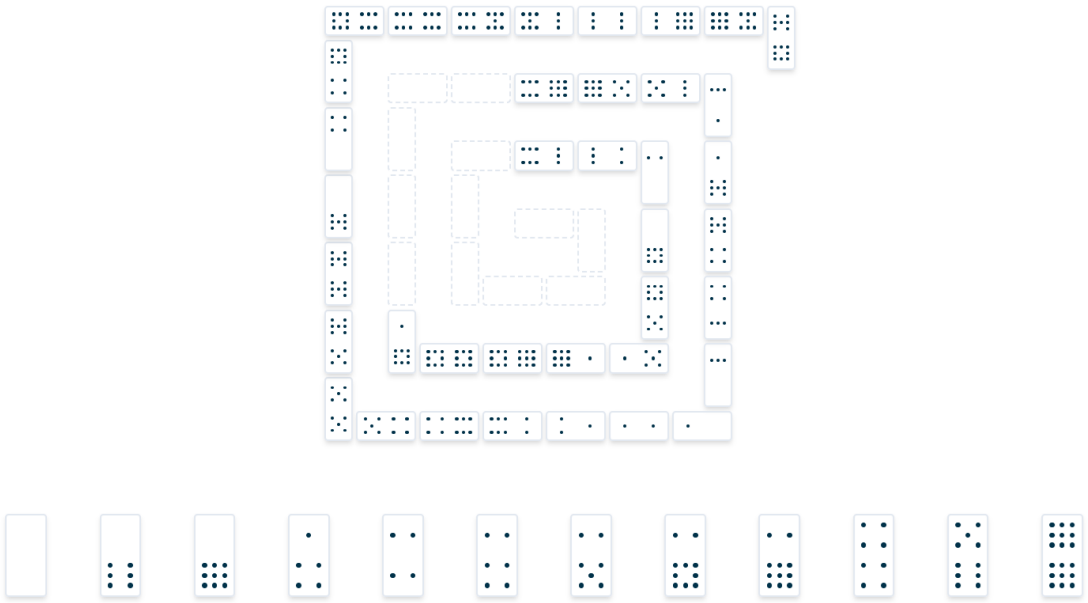

Figura 32: Esempio di puzzle con n=9 e c=2

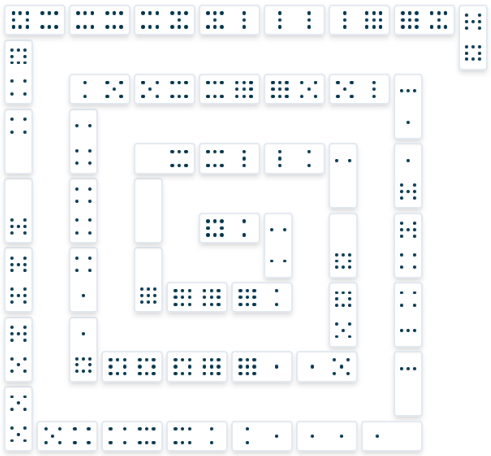

Figura 33: Soluzione del puzzle con n=9 e c=2

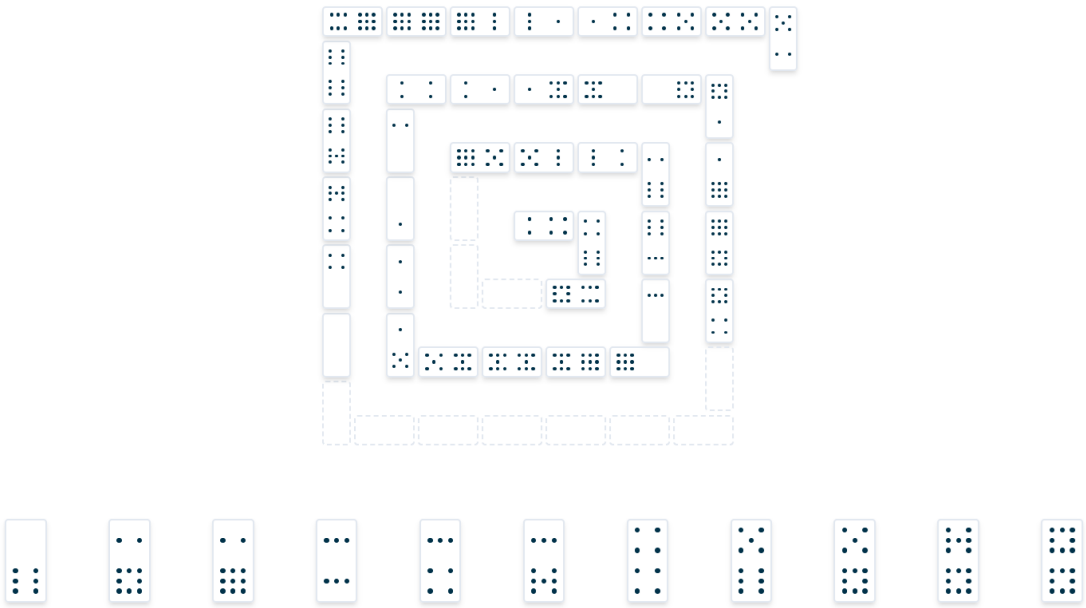

Figura 34: Esempio di puzzle con n=9 e c=3

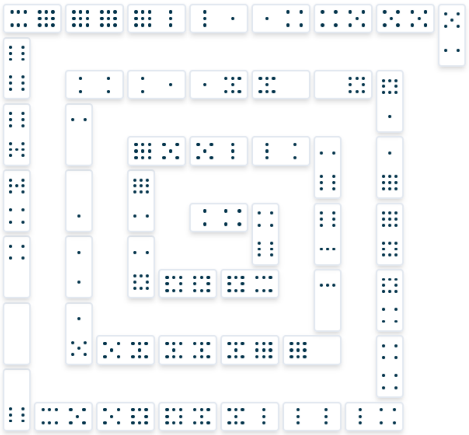

Figura 35: Soluzione del puzzle con n=9 e c=3

## 4.5 Metodi alternativi alla PL: la risoluzione con il completamento degli orientamenti

Ricordando le nozioni dei capitoli introduttivi al gioco, in particolare relativi alle tessere doppie e ai set da gioco con N dispari rispettivamente spiegate in 3.2, 3.3 e i concetti relativi alla teoria dei grafi, relativi soprattutto alla topologia e alla colorazione di un grafo (capitoli 2.6.1 e 2.6.2) in questo capitolo viene presentato un metodo per la risoluzione di puzzle a mezzo della teoria dei grafi. La premessa fatta è che per l'attuazione del metodo seguente è necessario in virtù dei concetti citati considerare puzzle senza tessere doppie e che se sono generati a partire da set da gioco con N dispari non utilizzano tile che rendano il grado di un singolo nodo diverso da 0.

Seguendo l'analogia tra un set da gioco ed un grafo regolare, tra una sequenza e un cammino sulla versione orientata del grafo regolare e infine tra un puzzle ed un grafo regolare parzialmente orientato seguendo quanto esposto da Bang-Jensen J., Huang J., Zhu X, nell'articolo 'Completing orientations of partially oriented graphs'[1], possiamo individuare un metodo univoco per il completamento dell'orientamento del grafo regolare/completamento del puzzle. Gli autori spiegano inizialmente che dato un grafo parzialmente orientato indicato come G, che rispecchia le caratteristiche sopra riportate, e necessario calcolare un grafo definito come sottostante, che includa tutti e soli gli spigoli non orientati nel grafo di partenza.

Ad esempio dato un grafo regolare parzialmente orientato con N pari a 3 dopo aver 'normalizzato' il grafo rimuovendo gli archi dati dal fatto che il grafo ha N dispari e rimuovendo gli auto-anelli abbiamo:

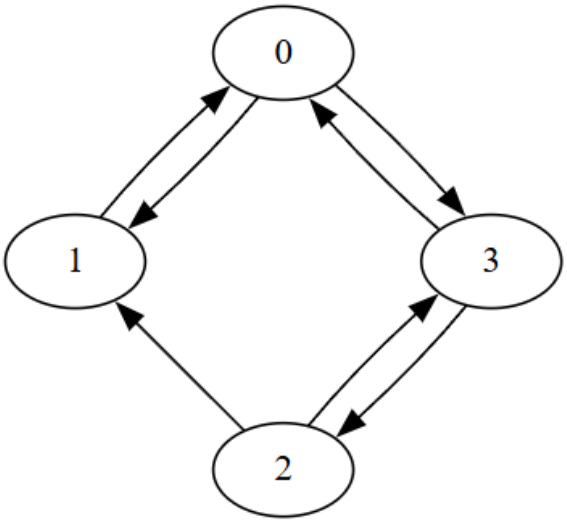


Figura 36: Un grafo parzialmente orientato rappresentante un puzzle con N=3

Successivamente si calcola il grafo sottostante che comprende solamente gli spigoli presenti nel grafo parzialmente orientato, che sono stati rappresentati usando archi in entrambe le direzioni.

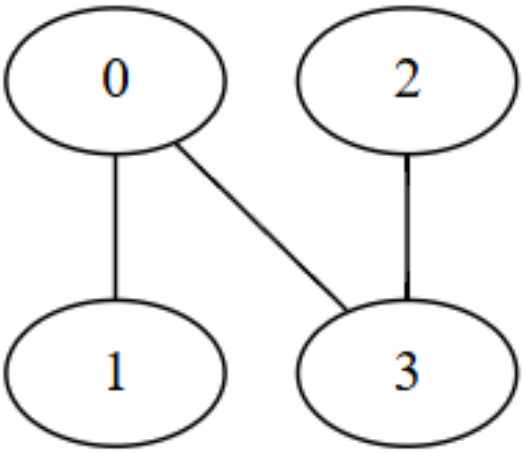


Figura 37: Un grafo sottostante rappresentante un puzzle con N=3

Ed infine si computa ciò che l'autore definisce un **grafo ausiliario**. Il grafo ausiliario avrà per **vertici** entrambi i possibili orientamenti degli spigoli del grafo sottostante (figura 37) appena costruito. Gli **spigoli** del grafo ausiliario saranno determinati seguendo la regola seguente, dati due vertici del grafo ausiliario uv e u'v' essi sono adiacenti nel grafo ausiliario se soddisfano una delle tre condizioni qui riportate:

1. u = v' e v = u'
2. u = u' e vv' non appartiene agli spigoli del grafo G
3. v = v' e uu' non appartiene agli spigoli del grafo G

Ad esempio dato il grafo G (figura 36) avremo che i nodi del grafo ausiliario sono: 01, 10, 03, 30, 23 e 32 in quanto rappresentano tutti i possibili orientamenti degli spigoli di G e gli spigoli del grafo ausiliario seguendo le regole elencate in precedenza saranno: 01–10,

10–30, 01–03, 30–03, 03–23, 23–32 e 30–32. Ne risulterà la seguente rappresentazione grafica:

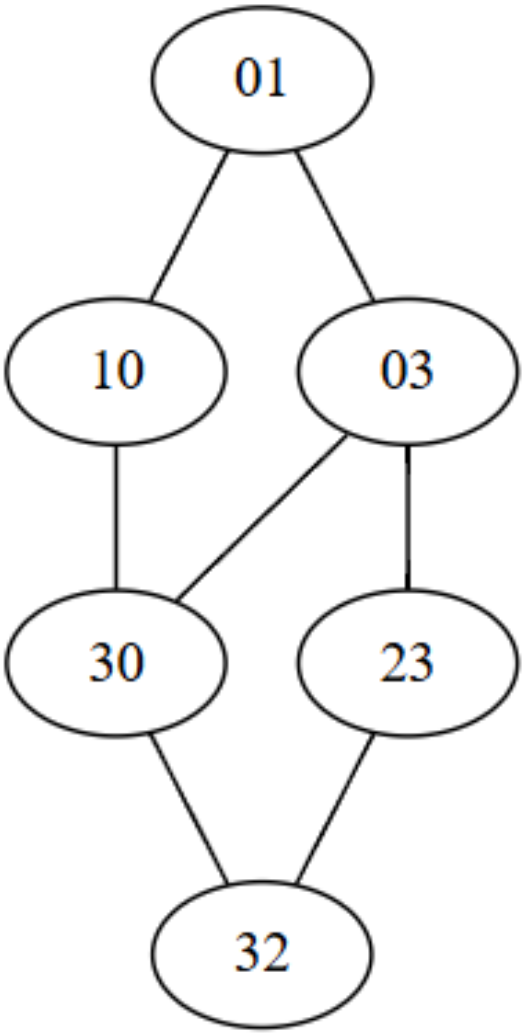


Figura 38: Un grafo ausiliario rappresentante un puzzle con N=3

Si nota come il concetto alla base del grafo ausiliario sia quello di rendere adiacenti gli archi orientati che in G non potrebbero coesistere.

Ognuna delle condizioni di adiacenza dei nodi del grafo ausiliario infatti rappresentano le condizioni che due spigoli nel grafo di partenza non possono mai trovarsi a rispettare. Il parallelismo tra i criteri enunciati nell'esempio in cui sono stati utilizzati uv e u'v' è più generalmente riportato in questo modo:

1. Nel grafo parzialmente orientato uno spigolo non può essere orientato sia in un verso che nel suo verso opposto

2. Nel grafo parzialmente orientato due spigoli da orientare non possono formare un sottografo ciclico con un terzo spigolo non orientato, in quanto un ciclo non orientato non è a sua volta orientabile.

Di conseguenza possiamo chiamare per chiarezza questo grafo ausiliario anche grafo delle non adiacenze. É interessante notare come anche nel gioco del domino queste regole di non adiacenza vengono tradotte:

1. Nella soluzione un puzzle una tessera generica uv non può apparire sia orientata in un verso che nel suo opposto

2. Nella soluzione di un puzzle due tessere che hanno una faccia in comune e lo stesso orientamento (esempio: u sulla faccia sinistra e rispettivamente v e v' sulle facce a destra) possono essere contenute nella soluzione se e solo se la soluzione contiene anche la tessera vv' orientata da sinistra verso destra

Definito quindi il grafo ausiliario gli autori spiegano come è possibile determinare se e come si completa l'orientamento di G. Infatti è necessario che il grafo ausiliario sia bipartiblie e che una volta portata a termine una colorazione dei nodi di questo grafo ausiliario il grafo abbia solamente 2 classi di colorazione. Gli archi da orientare seguendo le regole esposte nell'articolo corrispondono alla prima classe di colorazione individuata, nell' esempio riportato fin'ora nel capitolo:

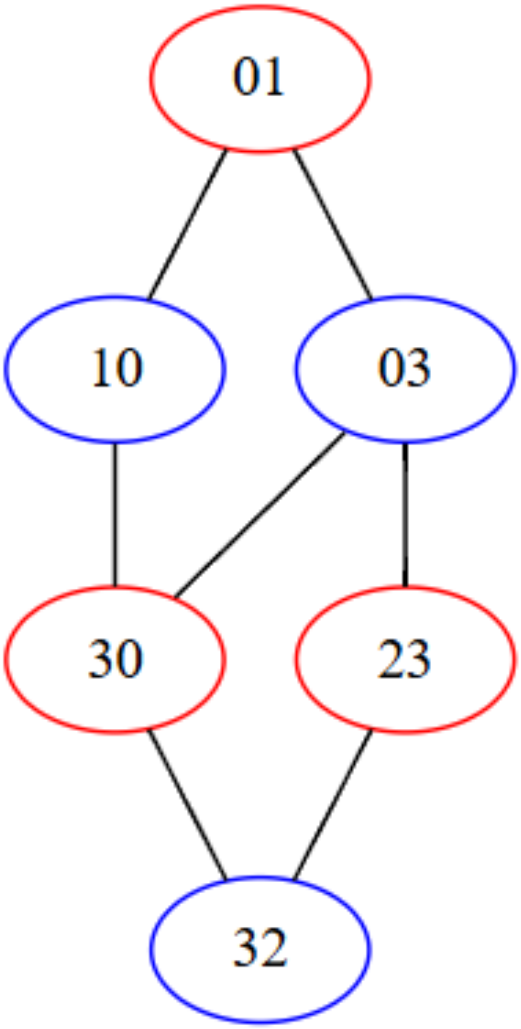


Figura 39: Un grafo ausiliario con colorazione rappresentante un puzzle con N=3

Dal quale si ricava che l'orientamento del puzzle si completa usando gli archi 01, 30 e 23. La motivazione della scelta della classe di colorazione rossa è data dal significato della colorazione. Le classi di colorazione in accordo con il significato delle adiacenze dei nodi del grafo ausiliario rappresentano gli insiemi di orientamenti di archi compatibili. Come ultimo elemento per comprendere questo metodo di completamento degli orientamenti c'è da determinare l'ordinamento con il quale i nodi vengono assegnati ad una classe di colorazione. L'autore stabilisce che affinchè la prima classe di colorazione applicata al grafo ausiliario corrisponda con l'insieme degli spigoli del grafo parzialmente orientato ancora da orientare, è necessario determinare:

1. Un ordinamento di eliminazione eccellente sul grafo G parzialmente orientato, di modo che l'ordine stablito dei vertici di G ad ogni eliminazione non consenta mai al grafo di essere sconnesso

2. Un ordinamento lessicografico dei nodi del grafo ausiliario, una volta trovato l'ordinamento di eliminazione eccellente si ordinano lessicograficamente i vertici del grafo ausiliario

Nell'esempio qui presentato si ha come ordinamento di eliminazione eccellente: 0,1,2,3 e il conseguente ordinamento lessicografico dei vertici del grafo ausiliario è: 01, 03, 10, 23, 30 e infine 32. L'autore una volta spiegato l'ordinamento da seguire nella colorazione spiega che una volta estratto un vertice dalla testa dell'ordinamento (al primo passo 01) bisogna estendere la colorazione con la classe opposta ai vertici adiacenti. Se ne ricavano nella prima classe di colorazione del nostro esempio 01, 30 e 23 che quindi con il reinserimento portano ad un'orientamento di questo tipo del grafo iniziale:

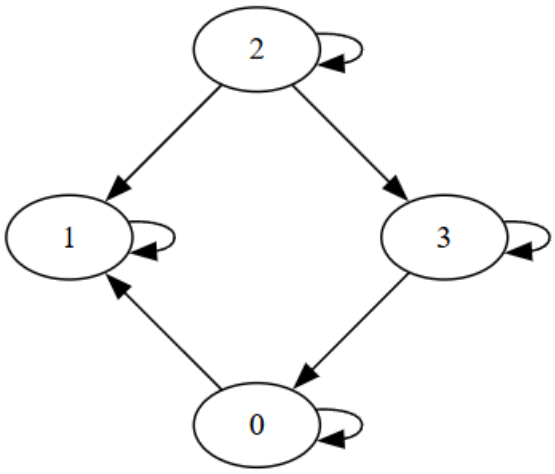


Figura 40: Il risultante orientamento del grafo che rappresenta un puzzle con N=3

## 4.6 Gli algoritmi del completamento degli orientamenti e la loro complessità

Riassumendo la procedura illustrata nel capitolo in forma compatta gli algoritmi necessari al completamento di un orientamento sono:

**Algorithm 2:** Algoritmo per la generazione del grafo ausiliario

**Data:** Il grafo parzialmente orientato P
**Result:** Un grafo ausiliario non orientato

```
1  Si ricava un grafo sottostante G non orientato rimuovendo gli
     archi orientati dal grafo P;
2  Si crea il grafo risultato G+ inizialmente vuoto;
3  foreach uv ∈ G do
4      Si aggiungono a G+ i vertici corrispondenti ai due
         orientamenti possibili ad arco dello spigolo corrente uv e
         vu;
5  foreach (uv, u'v') ∈ G+ do
6      if u == v' and v == u' then
7          Si aggiunge uno spigolo in G+ collegante uv con u'v';
8      else if u == u' and vv' ∉ G then
9          Si aggiunge uno spigolo in G+ collegante uv con u'v';
10     else if v == v' and uu' ∉ G then
11         Si aggiunge uno spigolo in G+ collegante uv con u'v';
12 return Il grafo G+;
```

L'algoritmo per la generazione del grafo ausiliario ha una complessità pari a $N^2$

**Algorithm 3:** Algoritmo per la ricerca di un ordinamento di eliminazione perfetto usando LexBFS

**Data:** Il grafo connesso e non orientato $G = (V, E)$
**Result:** Un vettore con l'ordine perfetto di eliminazione dei vertici di $G$

```
1  Si inizializza il vettore 'risultato' come un vettore vuoto;
2  Si inizializza un dizionario 'label' dove ogni vertice in V ha
     un'etichetta vuota;
3  Si sceglie un vertice v_0 arbitrariamente come punto di
     partenza;
4  Si assegna a v_0 l'etichetta [v_0];
5  Si inizializza una coda Q contenente solo v_0;
6  while la coda Q non è vuota do
7      Si estrae dalla coda il vertice v con l'etichetta più grande
         (lessicograficamente);
8      Si aggiunge v al vettore 'risultato';
9      foreach vicino u di v non ancora visitato do
10         Si prepone l'identificatore di v all'etichetta di u, ovvero
             u ← v_u;
11         Si aggiunge u alla coda Q (se non già presente) con la
             nuova etichetta;
12     Si rimuove v dal grafo insieme ai suoi spigoli;
13 return Il vettore 'risultato' contenente l'ordine di
     eliminazione perfetto;
```

L'algoritmo appena presentato ha complessità $V + E$ in quanto suddividendo l'algoritmo nella fase di inizializzazione del dizionario e la ricerca del nodo lessicograficamente maggiore si riscontra come la complessità sia il risultato dell'espressione:
$O(V+E)$ e notando come nel nostro caso l'utilità dell'algoritmo è data dall'applicazione su grafi connessi dove il numero di spigoli E è pari

all'incirca a $V^2$ si ottiene $O(V + V^2) = O(V^2)$. Se si considera il tutto in funzione del valore di N iniziale $O(N^2)$

**Algorithm 4:** Algoritmo per la colorazione lessicografica a 2

**Data:** Un grafo sottostante G, Un grafo ausiliario/delle non-adiacenze $G^+$

**Result:** Una mappa contenente le associazioni tra ogni vertice di $G^+$ ed una classe di colorazione

1 Si inizializza una mappa vuota delle colorazioni effettuate su $G^+$;
2 Si trova un ordine di eliminazione perfetto con l'algoritmo 3 in funzione di G;
3 Si stabilisce un ordine lessicografico di $G^+$ dove uv precede u'v' se u $<$ u' (secondo l'ordinamento di eliminazione perfetto) oppure u = u' e v $<$ v' (secondo l'ordinamento di eliminazione perfetto);
4 **while** *Tutti i nodi non sono colorati* **do**
5  Si trova il nodo non colorato minore secondo l'ordine lessicografico stabilito;
6  Lo si colora con la prima classe di colorazione;
7  Si estende la colorazione ai vertici di $G^+$ connessi al vertice appena colorato, i vertici connessi di distanza dispari avranno la seconda classe mentre quelli a distanza pari avranno la prima;
8 **return** *La mappa delle colorazioni effettuate*

L'algoritmo per la colorazione del grafo ausiliario è suddivisibile nella ricerca di un ordine di eliminazione perfetto (complessità $O(N^2)$), nella costruzione di un ordine lessicografico (complessità $O(N^2)$ in quanto le combinazioni possibili di uv nel grafo ausiliario sono pari alle combinazioni possibili di N nodi presi a due a due) ed infine nella ricerca di un nodo non colorato minore secondo l'ordine lessico-

grafico che coincide con la ricerca su di un vettore lungo $N^2$ o utilizzando migliori strutture dati una complessità minore. In conclusione $O(N^2 + N^2 + N^2) = O(N^2)$.

**Algorithm 5:** Algoritmo per la validazione del puzzle P

**Input:** Un grafo parzialmente orientato
**Output:** Un risultato booleano positivo se il puzzle è valido
negativo altrimenti

```
1  Si ricava N conteggiando il numero di vertici nel grafo e
     sottraendo 1;
2  if N è pari then
3      Si genera il grafo ausiliario G+ con l'algoritmo 2;
4      Si colora il grafo ausiliario G+ con l'algoritmo 4;
5      if Le classi di colorazione sono 2 then
6          return Il grafo è valido;
7  else
8      Si inizializza un vettore dei vertici.;
9      Si sceglie un vertice arbitrario di partenza.;
10     while Il vertice scelto ha dei vicini non visitati do
11         Si marca un vicino non visitato arbitrario come visitato;
12         Si sceglie il vicino marcato come nuovo vertice scelto;
13     if Tutti i vicini sono stati visitati then
14         return Il grafo è valido;
15 return Il grafo non è valido;
```

L'algoritmo di validazione è composto di due blocchi condizionali, se N è pari la sua complessità è uguale alla somma delle complessità dovute alla generazione e colorazione del grafo ausiliario $O(N^2 + N^2)$ altrimenti alla verifica che il grafo sia connesso comporta per ogni vertice (N+1 vertici) la verifica di ognuno dei suo vicini (al massimo N+1) e quindi ha una complessità di $O(N^2)$. In entrambi i casi le operazioni

comportano una complessità di $O(N^2)$

# 5 Implementazione software di una piattaforma per il gioco

A seguito degli enunciati dei capitoli precedenti sul gioco del domino in solitaria, a completamento della presente tesi di laurea è stato realizzato un software per rendere il gioco accessibile pubblicamente e in maniera gratuita. Anche il codice sorgente del software è reso disponibile e riutilizzabile sotto licenza MIT per chiunque abbia intenzione di riprodurre una variante del gioco all'indirizzo https://github.com/DominoOrg/domino-app. E' possibile anche disporre secondo le stesse modalità di utilizzo di un container docker pubblicato sul github container registry all'indirizzo github: ghcr.io/nunzioono/domino-app/domino-app.

## 5.1 Architettura client-server

L'architettura scelta per l'applicazione è quella a due componenti composta da un client ed un server. La scelta in questione è stata presa a seguito dell'analisi delle tempistiche necessarie all'elaborazione degli schemi del gioco. In maniera funzionale l'architettura client-server in questa applicazione permette di avere in maniera centralizzata e asincrona rispetto agli utenti accesso ad una vasta gamma di schemi in breve tempo. Dall'altro lato per l'utente vi è il vantaggio di poter condividere con altri utenti gli schemi da risolvere tramite identificativi unici dei puzzle mantenuti in corrispondenza degli stessi nel database presente sul server.

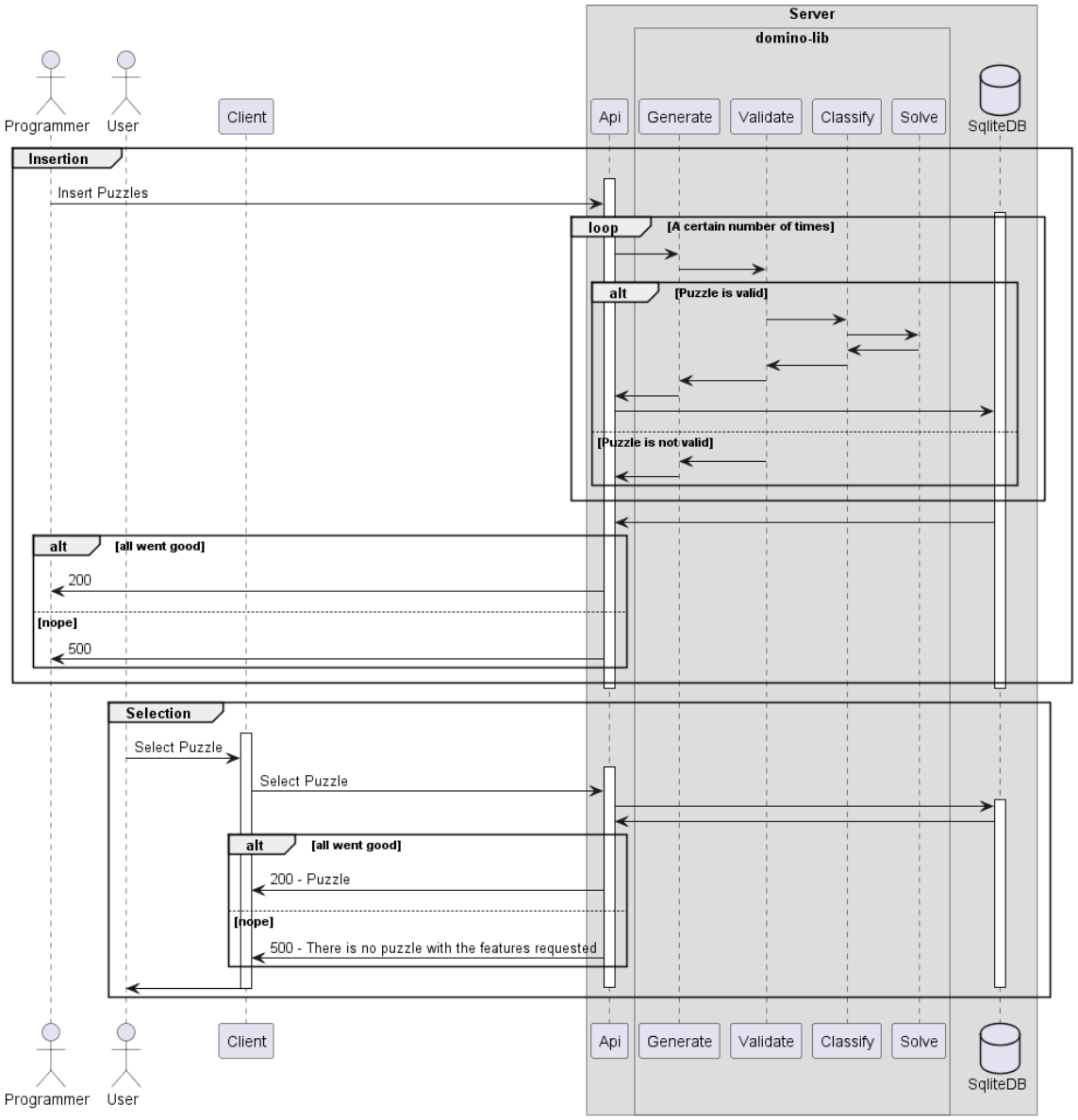


Figura 41: Rappresentazione grafica dell'architettura del software tramite un modello a sequenza usando plantuml

## 5.2 Deployment e compatibiltà dell'applicazione con le piattaforme

La modalità scelta per lo sviluppo dell'applicazione prevede un client eseguibile all'interno di tutti i browser maggiormente utilizzati tramite l'uso dei protocolli standard web sviluppati dalla mozilla developer network, d'altra parte anche il server può essere eseguito su qualsiasi tipo di server in quanto sviluppato come container eseguibile all'interno dell'ambiente di virtualizzazione docker.

## 5.3 Indicazioni sulla configurazione di rete

Affinchè l'applicazione sia distribuita sulla rete sono state effettuate configurazioni di rete sui router della rete locale che vengono qui condivise quantomeno nel loro significato affinchè il deployment dell'applicazione sia completamente trasparente e riproducibile. Al fine di distribuire l'applicazione se installata su di un server proprietario necessita di esporre una porta dell'host a internet. La configurazione del router in questo caso opera il mapping delle porte su cui si espone l'applicativo con le porte del router stesso (occorre non sovrascrivere porte comunemente usate da altri servizi per non compromettere la propria rete locale). In piu' si è usato un comune reverse proxy (nginx) per reindirizzare le richieste che giungono al server rimpiazzando l'indirizzo sorgente delle richieste con l'indirizzo del server stesso. L'utilizzo del reverse proxy è mirato solamente all' aggiunta di un livello di protezione aggiuntiva, infatti questo permette di esporre il server solamente alle richieste ad uno specifico dominio perchè queste ultime vengono mappate a richieste locali e quindi servite mentre richieste all'indirizzo del server non venendo direttamente mappate non vengono servite. Per la scelta di un domino e il routing delle richieste dns data la natura dell sistema di certificazione dei server dns e la complessità

dei processi di autenticazione presso lo IANA si è utilizzato un servizio di aziende terze.

## 5.4 Struttura del server

Internamente il server è scritto in rust facendo leva sulle proprietà di sicurezza e velocità del linguaggio. Il server web è sviluppato utilizzando il framework rocket che fornisce un api semplice per scrivere gli endpoint necessari alle applicazioni tramite semplici macro e derive dei tratti, inoltre questo framework offre un interoperabilità con molti tool, quali object-relational mappers (ORM) come Sea-orm, driver per database come sqlx per connettere semplicemente le applicazioni ai database supportati (tra i piu' conosciuti sqlite, postgresql e mysql) e altre integrazioni comode per il riutilizzo di codice per operazioni comuni nel mondo dello sviluppo di server web. Il dbms (database management system) scelto per l'applicativo è Sqlite, conosciuto per la stabilità, la sicurezza (progetto open source il cui sviluppo è iniziato nel 2004) e la larga adozione sul web e nel mondo delle applicazioni mobile. Sqlite è caratterizzato dall'essere implementato all'interno di un singolo file rendendo chiaro che le dimensioni dell'overhead introdotto dal lato di gestioni dei dati al database stesso sono minime. Di contro questa scelta implementativa rende questa tipologia di database single-threaded e quindi limitata nel numero di accessi/query che il database opera in un'unità di tempo prestabilita rendendo questo tipo di database adatta maggiormente ad applicazioni che non posseggono un numero di richieste grande e consistente nel tempo. Nonostante questo difetto che rende questa tipologia non scalabile con una crescita del traffico delle applicazioni web sono noti casi di utilizzo di questo database per ridurre la latenza delle richieste degli utenti utilizzando questa tecnologia come cache di dati per le richieste frequenti. Al momento attuale sqlite tramite il suo sito ufficiale stima l'utilizzo del database stesso su oltre 1 triliardo di dispositivi ($10^{12}$ dispositivi).

La struttura scelta per lo schema del database invece è la seguente:

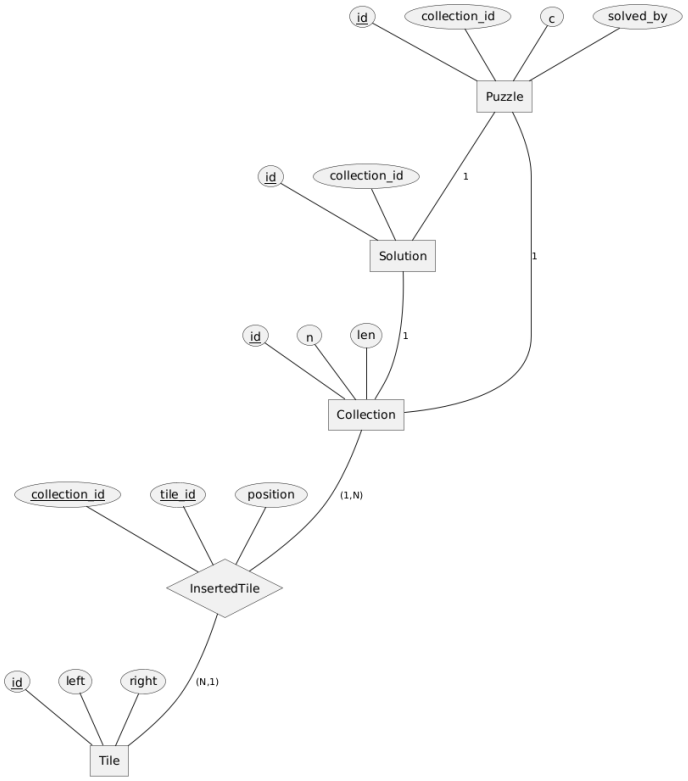


Figura 42: Schema sql per il database

Questo schema del database permette di tenere traccia del mapping tra l'id dei puzzle e la loro composizione in termini di collezioni di tessere. Al tempo lo schema permette di rappresentare la soluzione di un puzzle come collezione avendo un indicatore al quale poter collegare il puzzle e la sua soluzione. La scelta piu' significativa dello schema per quanto esso sia abbastanza semplice e' quella di far puntare le collezioni alle tessere tramite una tabella aggiuntiva e rappresentando le tessere come un'entità a se stante in questo modo rappresentando le collezioni come un insieme di riferimenti alle tessere e non a colonne aggiuntive nella tabella delle tessere inserite si ottiene un ottimizzazione significa-

tiva della dimensione dei dati conservati nel database a fronte di una stessa capacità di rappresentare le stesse informazioni. Nell'implementazione qui descritta non sono stati adoperati indici, trigger o viste che ottimizzino le query piu' comuni ma si lascia questa possibilità di integrazione aperta al lettore. Potendo modificare il database direttamente senza compromettere le funzionalità dell'applicazione lato server.

## 5.5 Struttura del client

Il client dell'applicazione è scritto in typescript, la scelta del linguaggio ricade sulla versione tipizzata del linguaggio javascript al fine di ottenere un feedback durante lo sviluppo tramite tool appositi per evidenziare precocemente gli errori. Il client è sviluppato facendo uso della libreria react, comunemente diffusa per lo sviluppo di applicazioni web, supporta il re-rendering delle componenti di un sito web in maniera reattiva. Quest'ultima proprietà della libreria la rende particolarmente adatta allo scopo di questa applicazione in quanto capace di aggiornare le schermate modificando solamente alcune porzioni dello schermo , riducendo i tempi in cui il browser ricarica la pagina e fornendo quindi infine un esperienza piu' fluida (senza interruzioni) all'utente.

## 5.6 Struttura della libreria principale ‘domino-lib‘

La libreria è sviluppata al fine di assolvere alle comuni relative alla creazioni di schemi per il gioco ed è sviluppata in rust. Essendo le operazioni relative alla creazione degli schemi esse vengono performate prima che l'utente inizia ad utilizzare l'applicazione web. Data questa informazione la librerie viene utilizzata lato server. Tuttavia essendo realizzata in rust, è possibile modificare lievemente la libreria per compilare in wasm (web assembly). Al fine di poter estendere le funzionalità della stessa durante lo sviluppo si è seguito lo stile di progettazione SOLID, il codice scritto è quindi riconducibile per ogni scopo ad una funzione/struttura dati, ha possibilità di essere esteso ma non di essere modificato senza avere errori di compilazione, non utilizza il concetto di ereditarietà e sostituisce la dipendenza ai tipi di dato con la dipendenza alle interfacce rendendo quindi facilmente sostituibili le strategie implementative interne.

# 6 Conclusioni

Il lavoro svolto si conclude con un risultato evidente dell'applicabilità delle tecniche di ricerca operativa al gioco del domino. Inoltre la letteratura citata presenta interesse accademico preesistente sul gioco del domino spesso accompagnato da metodi per la risoluzione di problemi considerati complessi senza i metodi introdotti contestualmente al gioco. Tra i risultati dello studio presentato si è riscontrata una difficoltà nel conciliare la generazione dei puzzle e al contempo far si che essi presentino una complessità data lasciando aperta la questione teorica riguardante l'esistenza di metodi strutturali che individuino la complessità univocamente come un aspetto facente parte della rappresentazione matematica dei puzzle stessi. La valutazione della complessità fornita in questo studio è da intendersi efficace ma non collegata effettivamente con una metrica valutabile in maniera incrementale e precisa. Infatti si sottolinea che la variazione di complessità individuata dalla formula fornita non tiene conto della tessera rimossa ma della posizione e della relazione che essa ha con altre tessere rimosse. L'idea maggiormente innovativa presentata in questo documento risulta quindi essere la possibile applicazione del metodo di completamento degli orientamenti per la risoluzione di puzzle e il controllo della validità di questi ultimi. Sono stati individuati diversi metodi esposti quali tecniche relative alla PL e tecniche relative alla teoria dei grafi, queste sono state valutate per la loro efficienza e risultano ognuna nell'avere dei vantaggi e degli svantaggi come spesso accade nell'ambito ingegneristico. Si ribadisce come ad esempio le tecniche basate sulla PL siano in grado di risolvere puzzle comprendenti tessere doppie mentre i metodi basati sui grafi non prevedono il loro utilizzo. D'altra parte i metodi basati sui grafi se i problemi dati riguardanti i puzzle sono osservati su larga scala risultano più efficienti.

# 7 Bibliografia


1. Bang-Jensen J., Huang J., Zhu X.
   Completing orientations of partially oriented graphs.
   Journal of Graph Theory, p.285-304, 2018.

2. Bellman R.
   Dynamic Programming
   Princeton University Press, p.192, 1957

3. Biggs N., Lloyd K. E. , Wilson R. J.
   Graph Theory, 1736-1936
   Clarendon Press, p.252, 1986

4. Billaut J.-C., Della Croce F., Salassa F., T'kindt V.
   No-idle, no-wait: when shop scheduling meets dominoes, eulerian and hamiltonian paths
   Journal of Scheduling, Vol.22, p.59-68, 2019.

5. Dantzig, G.B.
   Maximization of a Linear Function of Variables Subject to Linear Inequalities.
   In: Koopmans, T.C., Ed., Activity Analysis of Production and Allocation,
   Wiley & Chapman-Hall, p. 339-347, 1947.

6. Demaine E.D., Ma F., Waingarten F.
   Playing Dominoes Is Hard, Except by Yourself.
   FUN 2014, LNCS, p.137-146, 2014

7. Doig A.G., Land A.H.
   An Automatic Method of Solving Discrete Programming Problems
   Econometrica, Vol. 28, No. 3, p. 497-520, 1960

8. Hansen, P., Mladenović, N.
Variable Neighborhood Search.
In: Burke, E.K., Kendall, G. (eds) Search Methodologies.
Springer, Boston, p.211-238, 2005

9. Hearn R.A., Demaine E.D.
Games, Puzzles, and Computation.
AK Peters Limited, p.248, 2009

10. Hierholzer Carl,
https://en.wikipedia.org/wiki/Eulerian_path
Wikipedia, 2025

11. Holger H. H., Thomas Stützle
Stochastic Local Search
Handbook of Approximation Algorithms and Metaheuristics, vol.1,
p.240-269, 2018

12. Kirkpatrick S., Gelatt Jr. C. D., Vecchi M. P.
Optimization by Simulated Annealing
Science, vol.220, p. 671-680, 1983

13. Euler L.
LEONHARD EULER AND THE KOENIGSBERG BRIDGES
Scientific American, a division of Nature America, Inc., vol. 189,
p. 66-72, 1953

14. Padberg, M., Rinaldi, G.
A Branch-and-Cut Algorithm for the Resolution of Large-Scale
Symmetric Traveling Salesman Problems.
SIAM Review, p.60-100, 1991.

15. Ramalhinho Lourenço, H.
Iterated Local Search: Applications and Extensions

In Proceedings of the 8th International Conference on Operations Research and Enterprise Systems, p.7-15, 2019.

16. Reddy, D. Raj.
Speech Understanding Systems: A Summary of Results of the Five-Year Research Effort at Carnegie-Mellon University.
Defense Technical Information Center, p.147, 1977

17. Tadei F., Della Croce F.
Elementi di Ricerca Operativa
Società Editrice Esculapio, p.159, 2010

18. Wikipedia contributors
Politopo
https://it.wikipedia.org/wiki/Politopo,
Wikipedia, 2025

19. Wikipedia contributors
Domino
https://it.wikipedia.org/wiki/Domino,
Wikipedia, 2024

20. Zambelli G.
Note sull'algoritmo del simplesso
https://www.math.unipd.it/ giacomo/ProgMat/Note/simplessoPM.pdf,
s.d.