

UNIVERSITÀ DEGLI STUDI DI MILANO  
Facoltà di Scienze Matematiche, Fisiche e Naturali  
Dipartimento di Tecnologie dell'Informazione



## **Algoritmi di ricerca locale per un problema di sequenziamento ottimo**

Relatore: Prof. Giovanni RIGHINI

Correlatore: Prof. Roberto CORDONE

Tesi di Laurea di:  
Andrea PINCIROLI  
Matr. n° 616485

Anno Accademico 2002/2003



*Dedica*



## ***Ringraziamenti***



## Indice

<b>1. Descrizione del problema</b>	<b>11</b>
1.1 Il contesto	11
1.1.1 I processi di pianificazione e schedulazione	11
1.1.2 Requisiti della verniciatura	12
1.1.3 Requisiti della linea di assemblaggio	12
1.1.4 Ratio constraint ad alta priorità facili e difficili da soddisfare	14
1.1.5 Ruolo del giorno di produzione D - 1	14
1.2 Problema da risolvere	15
1.2.1 Ottimizzazione multi-obiettivo con massima priorità ai ratio constraint	15
1.2.2 Ottimizzazione multi-obiettivo con massima priorità ai cambi di colore	16
1.3 Dettagli sul calcolo del numero di violazioni dei ratio constraint	17
1.4 Dettagli sul calcolo del numero di cambi di colore	20
<b>2. Input</b>	<b>21</b>
2.1 Descrizione dei test set	21
2.1.1 Test set	21
2.1.2 Contenuto di un test set	22
2.2 Formato dei file	23
2.2.1 File della limitazione delle sequenze di auto con lo stesso colore	23
2.2.2 File degli obiettivi di ottimizzazione	23
2.2.3 File dei ratio constraint	26
2.2.4 File dei veicoli	27

<b>3. Tecniche risolutive</b>	<b>31</b>
3.1 Il problema del tempo	31
3.2 Diagramma di flusso globale	32
3.2.1 Inizializzazione e caricamento dati	34
3.2.2 Funzionamento	35
3.2.3 Strutture dati	36
 <b>4. Tecniche costruttive</b>	 <b>39</b>
 <b>5. Ricerca locale</b>	 <b>43</b>
5.1 Introduzione	43
5.2 Diagramma a blocchi della ricerca locale	45
5.2.1 Funzionamento generale ricerca locale	46
5.3 Strutture dati	47
5.3.1 Matrice optional finestra	48
5.3.2 Matrice costi di scambio	50
5.3.3 Matrice costi di spostamento	51
5.4 Pesatura degli obiettivi	52
5.4.1 Dettagli sui costi calcolati	52
5.5 Algoritmi di calcolo dei costi	54
5.5.1 Calcolo dei costi di scambio relativi ai ratio constraint	55
5.5.2 Calcolo dei costi di spostamento relativi ai ratio constraint	58
5.5.3 Calcolo dei costi di scambio relativi ai cambi di colore	63
5.5.4 Calcolo dei costi di spostamento relativi ai cambi di colore	66
5.6 La fase di ottimizzazione	69
5.6.1 Dettagli sull'ottimizzazione	70
5.6.2 L'aggiornamento delle strutture dati	72
5.7 Algoritmo di calcolo del costo di una soluzione	75
 <b>6. Output</b>	 <b>79</b>
6.1 File delle soluzioni	79
6.2 Programmi	80



---

6.2.1 Limite al tempo di esecuzione ed ambiente di test . . . . .	80
6.2.2 Gestione del limite di tempo . . . . .	80
6.2.3 Organizzazione gerarchica dei file . . . . .	82
6.3 Procedure di valutazione e classificazione . . . . .	83
6.3.1 Il programma di valutazione . . . . .	83
6.3.2 Metodo di valutazione di uno scenario . . . . .	83
6.3.3 Valutazione globale di un test set . . . . .	85
 <b>7. Conclusioni</b>	 <b>87</b>
7.1 Risultati del test set A . . . . .	87
7.2 Risultati attività di profiling . . . . .	89
7.3 Possibili miglioramenti . . . . .	93
7.3.1 Migliorare la qualità dei risultati . . . . .	93
7.3.2 Incrementare la velocità di esecuzione . . . . .	93
7.3.3 Miglioramenti generali . . . . .	94
 <b>Appendice</b>	 <b>95</b>
A. Indice dei diagrammi . . . . .	95



## **1. Descrizione del problema**

### **1.1 Il contesto**

#### **1.1.1 I processi di pianificazione e schedulazione**

Il problema centrale può essere definito come car sequencing problem, ossia il problema della schedulazione giornaliera della produzione di autovetture.

In questo contesto gli ordini dei clienti sono trasmessi ogni giorno in tempo reale alle fabbriche di automobili.

Il lavoro giornaliero delle fabbriche è (1) quello di assegnare un giorno di produzione ad ogni vettura ordinata, tenendo presente la capacità produttiva e le date di consegna che sono state promesse ai clienti dai venditori.

Le fabbriche devono successivamente (2) schedulare l'ordine di automobili che devono essere prodotte ogni giorno cercando di soddisfare più requisiti possibili dell'impianto: corpo vettura, verniciatura e linea di assemblaggio.

La questione centrale verte sui requisiti della verniciatura e della linea di assemblaggio dato che il corpo vettura non impone requisiti per la schedulazione giornaliera.

Schedulando la produzione di ogni giorno, l'insieme di vetture del giorno di produzione, che è determinato al punto (1), non può essere cambiato.

Un'applicazione industriale si occupa del processo di pianificazione / schedulazione, usando tecniche di programmazione lineare per il livello (1) e di simulated annealing per il livello (2).

### **1.1.2 Requisiti della verniciatura**

La verniciatura deve minimizzare il consumo di solvente. Il solvente è utilizzato per lavare le pistole spray ogni volta che il colore viene cambiato tra 2 vetture schedate consecutive.

C'è quindi il requisito di raggruppare i veicoli per colore, così da minimizzare il numero di cambiamenti di colore in una sequenza di vetture schedate. In altre parole si cerca di minimizzare i lavaggi delle pistole spray, quindi di schedare sequenze di colore il più lunghe possibile.

Le sequenze di colore hanno una limitazione sul massimo numero di vetture che le compongono, dato che le pistole spray devono essere lavate regolarmente, anche se non ci sono cambiamenti di colore nella sequenza di veicoli schedati. Questa limitazione è un Hard Constraint (vincolo forzato).

### **1.1.3 Requisiti della linea di assemblaggio**

Per livellare il carico di lavoro sulla linea di assemblaggio, i veicoli che richiedono speciali operazioni di assemblaggio devono essere distribuiti attraverso l'intero insieme di auto processate. Questi veicoli sono considerati "difficili da produrre" e non possono superare una data quota su una qualsiasi sequenza di veicoli.

Questi requisiti sono modellati come Ratio Constraint N/P (vincoli di rapporto) e sono associati alle caratteristiche delle auto che richiedono operazioni extra nella linea di assemblaggio (ad esempio aria condizionata, tetto panoramico, ecc.).

Un ratio constraint N/P significa che al massimo N auto in ogni sequenza consecutiva di P sono associate al vincolo. Per esempio, se  $N/P = 3/5$ , non ci devono essere più di 3 auto vincolate su ogni sequenza consecutiva di 5 veicoli.

Se  $N/P = 1/P$ , significa che 2 auto vincolate devono essere separate da almeno P-1 auto consecutive non vincolate. Per esempio, con un ratio constraint 1/5: X \_ \_ \_ \_ X è una sequenza accettabile, dove 'X' è un veicolo che necessita di una operazione extra in fase di assemblaggio e '\_' è un veicolo non riguardante il ratio constraint.

Questi vincoli di rapporto sono suddivisi in due classi: ratio constraint ad alta priorità ed a bassa priorità. Quelli ad alta priorità riguardano caratteristiche dell'auto che impongono un elevato carico di lavoro sulla linea di assemblaggio, quelli a bassa priorità causano solamente lievi inconvenienti alla produzione.

I ratio constraint ad alta priorità devono essere soddisfatti in via preferenziale rispetto a quelli a bassa priorità.

Questi vincoli sono considerati soft constraint: il pieno rispetto di tutti i ratio constraint non può essere assicurato prima che la produzione giornaliera sia schedulata.

Un problema che potrebbe sorgere, riguardante questi vincoli, è quello dell'over-constraint, ovvero troppe auto vincolate rispetto al rapporto N/P.

L'obiettivo dell'ottimizzazione è quindi di minimizzare il numero di violazioni dei ratio constraint.

#### **1.1.4 Ratio constraint ad alta priorità “facili” e “difficili” da soddisfare**

La Renault ha fornito scenari con ratio constraint ad alta priorità che possono essere suddivisi in “facili da soddisfare” o “difficili da soddisfare” e che causano un differente impatto sulla valutazione delle soluzioni fornite.

La classificazione “facile da soddisfare” e “difficile da soddisfare” non si applica ai ratio constraint a bassa priorità.

Uno scenario possiede vincoli di rapporto ad alta priorità “facili da soddisfare” se l'applicazione industriale attualmente utilizzata dalla Renault è in grado di generare una sequenza di vetture per il giorno di produzione senza nessuna violazione dei ratio constraint.

Uno scenario si considera avere ratio constraint ad alta priorità “difficili da soddisfare” se l'applicazione industriale Renault non riesce a generare uno schedule del giorno di produzione senza nessuna violazione dei sopraccitati vincoli.

Questo accade quando il numero di veicoli, per i quali sono richieste operazioni extra in fase di assemblaggio, supera il rapporto N/P: un chiaro esempio di vincolo di rapporto difficile da soddisfare si ottiene se il giorno di produzione contiene il 25% di veicoli vincolati e c'è un ratio constraint ad alta priorità di 1/5.

#### **1.1.5 Ruolo del giorno di produzione D – 1**

I veicoli programmati nella sequenza di produzione del giorno D – 1 devono essere tenuti in considerazione durante la decisione della sequenza di vetture del giorno in esame.

Le vetture prodotte il giorno D – 1 sono già schedate e quindi la loro posizione non può essere cambiata.

Il calcolo del numero di violazioni dei ratio constraint sul giorno di produzione  $D$  deve tenere in considerazione gli ultimi veicoli schedati nel giorno di produzione  $D - 1$ ; invece, per quanto riguarda il giorno di produzione  $D + 1$ , esso viene ignorato durante la generazione della sequenza di vetture del giorno  $D$ .

## 1.2 Problema da risolvere

Il problema consiste nel fornire una sequenza di veicoli che soddisfi al meglio i requisiti della verniciatura e della linea di assemblaggio. A seconda della fabbrica possono essere seguite due differenti strategie:

- Massima priorità ai ratio constraint: i requisiti della linea di assemblaggio hanno un livello di priorità più alto di quelli della verniciatura.
- Massima priorità alla verniciatura: i requisiti della verniciatura sono più critici di quelli della linea di assemblaggio.

Per ognuna delle strategie sopra descritte si possono presentare differenti scenari di ottimizzazione.

### 1.2.1 Ottimizzazione multi-obiettivo con massima priorità ai ratio constraint

I seguenti obiettivi devono essere ottimizzati, dall'obiettivo con livello di priorità più alto all'obiettivo con livello di priorità più basso, senza nessuna compensazione tra gli obiettivi, per schedare i veicoli di un giorno di produzione:

1. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
2. Minimizzare il numero di cambi di colore
3. Minimizzare il numero di violazioni dei ratio constraint a bassa priorità

Oppure

1. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
2. Minimizzare il numero di violazioni dei ratio constraint a bassa priorità
3. Minimizzare il numero di cambi di colore

Oppure

1. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
2. Minimizzare il numero di cambi di colore

L'ordine degli obiettivi dipende dalle fabbriche, alcune delle quali non dichiarano ratio constraint di bassa priorità.

### **1.2.2 Ottimizzazione multi-obiettivo con massima priorità ai cambi di colore**

La strategia prevede l'ottimizzazione dei seguenti obiettivi, dal livello di priorità più alto al più basso, senza nessuna compensazione tra gli obiettivi, al fine di schedulare i veicoli di un giorno di produzione:

1. Minimizzare il numero di cambi di colore
2. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
3. Minimizzare il numero di violazioni dei ratio constraint a bassa priorità

Oppure

1. Minimizzare il numero di cambi di colore
2. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità

Per le due sopracitate ottimizzazioni multi-obiettivo, c'è solo un hard constraint: la limitazione alla dimensione massima di una sequenza di auto dello stesso



colore; infatti le pistole spray utilizzate per la verniciatura devono essere lavate regolarmente, anche se non ci sono cambi di colore.

Non ci deve essere nessuna compensazione tra gli obiettivi: l'ottimizzazione dell'obiettivo con posizione O+1 non può diminuire il risultato dell'ottimizzazione dell'obiettivo con posizione O.

### **1.3 Dettagli sul calcolo del numero di violazioni dei ratio constraint**

La miglior soluzione per un ratio constraint N/P sarebbe un giorno di produzione schedato in cui ci siano al massimo N veicoli vincolati in una sequenza qualsiasi di P veicoli consecutivi. La miglior soluzione per un vincolo di rapporto 1/P sarebbe un giorno di produzione in cui qualsiasi coppia di auto vincolate sia separata da almeno  $P - 1$  vetture non vincolate: per esempio, X \_ \_ \_ X è una sequenza accettabile per un ratio constraint 1/4, dove 'X' è un'auto associata alla limitazione in esame, mentre '\_' è un veicolo non considerato dal ratio constraint.

Se non fosse possibile generare una soluzione priva di violazioni dei vincoli di rapporto, è necessario che le vetture che violano queste limitazioni, vengano disposte in modo uniforme all'interno dell'intera sequenza di produzione del giorno in esame. Il proposito è di cercare di limitare il più possibile il carico di lavoro nella fase di assemblaggio.

Per ottenere questa distribuzione di veicoli vincolati, è utile calcolare il numero delle violazioni, sulle limitazioni di rapporto, per ogni sequenza generata. Naturalmente, minore sarà la distanza tra le vetture che richiedono montaggi particolari, maggiori risulteranno le violazioni ed è necessario considerare che ogni vettura che richiede operazioni di montaggio speciali, ripercuoterà la sua violazione all'interno di P intervalli.

Esempio: ratio constraint 1/5 => il numero di violazioni del ratio constraint è calcolato su sequenze di 5 veicoli consecutivi

\_ \_ \_ X \_ \_ \_ X : 1 violazione nella sequenza \_ \_ \_ **X** \_ \_ \_ **X**  
 \_ \_ \_ X \_ \_ X \_ : totale di 2 violazioni sulle sequenze consecutive \_ \_ \_ **X** \_ \_ **X** \_ e  
 \_ \_ \_ **X** \_ \_ **X** \_

Il numero di violazioni di un ratio constraint su una qualsiasi sequenza è calcolato secondo la formula 1:

= (numero di veicoli associati al ratio constraint nella sequenza) – (numeratore del ratio constraint) se (numero di veicoli associati al ratio constraint nella sequenza) > (numeratore del ratio constraint)  
 = 0 altrimenti □

Questo calcolo del numero di violazioni si applica a ratio constraint 1/P e ratio constraint N/P (con  $N > 1$ ).

Esempio: ratio constraint 1/5

\_ \_ X X \_ \_ X \_ : ci sono 2 violazioni nella sequenza **X** **X** \_ \_ **X**: 3 (veicoli vincolati) – 1 (numeratore del ratio constraint).

L'obiettivo dell'ottimizzazione è quello di minimizzare il numero totale di violazioni dei ratio constraint attraverso tutte le sequenze consecutive di un giorno di produzione.

Schedulando la produzione del giorno D, gli ultimi veicoli schedulati il giorno di produzione D – 1 devono essere presi in considerazione per il calcolo del numero di violazioni dei ratio constraint del giorno in esame. Questo implica che le sequenze consecutive (sulle quali il numero di violazioni dei ratio constraint è calcolato) devono iniziare il giorno di produzione D – 1.

Per un ratio constraint N/P, la prima sequenza contiene gli ultimi P – 1 veicoli schedulati il giorno di produzione D – 1 ed il primo veicolo schedulato il giorno di produzione D. Si deve ricordare che la sequenza di veicoli del giorno di

produzione  $D - 1$  è già stata fissata, quindi la loro posizione non può essere cambiata.

Esempio: la prima sequenza per un ratio constraint 1/5

XXXXXXY \_ \_ \_ \_

X: ultimi veicoli schedulati il giorno di produzione  $D - 1$

Y: prima auto schedulata il giorno di produzione D

Dato che la produzione del giorno  $D + 1$  è ignorata durante la generazione della sequenza di autovetture per il giorno di produzione D, per un ratio constraint N/P, l'ultima sequenza consecutiva del giorno di produzione D contiene le ultime P auto schedulate il giorno di produzione D. Questa ultima sequenza consecutiva inizia nella posizione  $L - P$ , con L pari al numero di veicoli del giorno di produzione D ed una posizione compresa tra 0 ed  $L - 1$ .

Ciò significa che, per ogni ratio constraint, si devono calcolare le violazioni determinate dalle ultime auto del giorno di produzione D, come se le prime auto della sequenza del giorno  $D + 1$  non richiedano operazioni extra in fase di assemblaggio.

Per un ratio constraint N/P, le ultime sequenze consecutive avranno una lunghezza compresa tra  $P - 1$  e  $N + 1$ . Se in una di queste sequenze consecutive, il numero di auto associate al ratio constraint è strettamente maggiore di N, allora ci saranno delle violazioni, sia che il primo veicolo schedulato il giorno di produzione  $D + 1$  sia associato al ratio constraint o no.

Questo metodo di calcolo consente di dare lo stesso peso alle violazioni dovute alle ultime  $P - 1$  auto del giorno di produzione D (per un ratio constraint N/P) ed a quelle dovute ai veicoli precedenti.

Esempio: le ultime sequenze consecutive per un ratio constraint 1/5

YYYYYYYY

YYYYYYYY

YYYYYYY~~YY~~

Y: ultimi veicoli schedulati il giorno di produzione D

#### **1.4 Dettagli sul calcolo del numero di cambi di colore**

I requisiti sulla verniciatura impongono che le pistole spray siano lavate (1) dopo ogni cambio di colore e (2) regolarmente dopo ogni sequenza di auto consecutive dello stesso colore con una dimensione pari al limite M imposto. Il valore M è un hard constraint ed impone il lavaggio delle pistole spray ogni M veicoli consecutivi con lo stesso colore.

Se la vettura successiva alla vettura M ha un colore differente viene conteggiato un solo cambio di colore, che può essere riferito indistintamente ad una delle due condizioni.

Se si presenta una sequenza di X auto consecutive con lo stesso colore, con  $X > M$ , il numero di cambi di colore sarà dato da  $X / M$  arrotondato per difetto.

Esempio: limitazione  $M = 3$

La sequenza di colori 1 1 1 2 2 2 3 3 3 3 4 4 determina 4 cambi di colore: 3 dovuti al passaggio ad un colore differente ed 1 dovuto al superamento del limite M nella sequenza 3 3 3 3.

## 2. Input

### 2.1 Descrizione dei test set

Un test set è rappresentato da un insieme di files che descrivono differenti scenari. Il programma sviluppato carica i files corrispondenti allo scenario specificato e procede all'opportuna ottimizzazione in base alle caratteristiche specifiche dello scenario stesso.

La Renault ha fornito 3 differenti tese set, da utilizzare per lo sviluppo e l'ottimizzazione del software.

#### 2.1.1 Test set

Come soprariportato 3 test set sono stati forniti dalla Renault dando la possibilità di ottimizzare il funzionamento dell'algoritmo, in particolare sono stati rilasciati:

- Test set A: disponibile sin dall'inizio del concorso. La giuria utilizza questi dati per la scelta dei finalisti. Questo è il test set di qualificazione.
- Test set B: fornito ai finalisti per mettere a punto i loro programmi per il test set X.
- Test set X: utilizzato per classificare i finalisti alla conferenza Roadef 2005. Questo è il test set finale. Il test set X è sconosciuto ai finalisti fino alla fine del concorso.

La messa a punto del programma dove migliorare la robustezza dell'algoritmo nei confronti di dati variabili. Questa caratteristica è critica per applicazioni di schedulazione industriale. Non è immaginabile una messa a punto dei programmi ogni volta che ci sono grossi cambiamenti nei dati di produzione (ratio constraint, caratteristiche dei veicoli, ecc.) in una qualsiasi fabbrica Renault.

### **2.2.2 Contenuto di un test set**

Un test set contiene un insieme di scenari, ognuno dei quali rappresenta i dati di produzione per una fabbrica. La stessa fabbrica può essere utilizzata per produrre diversi scenari.

Uno scenario contiene:

- Ratio constraint ad alta priorità
- Ratio constraint a bassa priorità
- Limite al numero di auto consecutive con lo stesso colore
- Veicoli del giorno di produzione  $D$  e ultime auto schedate del giorno di produzione  $D - 1$  (ci sono  $\text{DenMax} - 1$  veicoli del giorno di produzione  $D - 1$ , con  $\text{DenMax}$  il massimo dei denominatori dei ratio constraint)
- Un veicolo è identificato da: un ID, la data del giorno di produzione (data  $D$  o data  $D - 1$ ), il colore della verniciatura ed un flag (0/1) per ogni ratio constraint, che indica se il veicolo è associato o meno al vincolo, ed il numero di sequenza schedato dall'applicazione industriale Renault
- L'ordine degli obiettivi di ottimizzazione

Alcuni scenari possono non contenere nessun ratio constraint di bassa priorità in quanto alcune fabbriche possono dare alta priorità a tutti i ratio constraint oppure non avere ratio constraint a bassa priorità.

## 2.2 Formato dei file

Uno scenario è una directory chiamata `instance_XXXXXX` contenente i seguenti files:

- file della limitazione delle sequenze di auto con lo stesso colore <<paint\_batch\_limit.txt>>
- file degli obiettivi di ottimizzazione <<optimization\_objectives.txt>>
- file dei ratio constraint <<ratios.txt>>
- file dei veicoli <<vehicles.txt>>

Essi sono tutti files di testo ASCII che hanno il carattere ‘;’ come delimitatore.

### 2.2.1 File della limitazione delle sequenze di auto con lo stesso colore

Questo file contiene il limite alla dimensione di sequenze di auto consecutive con lo stesso colore, ossia ogni quante auto consecutive dello stesso colore si deve procedere al lavaggio delle pistole spray con il solvente.

Esempio del contenuto di un file <<paint\_batch\_limit.txt>>

*limitation;*

*10;*

### 2.2.2 File degli obiettivi di ottimizzazione

Il file contiene l’ordine degli obiettivi di ottimizzazione, dall’obiettivo con il più alto livello di priorità a quello con il livello di priorità più basso. Il file contiene una lista di valori del tipo <<posizione; nome dell’obiettivo>>. Il nome

dell'obiettivo indica il tipo (cambi di colore o ratio constraint) e se i ratio constraint ad alta priorità sono facili da soddisfare o difficili da soddisfare.

Diversi esempi del contenuto del file *<<optimization\_objectives.txt>>*:

Contenuto del file nel caso di massima priorità ai ratio constraint e ratio constraint ad alta priorità facili da soddisfare:

```
rank;objective name;  
1;high_priority_level_and_easy_to_satisfy_ratio_constraints;  
2;paint_color_batches;  
3;low_priority_level_ratio_constraints;
```

Oppure

```
rank;objective name;  
1;high_priority_level_and_easy_to_satisfy_ratio_constraints;  
2;low_priority_level_ratio_constraints;  
3;paint_color_batches;
```

Oppure

```
rank;objective name;  
1;high_priority_level_and_easy_to_satisfy_ratio_constraints;  
2;paint_color_batches;
```

Contenuto del file in caso di massima priorità ai cambi di colore e ratio constraint ad alta priorità facili da soddisfare.

```
rank;objective name;  
1;paint_color_batches;  
2;high_priority_level_and_easy_to_satisfy_ratio_constraints;
```



*3;low\_priority\_level\_ratio\_constraints;*

Oppure

*rank;objective name;*

*1;paint\_color\_batches;*

*2;high\_priority\_level\_and\_easy\_to\_satisfy\_ratio\_constraints;*

Contenuto del file in caso di massima priorità ai ratio constraint e ratio constraint ad alta priorità difficili da soddisfare

*rank;objective name;*

*1;high\_priority\_level\_and\_difficult\_to\_satisfy\_ratio\_constraints;*

*2;paint\_color\_batches;*

*3;low\_priority\_level\_ratio\_constraints;*

Oppure

*rank;objective name;*

*1 ;high\_priority\_level\_and\_difficult\_to\_satisfy\_ratio\_constraints;*

*2;paint\_color\_batches;*

Oppure

*rank;objective name;*

*1;high\_priority\_level\_and\_difficult\_to\_satisfy\_ratio\_constraints;*

*2;low\_priority\_level\_ratio\_constraints;*

*3;paint\_color\_batches;*

### 2.2.3 File dei ratio constraint

Il file descrive ratio constraint di alta priorità e bassa priorità ognuno dei quali è definito da un rapporto N/P (N numero massimo di vetture associato al vincolo di rapporto su una qualsiasi sequenza di P veicoli), da un flag per il livello di priorità (1=alta priorità, 0=bassa priorità) e da un identificatore (la posizione del ratio constraint).

Esempio del contenuto del file <<ratios.txt>>

```
Ratio;Prio; Ident;  
1/2; 1; HPRC1;  
1/8; 1; HPRC2;  
2/3; 1; HPRC3;  
1/4; 1; HPRC4;  
3/4; 0; LPRC1;  
1/3; 0; LPRC2;  
2/3; 0; LPRC3;
```

In questo esempio ci sono quattro ratio constraint ad alta priorità e tre a bassa priorità.

## 2.2.4 File dei veicoli

Il file contiene i veicoli del giorno di produzione D e le ultime auto schedate il giorno di produzione D – 1.

Per ogni veicolo sono presenti i seguenti campi:

1. *Data*: il giorno di produzione del veicolo, col formato “AAAA WW D”, dove AAAA è l’anno, WW è la settimana industriale nell’anno (WW = 1..52), e D è il giorno della settimana (1 per lunedì, 2 per martedì, ecc.)
2. *Numero di sequenza*: la posizione del veicolo nel giorno di produzione. Questa posizione è definita dall’applicazione industriale Renault. Le posizioni in un giorno di produzione iniziano dal numero 1.
3. *ID*: identificativo del veicolo.
4. *Colore*: codice del colore della verniciatura del veicolo.
5. *HPRCi*: ‘1’ se il veicolo è associato al ratio constraint ad alta priorità HPRCi, ‘0’ altrimenti. Il numero di colonne HPRCi è uguale al numero di ratio constraint ad alta priorità.
6. *LPRCi*: la stessa situazione, ma con ratio constraint a bassa priorità.

Solo gli ultimi veicoli schedati il giorno di produzione D – 1 sono forniti in quanto solo queste ultime auto sono necessarie per calcolare il numero di violazioni dei ratio constraint nel giorno di produzione D. Il numero di veicoli del giorno di produzione D – 1 è uguale al massimo dei denominatori dei ratio constraint – 1.

Esempio del contenuto del file <<vehicles.txt>>

```
Date;SeqRank;Ident;Paint Color;HPRC1; HPRC2;
HPRC3;HPRC4;LPRC1;LPRC2; LPRC3
2003 18 3;997;025031830079;5;1;0;0;0;0;0;0
2003 18 3;998;025031850339;2;0;0;0;0;1;0;0
2003 18 3;999;025031920802;1;1;0;1;1;1;1;1
```

2003 18 3;1000;025031830269;1;0;0;1;0;1;0  
2003 18 3;1001;025031850094;2;0;0;0;0;0;0  
2003 18 3;1002;025031910010;2;1;1;0;0;1;0  
2003 18 3;1003;025031850332;2;0;0;0;0;1;0  
2003 18 3;1004;025031820317;1;1;0;1;1;1;1  
2003 18 5;1;025031910825;3;0;0;1;0;1;0;1;0  
2003 18 5;2;025031820293;7;1;0;1;1;1;0;1;0  
2003 18 5;3;025031850497;2;0;0;0;0;0;0;0;0  
2003 18 5;4;025031911085;5;1;0;1;0;1;1;1;0  
2003 18 5;5;025031850299;5;0;0;1;0;1;0;1;0  
2003 18 5;6;025031850082;2;1;0;0;0;0;0;0;0  
2003 18 5;7;025031850783;5;0;0;1;0;1;1;1;0  
2003 18 5;8;025031920415;5;1;1;1;1;0;0;1;0  
2003 18 5;9;025032010270;5;0;0;0;0;0;0;0;0  
2003 18 5;10;025031911083;5;1;0;1;0;1;1;1  
2003 18 5;11;025031910060;5;0;0;0;0;0;0;0;1  
2003 18 5;12;025031970126;5;1;0;1;1;0;0;1  
2003 18 5;13;025031910182;2;0;0;0;0;1;0;0  
2003 18 5;14;025031850845;1;1;0;1;0;1;1;1  
2003 18 5;15;025031910373;1;0;0;1;0;0;0;1  
2003 18 5;16;025031850597;1;1;0;0;0;1;0;0  
2003 18 5;17;025031830304;1;0;0;1;1;1;0;1  
2003 18 5;18;025031850802;1;1;0;1;0;1;1;1  
2003 18 5;19;025031850666;2;0;0;0;0;0;0;0  
2003 18 5;20;025031850746;2;1;0;0;0;1;0;0  
2003 18 5;21;025031850856;2;0;0;0;0;1;0;0  
2003 18 5;22;025031830202;9;1;0;1;1;1;0;1  
2003 18 5;23;025031850767;2;0;0;0;0;0;0;0

Nell'esempio riportato, ci sono 4 ratio constraint ad alta priorità, 3 ratio constraint a bassa priorità, 8 veicoli del giorno di produzione D – 1 e 23 auto del giorno di produzione D.

Data del giorno di produzione D – 1 <<2003 18 3>>

Data del giorno di produzione D <<2003 18 5>>

Come dimostrato nell'esempio, le date dei giorni di produzione D e D – 1 non sono sempre consecutive. Esse riguardano esclusivamente i giorni lavorativi nelle fabbriche, senza domeniche e festività. Nell'esempio la data <<2003 18 4>> è il 1° Maggio 2003 che è una festività.

Per ogni veicolo, i campi dei ratio constraint ad alta priorità sono definiti prima dei campi dei ratio constraint a bassa priorità. La sequenza dei campi appartenenti alle limitazioni di rapporto è identica alla sequenza dei ratio constraint nel file *<<ratios.txt>>*.

I veicoli del giorno di produzione  $D - 1$  sono definiti prima delle auto del giorno di produzione  $D$ .



### **3. Tecniche risolutive**

#### **3.1 Il problema del tempo**

La scelta delle tecniche risolutive è stata fortemente influenzata dalla limitazione imposta sul tempo di esecuzione del programma. Infatti, come descritto successivamente, l'algoritmo verrà eseguito su una macchina di riferimento (P4 1.6Ghz / 1Gb Ram) per un periodo di tempo pari a 600 secondi. Per far fronte a questo vincolo abbiamo implementato soluzioni che consentono un'esecuzione rapida dell'ottimizzazione ed anche le strutture dati di supporto sono state definite in modo tale da permetterne un utilizzo ed un aggiornamento rapido.

Anche la scelta del linguaggio in cui il software è stato implementato, è stata vincolata dalla limitazione di tempo imposta.

Il programma è infatti stato implementato utilizzando il linguaggio C che, se da un lato complica la scrittura del codice, dall'altro offre un'elevata velocità di esecuzione essendo un linguaggio particolarmente efficiente, quasi a basso livello. Come ambiente di sviluppo, essendo vietati prodotti commerciali, abbiamo utilizzato Dev-C++ prodotto dalla software house Bloodshed. Questo software si basa sul compilatore gratuito GCC 3.2 che è stato utilizzando sfruttandone le opzioni di ottimizzazione del codice.

### 3.2 Diagramma di flusso globale

Il funzionamento generale del software può essere rappresentato mediante il diagramma di flusso 1 che riportiamo di seguito:

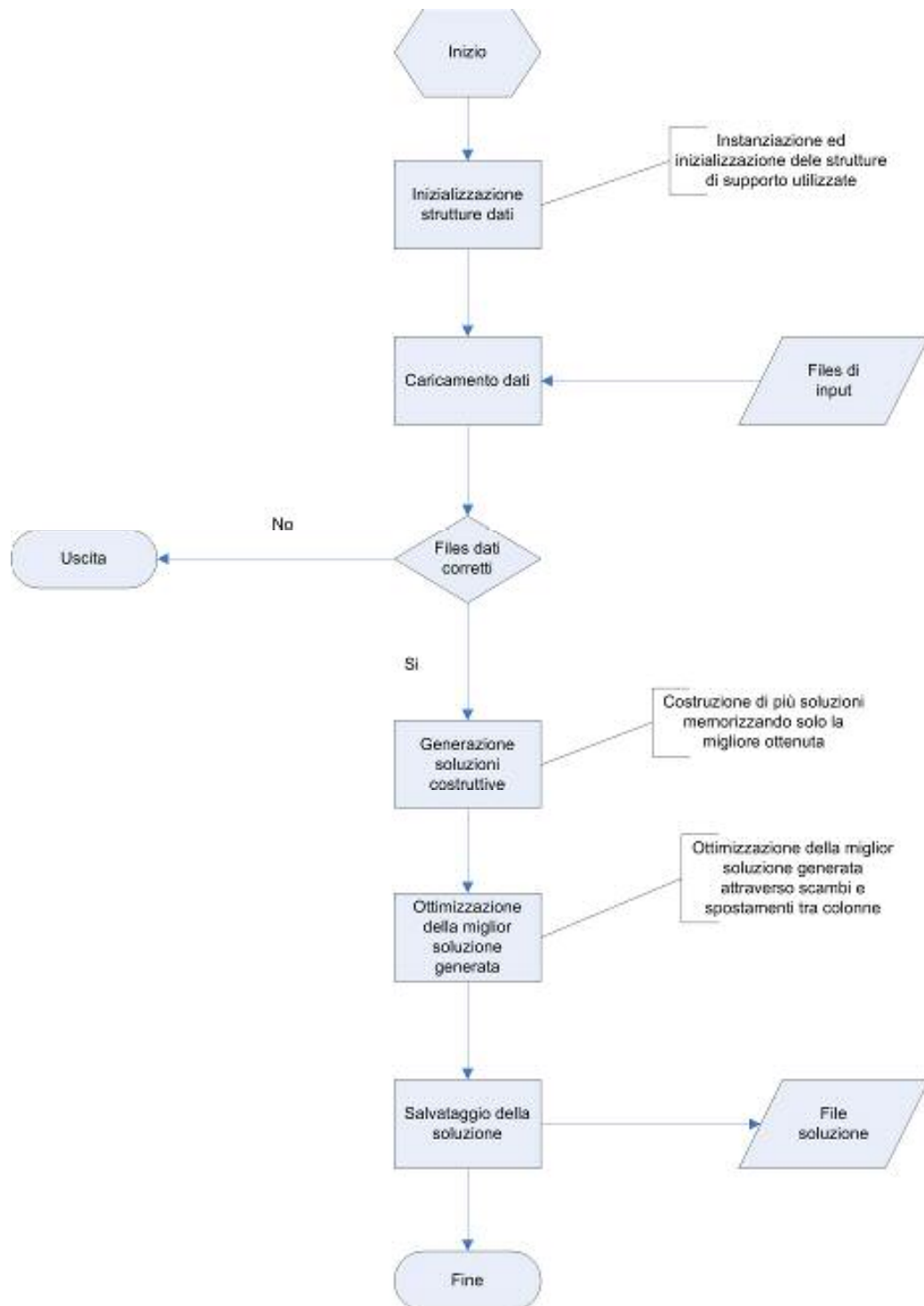


Diagramma 1: Funzionamento generale



Analizzando il diagramma soprariportato si nota la suddivisione del software in quattro parti fondamentali:

- Inizializzazione e caricamento dati
- Generazione di molteplici soluzioni con tecnica costruttiva
- Ottimizzazione della miglior soluzione generata mediante ricerca locale
- Salvataggio della soluzione

### 3.2.1 Inizializzazione e caricamento dati

La fase iniziale è rappresentata dalla definizione ed inizializzazione delle strutture utilizzate, nonché dal caricamento dei dati. Il funzionamento del software in questo primo step può essere rappresentato come segue (diagramma 2):

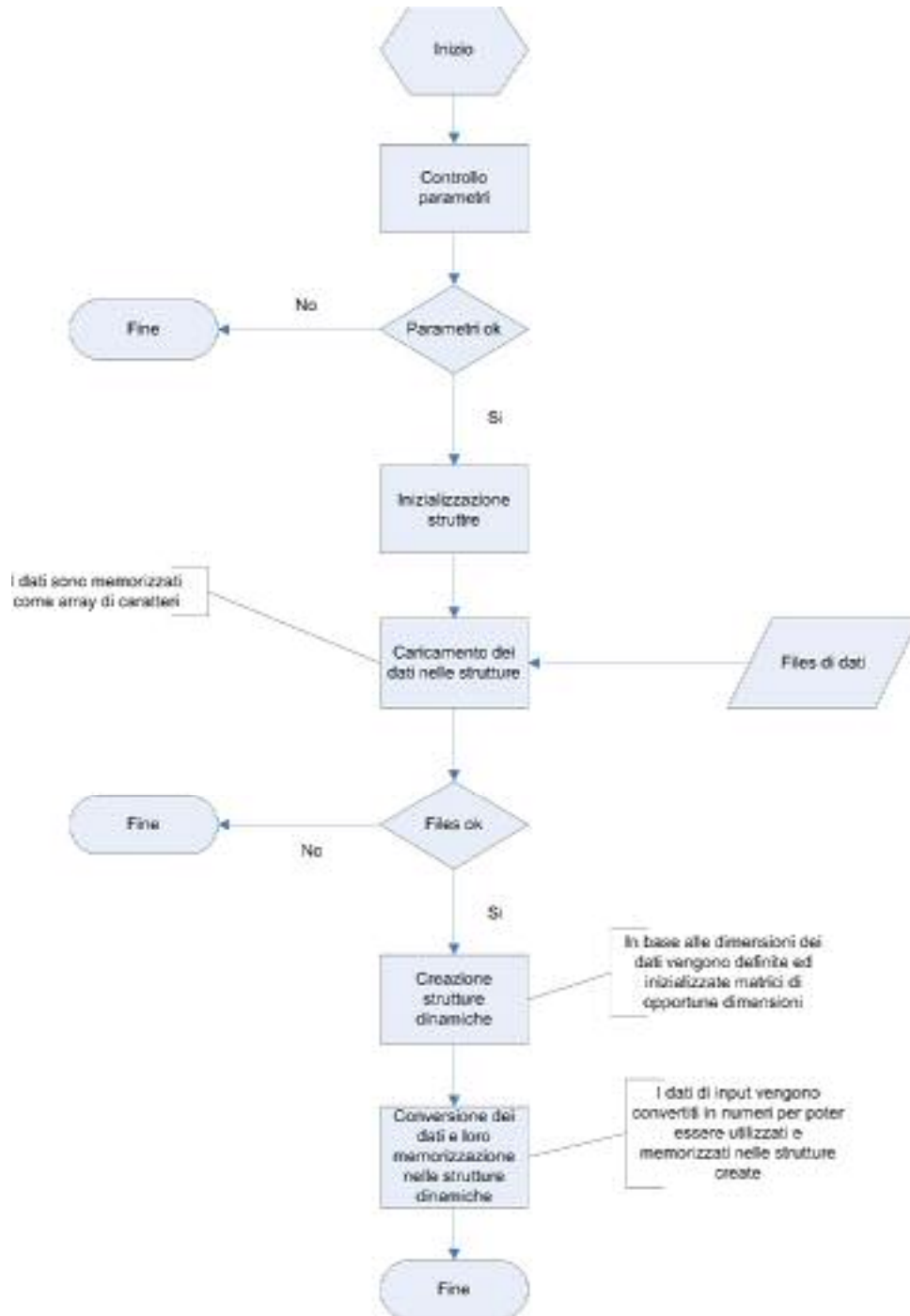


Diagramma 2: Inizializzazione e caricamento dati

### 3.2.2 Funzionamento

L'analisi del diagramma mostra il funzionamento iniziale del software. Infatti il primo passo effettuato consiste nell'effettuare il controllo dei parametri di input, ciò avviene in quanto deve essere possibile specificare il nome dello scenario i cui dati saranno processati nonché il tempo massimo di elaborazione ammesso. Quest'ultimo parametro è opzionale, in quanto se omissso è utilizzato il valore di default del timeout pari a 600 secondi (10 minuti). L'inserimento di parametri errati determina l'immediata uscita dal software senza eseguire alcuna elaborazione.

Se i parametri inseriti sono corretti il programma procede alla fase successiva rappresentata dalla lettura dei files di dati relativi allo scenario specificato. Come riportato in precedenza i dati sono contenuti in quattro files di testo ASCII, ognuno dei quali viene letto e memorizzato in opportune matrici costituite da array di caratteri.

Anche in questa fase la presenza di files il cui percorso, o nome, inserito non risulta corretto determina l'interruzione del programma e la comunicazione del file di input errato.

Sulla base delle dimensioni dei files di input si creano dinamicamente le strutture dati necessarie all'elaborazione rappresentate da array e matrici di numeri interi. Dopo la creazione, avvenuta mediante la primitiva *calloc*, ogni elemento di queste strutture è inizializzato con un valore convenzionale '- 1', in quanto questa cifra non sarà mai presente nei files di input. I dati letti in precedenza e memorizzati come caratteri sono quindi convertiti in numeri interi, attraverso il comando *atoi*, e salvati nelle strutture dinamiche create.

### 3.2.3 Strutture dati

I dati sono rappresentati da quattro files di testo ASCII ognuno dei quali contenente informazioni che sono memorizzate in una differente struttura dati. In base al file di dati di input si possono distinguere le strutture necessarie che sono descritte di seguito:

- <<vehicles.txt>>: fornisce le informazioni relative alle vetture, i dati sono memorizzati nelle seguenti matrici:
  - ❖ *Matrice*: struttura rappresentata da una matrice in cui ogni posizione è a sua volta un array di caratteri. I dati letti inizialmente sono salvati in questa struttura prima di essere convertiti in numeri.

<i>Auto 1</i>	<i>Auto 2</i>	<i>Auto 3</i>	<i>Auto 4</i>	.....
Day	Day	Day	Day	.....
ID	ID	ID	ID	.....
N°	N°	N°	N°	.....
Colore	Colore	Colore	Colore	.....
Flag Opt. 1	Flag Opt. 1	Flag Opt. 1	Flag Opt. 1	.....
Flag Opt. 2	Flag Opt. 2	Flag Opt. 2	Flag Opt. 2	.....
Flag Opt. 3	Flag Opt. 3	Flag Opt. 3	Flag Opt. 3	.....
.....	.....	.....	.....	.....

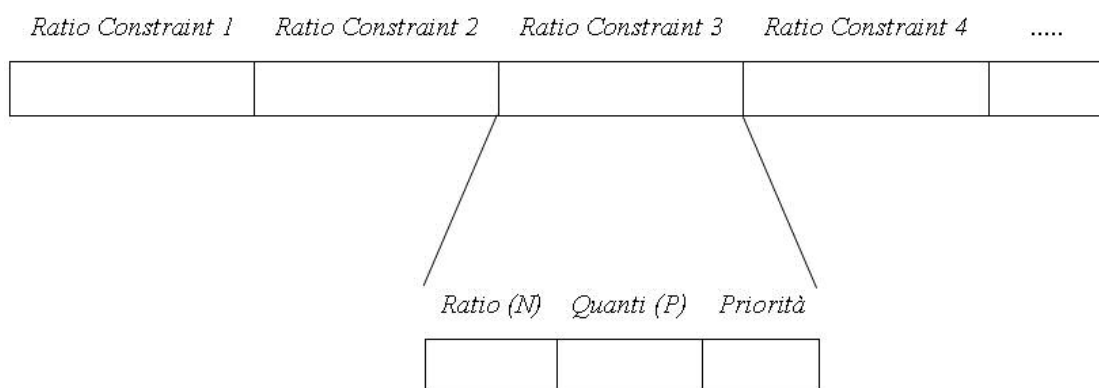
- ❖ *Matrice Ieri*: per ogni veicolo appartenente al giorno di produzione D – 1 sono salvati ID, Colore e valori relativi agli optional.

<i>Auto 1</i>	<i>Auto 2</i>	<i>Auto 3</i>	<i>Auto 4</i>	.....
ID	ID	ID	ID	.....
Colore	Colore	Colore	Colore	.....
Flag Opt. 1	Flag Opt. 1	Flag Opt. 1	Flag Opt. 1	.....
Flag Opt. 2	Flag Opt. 2	Flag Opt. 2	Flag Opt. 2	.....
Flag Opt. 3	Flag Opt. 3	Flag Opt. 3	Flag Opt. 3	.....
.....	.....	.....	.....	.....

- ❖ *Matrice Ridotta*: stesse informazioni di matrice ieri ma riferite ai veicoli del giorno di produzione D.

<i>Auto 1</i>	<i>Auto 2</i>	<i>Auto 3</i>	<i>Auto 4</i>	.....
ID	ID	ID	ID	.....
Colore	Colore	Colore	Colore	.....
Flag Opt. 1	Flag Opt. 1	Flag Opt. 1	Flag Opt. 1	.....
Flag Opt. 2	Flag Opt. 2	Flag Opt. 2	Flag Opt. 2	.....
Flag Opt. 3	Flag Opt. 3	Flag Opt. 3	Flag Opt. 3	.....
.....	.....	.....	.....	.....

- <<paint\_batch\_limit.txt>>: contiene la limitazione alla dimensione massima di una sequenza di auto consecutive senza che questa richieda il lavaggio delle pistole spray. Il valore fornito è convertito in numero intero e memorizzato in una semplice variabile intera chiamata *limit\_brush*.
- <<ratios.txt>>: il file specifica i ratio constraint di tipo N/P per ognuno dei quali sono indicati i valori di N e P nonché la priorità. Queste informazioni sono memorizzate in un array chiamato *ratios* in cui ogni posizione è composta da una struttura dati contenente:
  - ❖ *Ratio*: campo intero corrispondente al valore P
  - ❖ *Quanti*: il valore N, anch'esso di tipo intero
  - ❖ *Priorità*: indica la priorità del ratio constraint. Un valore pari a '1' rappresenta un ratio constraint ad alta priorità mentre lo '0' coincide con ratio constrain a bassa priorità.



- <<optimization\_objectives.txt>>: specifica quali sono gli obiettivi di ottimizzazione nonché l'ordine in cui questi obiettivi devono essere raggiunti. Ciò è memorizzato in un vettore chiamato *pesi* formato da 3 numeri interi che indicano il peso di ciascun obiettivo secondo il metodo di valutazione riportato in seguito. In particolare il primo obiettivo avrà peso 10000, il secondo 100 mentre il terzo, se presente, 1. La corrispondenza tra tipo di obiettivo e posizione nell'array *pesi* è la seguente:

- ❖ *Pesi*[0] → cambi di colore
- ❖ *Pesi*[1] → ratio constraint ad alta priorità
- ❖ *Pesi*[2] → ratio constraint a bassa priorità

*Pesi*

<i>Posizione</i>	<i>Contenuto</i>
[0]	Peso dell'obiettivo cambi di colore
[1]	Peso dei ratio constraint ad alta priorità
[2]	Peso dei ratio constraint a bassa priorità

## 4. Tecniche Costruttive

La prima parte dell'implementazione si occupa di costruire, partendo dai dati precedentemente letti, delle soluzioni ammissibili ed ottimizzate sulla base della priorità degli obiettivi. Ciò avviene mediante tecniche greedy in grado di fornire un gran numero di soluzioni ammissibili in breve tempo effettuando scelte sempre ottime senza considerare gli effetti delle decisioni prese. Questa soluzione dovrebbe consentire di avvicinarsi all'ottimo globale della funzione che sarà raggiunto successivamente dalla seconda parte del software che sfrutta la ricerca locale.

La sequenza di vetture è generata mediante due differenti tecniche:

1. Generazione soluzione per righe
2. Generazione soluzione per colonne

Inizialmente il programma calcola, per ogni ratio constraint, la densità delle vetture che necessitano di questo particolare accessorio, ordinandole in modo decrescente. Successivamente l'algoritmo procede, posizione per posizione, alla determinazione della vettura, scelta tra quelle disponibili, che meglio si adatta alla locazione in esame della sequenza finale. La scelta dell'auto avviene eliminando, da alcune strutture di supporto ausiliarie, tutte le automobili che causano una o più violazioni degli obiettivi nella posizione in esame in quanto non corrispondono alle scelte effettuate.

La selezione della vettura da fissare nella sequenza finale schedulata per il giorno di produzione D avviene sfruttando due diversi metodi; il primo stabilisce, vincolo

per vincolo, quali siano i valori che le vetture devono possedere per poter essere collocate in una determinata posizione all'interno della sequenza finale. Il secondo, invece, determina univocamente le caratteristiche ideali, in termini di ratio constraint e colori, che un'automobile dovrebbe possedere perché si possa generare una sequenza senza alcun tipo di violazioni. Stabilita l'automobile con le caratteristiche ideali, essa viene ricercata all'interno dell'insieme di quelle da produrre individuando la vettura che più assomiglia a quella desiderata in termini di requisiti, e la si pone direttamente nella posizione in esame.

Questo secondo metodo produce normalmente soluzioni con un costo globale inferiore in quanto analizza tutti i vincoli in modo diretto decidendo, posizione dopo posizione, le vetture compatibili, e fissando successivamente quella migliore.

In entrambi i procedimenti, l'algoritmo si adatta alla priorità degli obiettivi, in particolare distinguendo i casi in cui la massima priorità va alla minimizzazione del numero di cambi di colore da quelli in cui la massima priorità va alla minimizzazione del numero di violazioni dei ratio constraint. Se l'obiettivo primario riguarda i vincoli sulla linea di assemblaggio, il software cerca di non inserire violazioni sui ratio constraint, preoccupandosi delle tinte delle vetture solo al momento di decidere quale specifica auto porre in una certa posizione, cercando tra tutte le possibili quella, se presente, con il medesimo colore della vettura precedente.

Qualora il primo vincolo fosse la limitazione del numero totale di lavaggi delle pistole, invece, il software esegue come primo passo la generazione di una sequenza di colori da rispettare nella costruzione della soluzione finale. Questa sequenza deve possedere il numero minimo possibile di cambi di tonalità. Per raggiungere questo obiettivo, in ogni posizione, vengono tenute in considerazione le sole vetture del colore imposto dalla sequenza generata per tale locazione, eliminando tutte quelle che possiedono un colore differente.



L'algoritmo descritto è in grado di fornire soluzioni in un breve periodo di tempo (1-2 minuti sulla macchina di riferimento), soprattutto nel caso in cui l'obiettivo primario è rappresentato dalla minimizzazione del numero di cambi di colore. Tuttavia questa rapidità di esecuzione si paga dal punto di vista della qualità delle soluzioni generate che non sempre è buona. Questo in quanto l'algoritmo è sensibile alle caratteristiche del particolare scenario considerato, infatti in alcuni casi il miglioramento raggiunto con tecnica greedy rispetto alla soluzione iniziale è ridotto, mentre in altri la soluzione ottenuta è quasi ottima tanto che il successivo passo di ricerca locale riesce a diminuire il costo della soluzione generata solo in modo marginale.



## **5. Ricerca locale**

### **5.1 Introduzione**

La seconda parte del software è rappresentata da algoritmi di ricerca locale, applicati alla migliore soluzione ottenuta mediante tecnica costruttiva. La ricerca locale appartiene alle tecniche definite euristiche, ossia algoritmi che non garantiscono di ottenere la soluzione ottima, ma che in generale sono in grado di fornire una buona soluzione del problema.

Normalmente gli algoritmi euristici hanno una bassa complessità, ma, in caso di problemi di grandi dimensioni o strutture complesse, può essere necessario sviluppare euristiche sofisticate o ad alta complessità. Inoltre è possibile, in generale, che un'euristica non sia in grado di fornire alcuna soluzione ammissibile al problema, anche senza dimostrare che non esistono.

Una tecnica utilizzata per la realizzazione di algoritmi euristici di ottimizzazione è la ricerca locale che si basa su un'idea molto semplice ed intuitiva: data una soluzione ammissibile, si esaminano le soluzioni ad essa vicine, in cerca di una soluzione migliore (tipicamente, con miglior valore della funzione obiettivo); se tale soluzione viene trovata essa diventa la nuova “soluzione corrente” ed il procedimento viene iterato, altrimenti, quando nessuna delle soluzioni vicine è migliore di quella “corrente”, l'algoritmo termina avendo determinata un ottimo locale.

Elemento caratterizzante l'algoritmo è la definizione di vicinanza tra le soluzioni. Il funzionamento degli algoritmi di ricerca locale può essere schematizzato come segue:

Inizio

$X$  = soluzione ammissibile

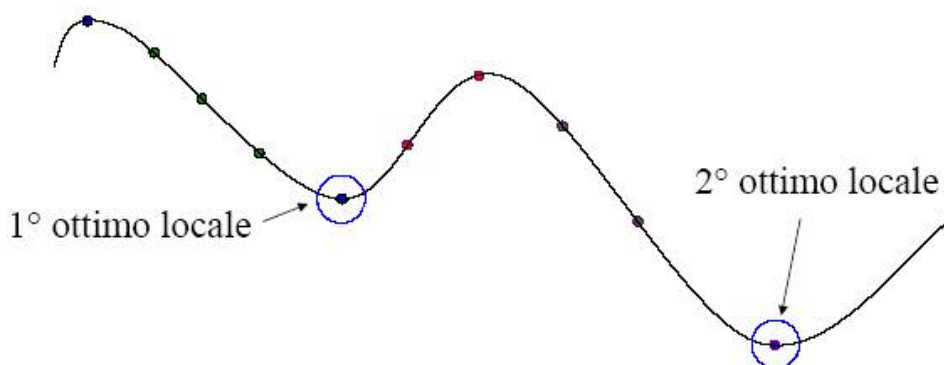
Finchè (Cerca soluzione migliore( $X$ )  $\neq X$ )

$X$  = Cerca soluzione migliore ( $X$ )

Fine

Come è possibile notare l'algoritmo necessita di una soluzione ammissibile da cui partire; essa può essere determinata mediante algoritmi di tipo greedy che generano un gran numero di soluzioni ammissibili.

In generale, la soluzione fornita da un algoritmo di ricerca locale non è ottima per il problema, ma solamente un ottimo locale relativamente all'intorno scelto per la determinazione delle soluzioni "vicine". La situazione può essere efficacemente esplicata nel modo seguente:



La qualità dell'ottimo locale raggiunto dipende in larga misura dalla soluzione iniziale. Per raggiungere un ottimo globale sono necessari alcuni accorgimenti, quello utilizzato consiste nel determinare la soluzione iniziale per la ricerca locale mediante euristiche costruttive basate su algoritmi greedy.

L'obiettivo di questa scelta è di ottenere una soluzione buona che sia “vicino” all'ottimo globale che potrà essere successivamente raggiunto dall'algoritmo di ricerca locale.

## 5.2 Diagramma a blocchi della ricerca locale

Il funzionamento dell'algoritmo di ricerca locale implementato può essere rappresentato sinteticamente col diagramma 3 che segue:

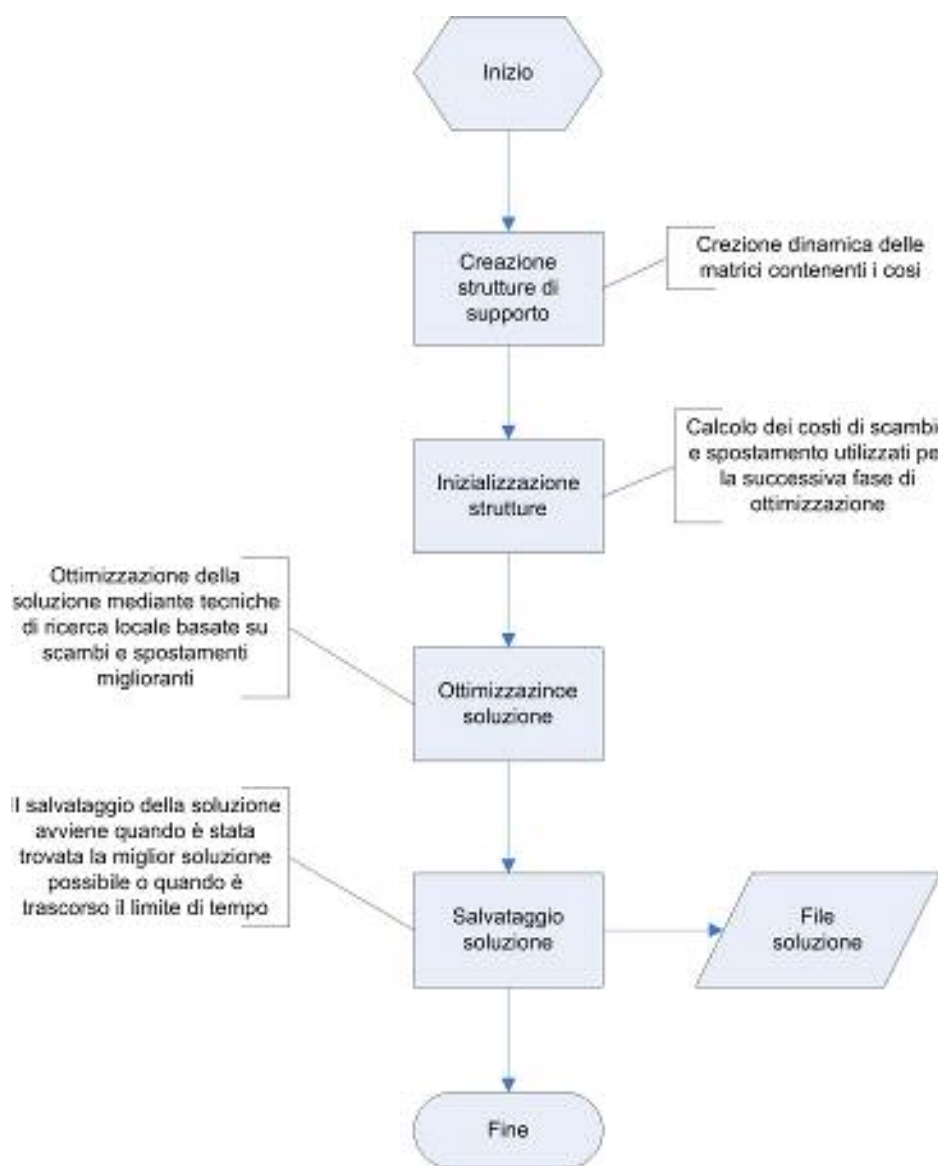


Diagramma 3 - Ricerca locale

### 5.2.1 Funzionamento generale ricerca locale

L'algoritmo di ricerca locale si applica alla miglior soluzione generata, nello step precedente, mediante euristiche costruttive. Ciò con lo scopo, come esposto sopra, di avvicinarsi ad un ottimo globale ed evitare di cadere in un ottimo locale.

Il primo passo compiuto dall'algoritmo di ricerca locale consiste nella creazione delle strutture dati necessarie all'algoritmo stesso per ottenere una maggior velocità di esecuzione. Queste strutture sono create dinamicamente attraverso opportune chiamate alla funzione *calloc* in modo tale da ottenere strutture di dimensioni sufficienti a contenere tutti i dati. Questa scelta è stata preferita all'utilizzo di strutture statiche sovradimensionate principalmente per due motivi:

- 1) Permette di allocare solo la quantità di memoria strettamente necessaria a contenere i dati
- 2) Consente di avere dati di ingresso di qualsiasi dimensione, non essendo vincolati ad una dimensione massima delle strutture dati

L'utilizzo di strutture dinamiche inoltre, data la particolare tipologia di software realizzato, non pone problemi relativi alla gestione della memoria stessa. Infatti, dato che il programma deve terminare dopo un periodo di tempo fissato a priori, non è necessario effettuare la deallocazione della memoria, che viene automaticamente liberata da parte del sistema operativo al termine dell'esecuzione del processo.

Una volta create, le strutture dati sono inizializzate con un valore convenzionale pari a “- 1” in modo tale da consentire l'individuazione immediata di eventuali valori non corretti, o non calcolati. Le strutture così inizializzate sono aggiornate inserendovi gli effettivi valori che vi devono essere contenuti e che sono utilizzati successivamente.

La fase che segue consiste nell'effettiva esecuzione dell'algoritmo di ricerca locale sui dati di ingresso, opportunamente modificati mediante tecnica greedy. L'ottimizzazione si basa sulla ricerca di tutti i possibili scambi tra qualsiasi coppia di colonne, nonché sullo spostamento di una qualsiasi colonna in una qualsiasi posizione, utilizzando per far ciò le strutture dati create. Quando viene trovata una "mossa" (scambio o spostamento) che migliora la soluzione corrente essa viene attuata ed il procedimento itera.

Questo ciclo di "ricerca mossa-attuazione mossa" prosegue finché l'algoritmo di ricerca locale è in grado di trovare "mosse" miglioranti. Se nessuna mossa migliorante è trovata, o se è trascorso il limite al tempo di esecuzione fissato, l'algoritmo termina. In questo caso la miglior soluzione trovata fino a quel momento viene salvata nella cartella destinata ai file soluzione ed il processo termina.

La soluzione trovata potrebbe essere un ottimo locale, un ottimo globale oppure nessuno dei due a seconda della soluzione iniziale e del tempo di esecuzione.

### 5.3 Strutture dati


Il funzionamento dell'algoritmo di ricerca locale richiede, come già citato in precedenza, l'utilizzo di strutture dati apposite. In particolare è possibile distinguere due categorie di strutture:

- Strutture di supporto al calcolo: utilizzate principalmente per contenere dati che consentano un calcolo più rapido dei costi. Appartiene a questa categoria "matrice optional finestra"
- Strutture dei costi: contenenti gli effettivi valori dei costi di scambio e spostamento sulla base dei quali è attuata l'ottimizzazione della soluzione. Fanno parte di questa categoria "matrice costi scambio" e "matrice costi spostamento"

[illegible]



 = prime finestre del giorno di produzione D.

 = finestre del giorno di produzione D.

 = ultime finestre del giorno di produzione D.

L'esempio di riferimento si riferisce ad un insieme di dati costituito da 8 autovetture da schedare il giorno di produzione D, 4 vetture schedate il giorno di produzione D - 1 e 5 ratio constraint del tipo seguente:

- Vincolo 1  $\rightarrow 1 / 3$
- Vincolo 2  $\rightarrow 1 / 4$
- Vincolo 3  $\rightarrow 1 / 2$
- Vincolo 4  $\rightarrow 2 / 3$
- Vincolo 5  $\rightarrow 1 / 5$

Una finestra K, riferita ad un generico ratio constraint N/P, può essere definita come una qualsiasi sequenza consecutiva di P veicoli compresi tra  $K - P + 1$  e K.

Le prime finestre schedate per il giorno di produzione D sono quindi rappresentate, per ciascun vincolo, da quelle finestre K tali per cui  $0 < K < (P - 1)$ . In tali finestre, le  $P - K$  vetture necessarie a completare una sequenza, sono rappresentate dalle ultime auto schedate per il precedente giorno di produzione.

Analogamente, le ultime finestre del giorno di produzione D sono costituite da tutte quelle finestre K tali per cui  $(N^{\circ} \text{ vetture} + 1) < K < (N^{\circ} \text{ vetture} + P - 1)$ . In tal caso le vetture mancanti per completare la sequenza si suppone non richiedano alcun optional.

Ciascuna cella della matrice descritta si riferisce ad una particolare finestra associata ad uno specifico ratio constraint ed indica il numero di richieste di optional contenute nella finestra stessa. Ad esempio la cella [2][5] conterrà il

numero di optional corrispondenti al ratio constraint 2 (definito da  $N_2 / P_2$ ) presenti nella sequenza di autovetture comprese tra  $(5 - P_2 + 1)$  e 5.

### 5.3.2 Matrice costi di scambio

Matrice costi di scambio contiene, per ciascuna coppia di colonne, il costo di scambio, ossia la variazione del costo totale della soluzione corrente che si avrebbe effettuando lo scambio. Il valore del costo può essere sia positivo che negativo; un valore positivo indica una “mossa” che aumenta il costo della soluzione e fornisce quindi una soluzione peggiore di quella corrente. Viceversa un costo negativo porta ad un miglioramento della soluzione che è tanto maggiore quanto più negativo è il costo.

La struttura deve contenere il costo di scambio per ciascuna possibile coppia di colonne e quindi deve avere dimensioni pari al numero di colonne disponibili, ossia al numero di autovetture da schedulare il giorno di produzione D.

Esempio:

**Matrice costi scambio**

0	0							
1		0						
2			0					
3				0				
4					0			
5						0		
6							0	
7								0
	0	1	2	3	4	5	6	7

La matrice rappresentata si riferisce a files di input caratterizzati dalla presenza di 8 autovetture da schedulare il giorno di produzione D. Ovviamente il costo di scambio di una colonna con se stessa è pari a 0 (tutti i valori sulla diagonale

principale) ed inoltre la matrice è simmetrica rispetto alla diagonale. Infatti scambiare la colonna 2 con la colonna 5 equivale a scambiare la colonna 5 con la colonna 2.

### 5.3.3 Matrice costi di spostamento

Il discorso effettuato per la matrice dei costi di scambio è valido anche per la matrice dei costi di spostamento. Le due strutture sono infatti identiche per dimensioni, ma con significato differente. Infatti in tal caso le celle contengono il costo di spostamento di una qualsiasi colonna in una qualsiasi posizione ed anche qui il costo può essere positivo (“mossa” peggiorante) o negativo (“mossa” migliorante).

Esempio:

**Matrice costi spostamento**

0	0							
1		0						
2			0					
3				0				
4					0			
5						0		
6							0	
7								0
	0	1	2	3	4	5	6	7

L'esempio riportato si riferisce agli stessi dati ipotizzati per la matrice dei costi di scambio (8 autovetture da schedare il giorno di produzione D). E' possibile notare come, anche per i costi di spostamento, tutti i valori sulla diagonale siano pari a 0 in quanto lo spostamento di una colonna nella medesima posizione in cui si trova non ha alcun effetto.

A differenza della matrice dei costi di scambio però la matrice dei costi di spostamento non è simmetrica rispetto alla diagonale principale.

## 5.4 Pesatura degli obiettivi

I costi di scambio e spostamento, rispettivamente tra colonne e di una colonna in una certa posizione, vengono calcolati tenendo in considerazione il peso di ciascun obiettivo. Secondo il metodo di valutazione stabilito per il concorso, infatti, una violazione all'obiettivo a massima priorità ha un peso molto superiore rispetto ad una violazione dell'obiettivo a priorità più bassa. In particolare, dati i tre obiettivi dell'ottimizzazione, i pesi di ogni singola defezione sono i seguenti:

- 1° obiettivo → peso violazione = 10000
- 2° obiettivo → peso violazione = 100
- 3° obiettivo → peso violazione = 1 (se l'obiettivo è presente, 0 altrimenti)

Questi valori, come detto in precedenza, sono memorizzati all'interno del vettore pesi al quale si accede in fase di calcolo dei costi.

### 5.4.1 Dettagli sui costi calcolati

I costi sono calcolati come aumento o diminuzione del numero totale di defezioni; è quindi possibile riassumerne il calcolo con la seguente formula:

$$C = W_1 * C_1 + W_2 * C_2 + W_3 * C_3$$

Dove:

$W_1, W_2, W_3$ : sono i pesi degli obiettivi di ottimizzazione, pari rispettivamente a 10000, 100, 1

$C_1, C_2, C_3$ : sono le variazioni alle defezioni di ciascun obiettivo di ottimizzazione

I valori  $C_1$ ,  $C_2$ ,  $C_3$  indicano quanto varia il numero totale di defezioni riferite ad ogni obiettivo di ottimizzazione e possono pertanto essere sia positivi che negativi. Se negativi significa che la “mossa” (scambio o spostamento) porta ad una riduzione del numero di violazioni dell’obiettivo in esame, se positivi si ha invece un incremento.  $C_1$ ,  $C_2$ ,  $C_3$  possono essere calcolati come:

$$C_x = D_{x, dopo} - D_{x, prima} \quad \text{con } X = 1, 2, 3 \quad \square$$

Dove:

$C_x$ : indica variazione del numero di defezioni dell’obiettivo X

$D_{x, dopo}$ : rappresenta il numero di defezioni dell’obiettivo X dopo la “mossa”

$D_{x, prima}$ : rappresenta il numero di defezioni dell’obiettivo X prima della “mossa”

#### Esempio:

*Ordine degli obiettivi:*

Minimizzazione del numero di violazioni dei ratio constraint ad alta priorità

Minimizzazione del numero di cambi di colore

Minimizzazione del numero di violazioni dei ratio constraint a bassa priorità

Numero di defezioni per obiettivo pari rispettivamente a ‘3’, ‘8’ e ‘4’.

Se uno scambio determina il calo del numero di cambi di colore da ‘8’ a ‘5’, lasciando inalterati i valori riferiti alle defezioni degli altri obiettivi, il costo di tale “mossa” sarà dato da:

$$C = 10000 * (3 - 3) + 100 * (5 - 8) + 1 * (4 - 4) = -300$$

In quanto si ha una riduzione pari a ‘3’ del numero di defezioni dell’obiettivo con peso 100 (in tal caso i cambi di colore).

## 5.5 Algoritmi di calcolo dei costi

La tecnica di ricerca locale che è stata implementata si basa su due strutture dati contenenti rispettivamente i costi di scambio ed i costi di spostamento. Tali costi sono definiti come descritto nel paragrafo precedente e la loro computazione avviene mediante algoritmi che ne garantiscano una valutazione rapida, data la presenza di un limite massimo al tempo di esecuzione. Le procedure definite per il calcolo dei costi sono quattro, utilizzate rispettivamente per:

- Costi di scambio relativi ai ratio constraint
- Costi di spostamento relativi ai ratio constraint
- Costi di scambio relativi ai cambi di colore
- Costi di spostamento relativi ai cambi di colore

La limitazione al tempo di esecuzione ha richiesto di adattare le funzioni sopracitate all'ordine degli obiettivi dello scenario considerato. L'ultimo obiettivo di ottimizzazione, infatti, ha un peso pari a 1 e quindi produce miglioramenti o peggioramenti poco rilevanti; ecco perchè è stato deciso di non considerarlo nel calcolo dei costi.

Questa scelta può portare a soluzioni che non coincidono con un ottimo per la funzione, ma, dato il tempo di esecuzione ridotto, si giunge a risultati migliori rispetto alla valutazione di tutti gli obiettivi. Ciò perchè il calcolo risulta più rapido ed è quindi possibile effettuare un numero superiore di “mosse” miglioranti per i due obiettivi più importanti che determinano anche i maggiori miglioramenti.

Questo è il motivo per cui, se la minimizzazione del numero di cambi di colore compare come ultimo obiettivo, le due funzioni per il calcolo dei costi relativi alle variazioni della verniciatura non vengono mai utilizzate.

### 5.5.1 Calcolo dei costi di scambio relativi ai ratio constraint

Il calcolo dei costi di scambio relativi ai ratio constraint avviene mediante un algoritmo che sfrutta le informazioni contenute nella struttura di supporto matrice optional finestra, descritta nei paragrafi precedenti.

Inizialmente la funzione richiesta era stata implementata in modo poco efficiente, infatti, (1) veniva calcolato il numero totale di defezioni relative al primo ratio constraint, (2) si simulava lo scambio delle vetture in questione, (3) si ricalcolava il numero totale delle defezioni e quindi (4) si ripristinava l'ordine iniziale dei veicoli. Il costo era determinato come differenza tra il numero di defezioni dopo lo scambio ed il numero di defezioni prima dello scambio. Per tenere in considerazione i pesi, la differenza determinata era moltiplicata per il peso del ratio considerato, ripetendo tutto il procedimento per ogni ratio constraint.

La soluzione descritta ha il vantaggio di essere abbastanza semplice da implementare, ma d'altra parte ha un onere computazionale elevato in quanto richiede, per ciascun ratio constraint, la valutazione di tutte le possibili finestre, il cui numero cresce linearmente con il numero di vetture da schedare per il giorno di produzione  $D$ . La procedura è stata perciò ridefinita in modo tale da riottenere i medesimi risultati, ma in un tempo molto inferiore.

L'algoritmo individuato per ottenere il risultato sopraindicato è indicato dal seguente pseudo-codice riferito al calcolo del costo di scambio, relativo ai ratio constraint, tra due colonne 'i' e 'j':

***Inizio algoritmo:***

$C = 0$

→ Costo iniziale

Per ogni riga  $r$

→ Per ogni ratio constraint di tipo N/P

Per ogni finestra  $K_i$  che comprenda  $i$  e non  $j$

→ Finestra  $K$  = sequenza di colonne di ampiezza  $P$  comprese tra  $K - P + 1$  e  $K$

Le finestre  $K_i$  sono definite come:

- Se  $j > i$   

$$i \leq K_i \leq \text{Min} \{ j - 1 ; i + P - 1 \}$$
- Se  $j < i$   

$$j \leq K_i \leq \text{Min} \{ i - 1 ; j + P - 1 \}$$

Definisco come  $a_{r,x}$  il valore presente nella riga  $r$  colonna  $x$  nella tabella che descrive le vetture; se tale valore è un 1 allora la vettura richiede l'optional  $r$ , se 0 non lo richiede. Definisco  $P_r$  il peso del ratio constraint  $r$ .

Se  $a_{r,i} = a_{r,j} \rightarrow C += 0$

→ Scambio due valori uguali quindi il costo che ne deriva è pari a 0

Se  $a_{r,i} < a_{r,j} \rightarrow C += N^\circ \text{ di finestre } K_i \text{ sature} * P_r$

→ Scambio uno 0 con un 1 quindi il costo che ne deriva è pari al numero di finestre che hanno già raggiunto o superato il limite previsto per il vincolo  $r$  in esame, infatti aggiungendo un 1 ad ognuna di esse si



introduce una nuova  
violazione del ratio constraint

Se  $a_{r,i} > a_{r,j} \rightarrow C - = N^\circ$  di finestra  $K_i$  violate  $* P_r \rightarrow$  Scambio un 1 con  
uno 0 perciò il costo  
diminuisce di un valore pari  
al numero di finestra violate,  
che cioè hanno già superato il  
limite  $N$  del vincolo  $N/P$ .  
Inserendo uno 0 in tali  
finestra si riduce infatti il  
numero di violazioni.

L'operazione deve quindi essere ripetuta considerando le finestre  $K_j$ , che  
considerano l'intorno della colonna  $j$ , definite come:

- Se  $j > i$   
$$\text{Max} \{ j ; i + P \} \leq K_j \leq (j + P - 1)$$
- Se  $j < i$   
$$\text{Max} \{ i ; j + P \} \leq K_j \leq (i + P - 1)$$

### ***Fine algoritmo.***

Il funzionamento dell'algoritmo descritto, nonchè la sua efficienza, si basa sulla  
struttura matrice optional finestra che permette di determinare in modo rapido  
quali sono le finestra sature e quali sono le finestre violate. Conoscendo questi  
valori il calcolo del costo può avvenire in modo molto efficiente rispetto  
all'algoritmo utilizzato inizialmente.

### 5.5.2 Calcolo dei costi di spostamento relativi ai ratio constraint

Il calcolo dei costi di spostamento relativi ai ratio constraint avviene in modo simile al caso dello scambio, sfruttando anche in questo caso la struttura di supporto matrice optional finestra. Inizialmente la procedura era stata implementata in modo molto semplice, sulla base di quanto fatto per i costi di scambio, seguendo i seguenti passi:

- 1) calcolo del numero totale di defezioni relative al primo ratio constraint
- 2) spostamento della vettura in questione nella posizione indicata
- 3) ricalcolo del numero totale delle defezioni
- 4) ripristino dell'ordine iniziale dei veicoli

Il costo derivava dalla differenza tra il numero totale delle violazioni dopo lo spostamento ed numero totale delle violazioni prima dello spostamento, il tutto moltiplicato per il peso del ratio constraint considerato. Il procedimento doveva essere ripetuto per ogni ratio constraint. Come è possibile notare il carico computazionale di questa soluzione era paragonabile al metodo inizialmente utilizzato per lo scambio, ed al pari di questo molto elevato.

La necessità di velocizzare il calcolo ha imposto la ridefinizione dell'algoritmo seguendo lo stesso schema utilizzato per i costi di scambio. Lo pseudo codice sottoriportato si riferisce al calcolo del costo di spostamento relativo ai ratio constraint di una colonna 'i' in 'pos':

#### ***Inizio algoritmo:***

Per ogni riga r

→ Per ogni ratio constraint di  
tipo N/P

Definisco come  $a_{r,x}$  il valore presente nella riga  $r$  colonna  $x$  nella tabella che descrive le vetture; se tale valore è un 1 allora la vettura richiede l'optional  $r$ , se 0 non lo richiede. Definisco  $P_r$  il peso del ratio constraint  $r$ .

Se  $i < pos$   $\rightarrow$  La colonna  $i$  viene spostata a destra

Per ogni finestra  $K$  definita come:

$i \leq K \leq \text{Min} \{ pos - 1 ; i + P - 1 \}$   $\rightarrow$  Valuto l'intorno della colonna  $i$

Se  $a_{r,k+1} > a_{r,i}$  e finestra satura  $\rightarrow C + = P_r \rightarrow$  Sostituisco uno 0 con un 1 in una finestra che ha già raggiunto il limite massimo di optional e quindi introduco una nuova violazione

Se  $a_{r,k+1} < a_{r,i}$  e finestra violata  $\rightarrow C - = P_r \rightarrow$  Sostituisco un 1 con uno 0 in una finestra che ha superato il limite  $N$  del vincolo  $N/P$ , quindi togliendo un 1 si elimina anche una violazione

$\text{Max} \{ pos ; i + P - 1 \} \leq K \leq (pos + P - 1)$   $\rightarrow$  Valuto l'intorno della posizione di destinazione della colonna  $i$

Se  $a_{r,i} > a_{r,K-P+1}$  e finestra satura  $\rightarrow C + = P_r \rightarrow$  In questa situazione il costo aumenta in quanto

sostituisco un 0 con uno 1  
all'interno di una finestra  
satura, ovvero con numero di  
optional pari a N

Se  $a_{r,i} < a_{r,K-P+1}$  e finestra violata  $\rightarrow C - = P_r \rightarrow$  Il costo diminuisce in  
quanto inserisco uno 0 in una  
finestra che ha già superato il  
limite massimo ammesso N

Se  $pos - i > P$

$\rightarrow$  Devo valutare anche le  
finestra comprese tra i e  
posizione

Per ogni finestra K definita da:

$$i + P \leq K \leq pos - 1$$

Se  $a_{r,K+1} > a_{r,K-P+1}$  e finestra satura  $\rightarrow C + = P_r \rightarrow$  Lo spostamento  
provoca l'inserimento di un 1  
in una finestra satura,  
determinando una violazione  
aggiuntiva

Se  $a_{r,K+1} < a_{r,K-P+1}$  e finestra violata  $\rightarrow C - = P_r \rightarrow$  Ciò causa una  
diminuzione al numero di  
defezioni dovuta alla  
riduzione al numero di  
optional della finestra K.

Se  $i > pos$

→ La colonna  $i$  è spostata a sinistra

Per ogni finestra  $K$  definita come:

$$pos \leq K \leq \text{Min} \{ i - 1 ; pos + P - 1 \}$$

Se  $a_{r,i} > a_{r,K}$  e finestra satura →  $C += P_r$  → Sto sostituendo uno 0 con un 1 e quindi il costo aumenta se la finestra è satura, ovvero che ha raggiunto il valore  $N$

Se  $a_{r,i} < a_{r,K}$  e finestra violata →  $C -= P_r$  → In tale situazione diminuisce il costo perché inserisco uno 0 in una finestra che è già violata in cui il numero di optional ha già superato il limite  $N$

$$\text{Max} \{ i ; pos + P - 1 \} \leq K \leq (i + P - 1)$$

Se  $a_{r,K-P} > a_{r,i}$  e finestra satura →  $C += P_r$  → Il costo aumenta in quanto inserisco un 1 in una finestra satura determinando una nuova violazione

Se  $a_{r,K-P} < a_{r,i}$  e finestra violata →  $C -= P_r$  → Sto sostituendo un 1 con uno 0 quindi se la finestra è violata riduco il costo,

perchè diminuisco le richieste di optional

Se  $i - pos > P$

→ Devo valutare anche le finestra comprese tra  $i$  e  $pos$

Per ogni finestra  $K$  definita da:

$$pos + P \leq K \leq i - l$$

Se  $a_{r,K} > a_{r,K-P}$  e finestra violata →  $C - = P_r$  → Lo spostamento provoca l'inserimento di uno 0 in una finestra violata, determinando una riduzione del numero di violazioni.

Se  $a_{r,K} < a_{r,K-P}$  e finestra satura →  $C + = P_r$  → Ciò causa una aumento al numero di defezioni dovuto all'incremento del numero di optional della finestra  $K$ .

***Fine algoritmo.***

Il metodo descritto è simile a quello utilizzato per il calcolo dei costi di scambio e sfrutta la struttura matrice optional finestra. Essa permette infatti di conoscere immediatamente se una particolare finestra è satura o violata, consentendo una determinazione rapida dei costi.

A differenza dei costi di scambio, tuttavia, il calcolo dei costi di spostamento risulta leggermente meno performante in quanto devono essere valutati tutti i ratio constraint. Nel caso degli scambi invece, se le due vetture in questione possiedono

la stessa richiesta di un particolare ratio constraint, questo può essere saltato, passando immediatamente al vincolo successivo perchè non si ha alcuna variazione al numero totale di defezioni.

Inoltre può essere necessaria la valutazione delle finestre comprese tra 'i' e 'pos' dato che l'azione di spostamento determina uno spostamento a sinistra o a destra delle colonne interposte tra 'i' e 'pos'.

### **5.5.3 Calcolo dei costi di scambio relativi ai cambi di colore**

I costi di scambio relativi ai cambi di colore possono essere calcolati in modo molto semplice effettuando una scansione della matrice contenente le informazioni riguardanti le autovetture, limitatamente ai colori della verniciatura. In questa prima fase di scansione vengono valutate le condizioni che determinano il lavaggio delle pistole spray individuando il numero totale di lavaggi necessari.

Nella seconda fase della procedura viene effettuato lo scambio tra le vetture e quindi si ripete la scansione dei colori della verniciatura, ripristinando successivamente la sequenza originale. Effettuando la differenza tra il numero di violazioni al vincolo dei cambi di colori dopo lo scambio e prima dello scambio si ottiene il costo. Questa soluzione garantisce buone prestazioni che possono tuttavia essere migliorate.

Un tentativo fatto era basato su un differente algoritmo che sfruttava una struttura dati appositamente creata contenente, per ogni posizione delle vetture da schedare, il numero di auto consecutive, compresa quella presente nella posizione valutata, con lo stesso colore, che si trovavano a sinistra ed a destra della posizione stessa. Questa tecnica è stata abbandonata principalmente perchè (1) non valutava tutti i possibili casi che si potevano incontrare, restituendo perciò

valori di costo errati, e (2) l'onere di gestione della struttura dati necessaria all'algoritmo ne superava i benefici.

L'algoritmo di calcolo basato sulla scansione è stato successivamente migliorato limitando la lettura ad una sola parte della matrice. La procedura che ne deriva è rappresentata dal seguente pseudo-codice riferito allo scambio tra due colonne 'i' e 'j':

***Inizio Algoritmo:***

$C = 0$  → Costo iniziale

Definisco  $b_x$  il colore di verniciatura della vettura x.

Se  $b_i = b_j \rightarrow C = 0$  → Le due vetture hanno lo stesso colore, quindi un loro scambio non porta alcuna variazione del costo totale

Altrimenti

Se  $i > j$  → La vettura i si trova in una posizione successiva rispetto all'auto j

Scambio i valori i e j → L'operazione di scambio è simmetrica

Colore =  $b_{i-1}$      $X = i - 1$  → Memorizzo il colore della vettura che precede la colonna i



Finché (Colore =  $b_x$ )

→ Individuo l'inizio della sequenza di auto con lo stesso colore che precedono i

$$X = X - 1$$

Colore =  $b_{j+1}$   $X = j + 1$

→ Memorizzo il colore della vettura che segue l'auto in posizione j

Finché (Colore =  $b_x$ )

→ Individuo la fine della sequenza di auto con lo stesso colore che seguono la vettura in posizione j

$$X = X + 1$$

Cambi\_Prima = numero di cambi di colore delle auto comprese tra i due limiti appena trovati.

Effettuo lo scambio dei colori delle colonne 'i' e 'j'

Cambi\_Dopo = numero di cambi di colore delle auto comprese tra i limiti sopra trovati

Ripristino colori originari delle vetture 'i' e 'j'

$C = \text{Cambi\_Dopo} - \text{Cambi\_Prima}$  → Il costo di scambio è calcolato seguendo la formula generale basata sulla differenza tra il numero di lavaggi delle pistole dopo e prima dello scambio

***Fine Algoritmo.***

L'algoritmo descritto offre buone prestazioni che tuttavia dipendono in larga misura dalle colonne 'i' e 'j'. Se il colore della verniciatura delle vetture che si trovano in posizione 'i' e 'j' è il medesimo, il costo è immediatamente determinato in quanto pari a zero. In caso contrario la resa della procedura dipende in modo inversamente proporzionale dalla differenza esistente tra le colonne 'i' e 'j'. Tanto più le vetture sono distanti tanto maggiore è il tempo di elaborazione necessario alla determinazione del costo di scambio.

#### 5.5.4 Calcolo dei costi di spostamento relativi ai cambi di colore

Il calcolo dei costi di spostamento relativi ai cambi di colore si realizza mediante il medesimo algoritmo a scansione, opportunamente adattato, utilizzato per gli scambi. Lo pseudo-codice della procedura adibita alla computazione del costo è la seguente, riferita allo spostamento di una vettura 'i' in 'pos':

**Inizio Algoritmo:**

$C = 0$  → Costo iniziale

Definisco  $b_x$  il colore di verniciatura della vettura x.

Altrimenti

Se  $i < pos$  → La vettura i deve essere  
spostata a destra

$Colore = b_{i-1} \quad X = i - 1$  → Memorizzo il colore della  
vettura che precede la  
colonna i

Finché (Colore =  $b_x$ )

→ Individuo l'inizio della sequenza di auto con lo stesso colore che precedono i

$$X = X - 1$$

Colore =  $b_{pos+1}$   $X = pos + 1$

→ Memorizzo il colore della vettura che segue l'auto in posizione pos

Finché (Colore =  $b_x$ )

→ Individuo la fine della sequenza di auto con lo stesso colore che seguono la vettura in posizione pos

$$X = X + 1$$

Altrimenti

→ La vettura in posizione i deve essere spostata a sinistra

Colore =  $b_{pos-1}$   $X = pos - 1$

→ Memorizzo il colore della vettura che precede la colonna pos

Finché (Colore =  $b_x$ )

→ Individuo l'inizio della sequenza di auto con lo stesso colore che precedono pos

$$X = X - 1$$

Colore =  $b_{i+1}$   $X = i + 1$

→ Memorizzo il colore della vettura che segue l'auto in posizione i

Finché (Colore =  $b_x$ )

→ Individuo la fine della sequenza di auto con lo stesso colore che seguono la vettura in posizione  $i$

$$X = X + 1$$

Cambi\_Prima = numero di cambi di colore delle auto comprese tra i due limiti appena trovati.

Effettuo lo spostamento della colonna 'i' in 'pos' limitatamente al colore della verniciatura

Cambi\_Dopo = numero di cambi di colore delle auto comprese tra i limiti sopra trovati

Ripristino la sequenza di colori originaria

$C = \text{Cambi\_Dopo} - \text{Cambi\_Prima}$  → Il costo di spostamento è calcolato seguendo la formula generale basata sulla differenza tra il numero di lavaggi delle pistole dopo e prima dello spostamento

### ***Fine Algoritmo.***

Le prestazioni offerte dall'algoritmo descritto sono mediamente leggermente peggiori rispetto al caso dello scambio, nonostante l'algoritmo sia molto simile. Ciò in quanto, nel caso dello spostamento, non vale la condizione per cui se le vetture analizzate possiedono lo stesso colore, allora il costo è pari a zero. In aggiunta la simulazione dello spostamento richiede un tempo superiore rispetto

allo scambio, soprattutto in caso di auto distanti, in quanto comporta uno spostamento a sinistra o a destra di tutte le vetture comprese tra 'i' e 'pos'.

Gli algoritmi per il calcolo dei costi di scambio e spostamento descritti hanno il pregio di non richiedere alcuna struttura dati aggiuntiva e di essere abbastanza semplici da implementare.

## 5.6 La fase di ottimizzazione

L'ottimizzazione della soluzione mediante ricerca locale è rappresentata nel diagramma 4:

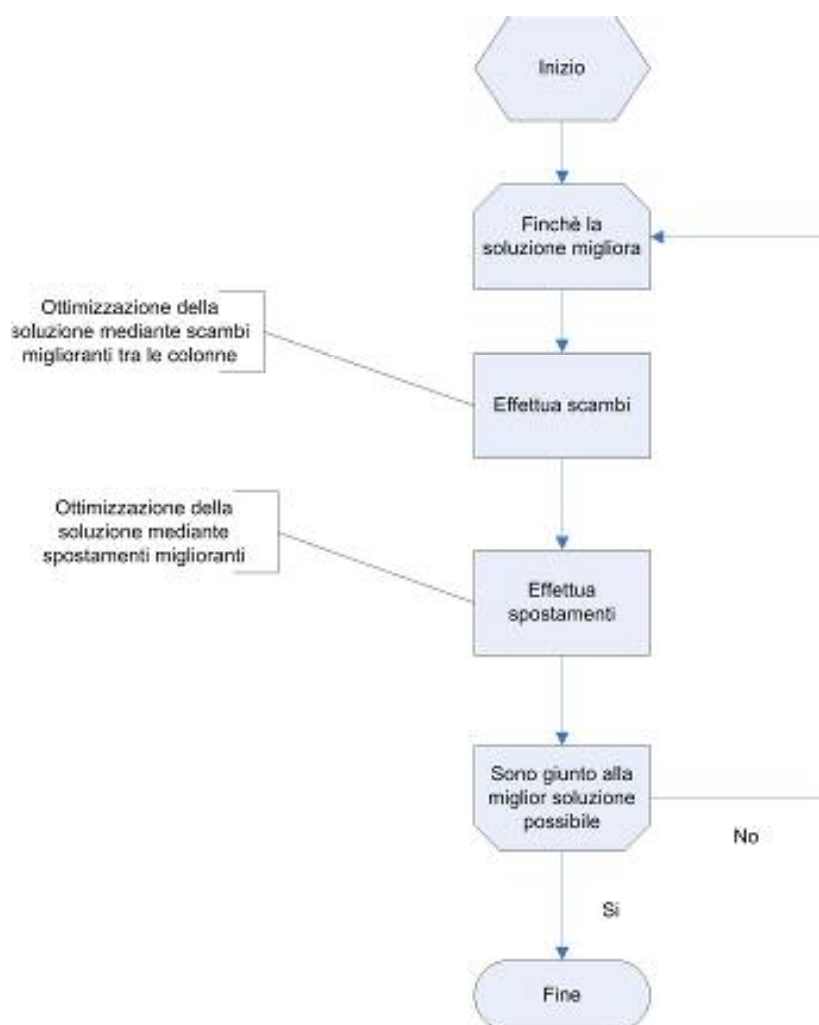


Diagramma 4 – Ottimizzazione

Il grafico dimostra lo schema di ottimizzazione utilizzato: finchè è possibile migliorare la soluzione corrente mediante scambi o spostamenti, il procedimento itera. E' possibile individuare un ciclo globale all'interno del quale sono presenti due parti distinte rappresentate da scambi e spostamenti.

### 5.6.1 Dettagli sull'ottimizzazione

La prima parte dell'ottimizzazione della soluzione avviene attraverso scambi tra colonne, mediante il procedimento seguente (diagramma 5):

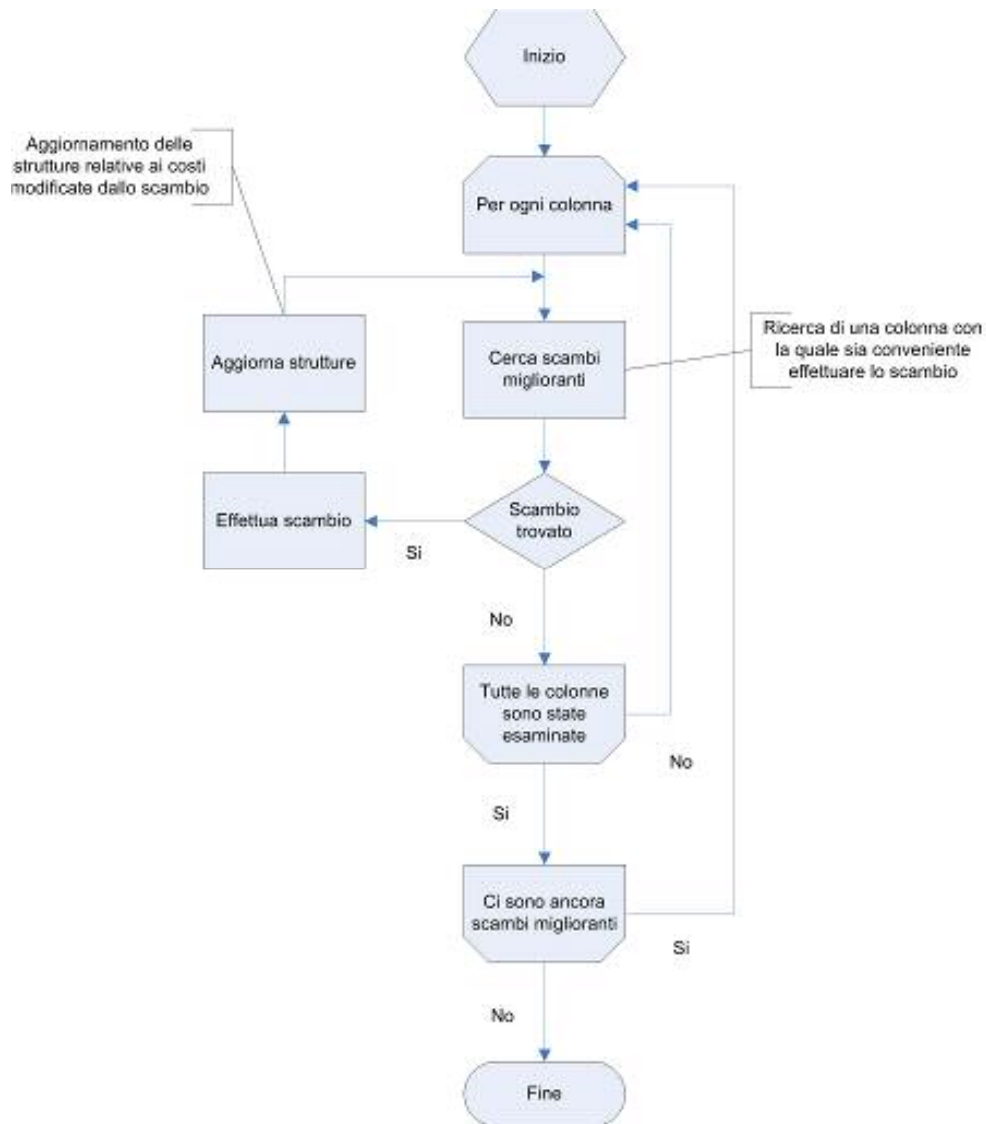
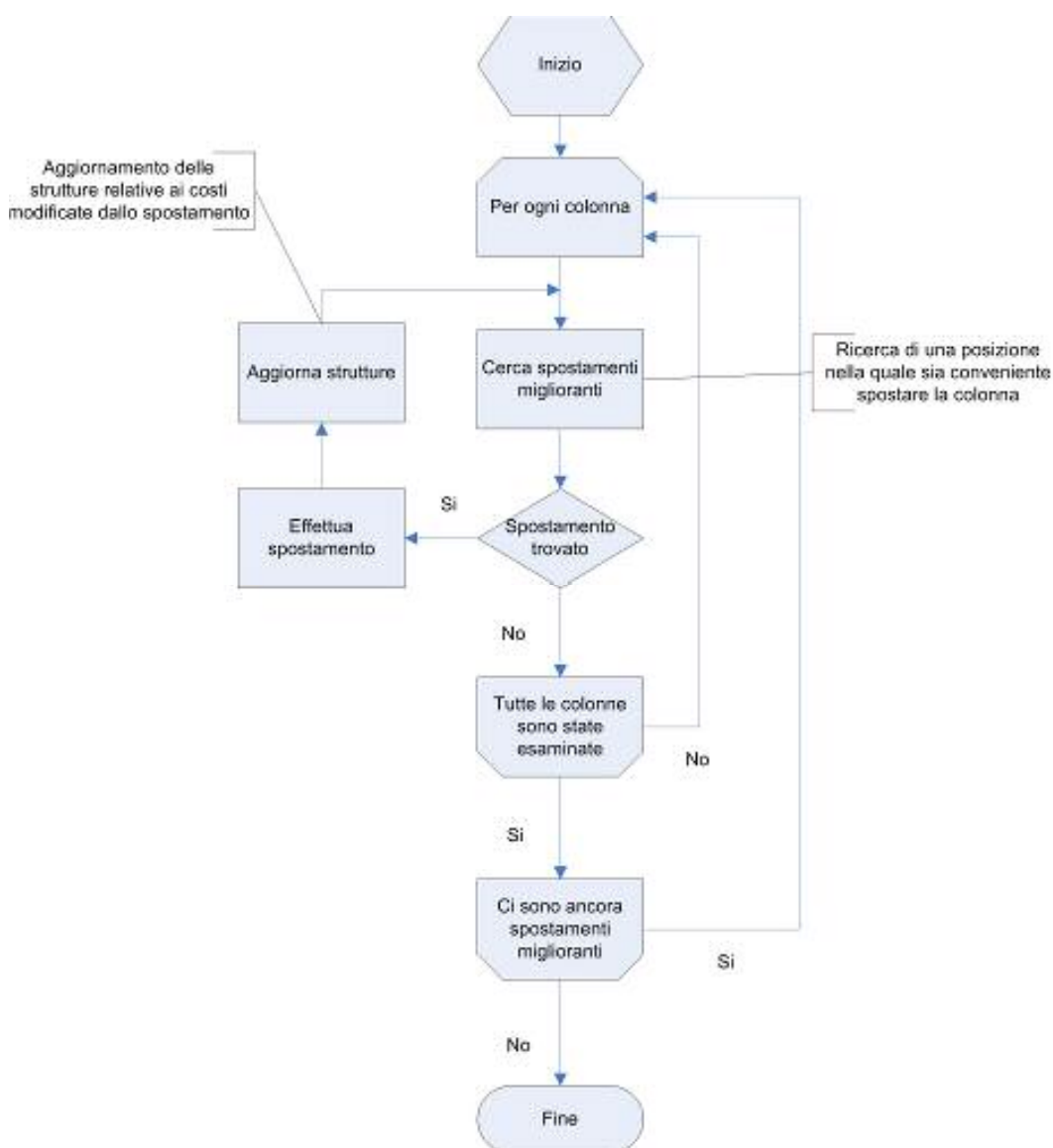


Diagramma 5 - Ottimizzazione con scambi

Il primo passo della ricerca locale si basa sugli scambi miglioranti: per ogni possibile posizione valuto tutti i costi con le altre vetture individuando quelli che apportano il maggior miglioramento della soluzione corrente. Successivamente effettuato lo scambio trovato, se presente, e aggiorno tutte le strutture dati utilizzate limitatamente alle posizioni influenzate dalla mossa attuata. Il procedimento itera per tutte le possibili colonne e quando ogni posizione è stata valutata il ciclo si ripete finchè sono presenti scambi miglioranti.

La fase successiva è rappresentata dal medesimo procedimento, applicato agli spostamenti, definito dal diagramma 6 seguente:



**Diagramma 6 - Ottimizzazione con spostamenti**

I diagrammi dimostrano come il meccanismo di ottimizzazione utilizzato sia il medesimo per scambi e spostamenti. Dato il limite al tempo di esecuzione, infatti, è necessario ridurre il costo totale della soluzione corrente scegliendo per ogni posizione la “mossa” che determina il maggior miglioramento, anche se le colonne coinvolte sono “lontane” tra loro.

### **5.6.2 L’aggiornamento delle strutture dati**

Le operazioni di scambio e spostamento impongono l’aggiornamento delle strutture dati coinvolte, rappresentate rispettivamente dalle matrici dei costi di scambio e di spostamento e, in entrambi i casi, della matrice optional finestra utilizzata per il calcolo dei costi.

Il limite al tempo di esecuzione ha richiesto di implementare le procedure utilizzate per l’aggiornamento in modo tale da imporre il minor onere possibile sulla CPU. Per effettuare ciò vengono individuate ed aggiornate tutte e sole le celle delle matrici sopracitate il cui valore è influenzato dalla mossa attuata.

La matrice contenente il numero di optional per finestra deve essere aggiornata sia che venga effettuato uno scambio che uno spostamento. A tal proposito sono utilizzati, opportunamente modificati, gli algoritmi per il calcolo dei costi in quanto essi sono in grado di individuare tutte e sole le finestre influenzate dalla mossa.

Le matrici riguardanti i costi di scambio e spostamento invece devono essere aggiornate in modo leggermente diverso a seconda della mossa realizzata. In particolare, quando l’ottimizzazione avviene mediante scambi, la matrice dei costi di spostamento non viene modificata, ricalcolandola interamente all’inizio della fase di ottimizzazione con spostamenti, in modo tale da velocizzare



l'esecuzione del processo. Situazione analoga nel caso degli spostamenti con riferimento alla matrice dei costi di scambio.

L'aggiornamento della matrice dei costi di scambio, in seguito allo scambio tra due colonne 'i' e 'j', avviene mediante l'algoritmo seguente:

***Inizio Algoritmo:***

max = massimo denominatore dei vincoli N/P

Se  $|i - j| > \max$

Aggiorno tutte le righe e le colonne X della matrice definite come

$$i - \max + 1 < X < i + \max - 1$$

e

$$j - \max + 1 < X < j + \max - 1$$

Altrimenti

Se  $i > j$

Aggiorno tutte le righe e le colonne X definite da

$$j - \max + 1 < X < i + \max - 1$$

Altrimenti

$$i - \max + 1 < X < j + \max - 1$$

***Fine Algoritmo.***

La matrice dei costi di scambio è simmetrica pertanto l'aggiornamento delle colonne avviene semplicemente attraverso assegnamenti ad indici invertiti. Ad esempio il costo in posizione [2][4] viene assegnato anche alla posizione [4][2].

Nel caso venga effettuato uno spostamento l'aggiornamento della matrice avviene utilizzando lo stesso algoritmo appena descritto, ma, dato che la matrice dei costi di spostamento non è simmetrica, devono essere calcolati tutti i valori sia per righe che per colonna. In aggiunta devono sempre essere aggiornate tutte le righe comprese tra la colonna spostata e la posizione di destinazione in quanto lo spostamento determina una traslazione di tali colonne, a sinistra o a destra a seconda della "mossa" realizzata.

Esempio di aggiornamento in seguito ad uno scambio:

numero di vetture = 10

max = 2

Scambio colonna 1 con colonna 10

**Matrice costi di scambio**

	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

 Celle aggiornate dall'algoritmo

Esempio di aggiornamento in seguito ad uno spostamento:

numero di vetture = 10

$\max = 2$

Spostamento colonna 1 con colonna 10

**Matrice costi di spostamento**

	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

 Celle aggiornate dall'algoritmo

In questo caso lo spostamento realizzato comporta il completo ricalcolo della matrice dei costi di spostamento in quanto spostando la prima colonna in ultima posizione, o viceversa, tutte le vetture sono coinvolte perchè scalate di una posizione a sinistra, o a destra.

## 5.7 Algoritmo di calcolo del costo di una soluzione

La ricerca locale opera sulla miglior soluzione generata dalla tecnica costruttiva la cui determinazione avviene mediante un algoritmo che si basa sul metodo di valutazione utilizzato per le soluzioni, ovvero il numero di violazioni di ogni obiettivo di ottimizzazione viene moltiplicato per il peso dell'obiettivo stesso pari a 10000, 100, 1 rispettivamente per il primo, il secondo ed il terzo obiettivo di ottimizzazione.

Esempio:

Ordine degli obiettivi:

- (1) Minimizzazione del numero di cambi di colore
- (2) Minimizzazione del numero di violazioni dei ratio constraint ad alta priorità
- (3) Minimizzazione del numero di violazioni dei ratio constraint a bassa priorità

Numero di violazioni:

Obiettivo (1) = 4

Obiettivo (2) = 15

Obiettivo (3) = 8

Costo soluzione =  $4 * 10000 + 15 * 100 + 8 * 1 = 41508$

Il calcolo del numero di violazioni dei ratio constraint avviene esaminando tutte le possibili finestre di autovetture e calcolando, per ognuna di esse, il numero di defezioni. Analogamente per il numero di cambi di colore, si scandisce il colore di verniciatura di tutte le auto valutando le condizioni che provocano il lavaggio delle pistole, ossia cambio di colore e superamento del limite di auto consecutive con lo stesso colore.

***Inizio Algoritmo***

$C = 0$

→ Costo iniziale totale della  
soluzione in esame

Per ogni riga  $r$

→ Per ogni ratio constraint

Definisco  $D_{r, K}$  il numero di violazioni dei ratio constraint della finestra  $K$ ;  $D_r$  il numero di violazioni del vincolo  $r$ ,  $D_C$  il numero di cambi di colore e  $P_r$  e  $P_C$  il peso rispettivamente del ratio constraint  $r$  e dei cambio di colore.

Per ogni possibile finestra  $K$

$$D_r += D_{r,K}$$

→ Sommo il numero di violazioni in ogni possibile finestra relativa al vincolo r.

Calcolo il valore  $D_C$  relativo al numero di lavaggi delle pistole necesario per la sequenza di vetture generata.

Per ogni ratio constraint r

$$C += D_r * P_r$$

→ Sommo al costo totale il prodotto del numero di violazioni di un ratio constraint per il suo peso.

$$C += D_C * P_C$$

→ Incremento il costo della soluzione del prodotto tra numero di cambi di colore e relativo peso

$C$  = Costo totale soluzione

***Fine Algoritmo.***



## 6. Output

### 6.1 File delle soluzioni

I files delle soluzioni contengono le sequenza di vetture generate ed ottimizzate dal programma, corrispondenti ai differenti scenari. Un file soluzione deve avere i seguenti campi: <<*Numero di sequenza; ID*>> con il numero di sequenza che inizia da 1. Il file soluzione dovrà avere lo stesso nome dello scenario.

Esempio del contenuto di un file soluzione

```
1;025031910825  
2;025031820293  
3;025031850497  
4;025031911085  
5;025031850299  
6;025031850082  
7;025031850783  
8;025031920415  
9;025032010270  
10;025031911083  
11;025031910060  
12;025031970126
```

Il file di output riportato contiene la sequenza schedulata dal software per il giorno di produzione D contenente 12 vetture, per ognuna delle quali viene inserito il numero di sequenza.

## **6.2 Programmi**

Il software può essere fornito come:

- un codice eseguibile che potrà essere eseguito direttamente sulla macchina fornita dalla Renault
- il codice C o C++ includendo un manuale per la compilazione, e l'associato *makefile* consentendo a Renault di creare un codice eseguibile

### **6.2.1 Limite al tempo di esecuzione ed ambiente di test**

Il tempo di esecuzione del programma è limitato a 600 secondi su un PC Pentium4/1.6Ghz/1Gb Ram. Questa limitazione rappresenta il massimo tempo di risposta accettabile da utenti industriali.

I programmi possono essere eseguiti sia in ambiente Windows2000 che RedHat Linux 7.3.

Software commerciali con codici di licenza sono proibiti.

### **6.2.2 Gestione del limite di tempo**

La limitazione del tempo di esecuzione ha imposto l'inserimento all'interno del programma di funzioni appositamente dedicate a tale scopo. Queste funzioni, se da un lato assicurano il rispetto del limite, dall'altro determinano un inevitabile



calo delle performance in quanto devono essere richiamate frequentemente ed in diverse sezioni del software stesso. Questo perché, per alcune parti del codice, non è possibile prevedere a priori il tempo di esecuzione; ad esempio non è possibile sapere quanti scambi / spostamenti saranno effettuati sui dati relativi ad un qualsiasi scenario, nè quanti cicli di scambio / spostamento saranno necessari.

Quando viene raggiunto il limite dei 10 minuti di esecuzione il software interrompe l'ottimizzazione e genera in uscita la miglior soluzione trovata fino a quel momento, anche se questa non rappresenta la migliore possibile. Se, viceversa, la miglior soluzione viene raggiunta prima della scadenza del limite di tempo, viene generato l'output e l'esecuzione termina.

Per compensare la “perdita” dovuta al controllo del tempo di esecuzione, e comunque per velocizzare il codice, si è pensato di utilizzare le funzionalità di ottimizzazione fornite dai compilatori. A tal proposito sono state effettuate differenti prove, compilando lo stesso codice sorgente, con diversi compilatori, ognuno dei quali è stato settato per fornire il codice eseguibile più veloce possibile. Sono stati testati i seguenti compilatori reperiti in Internet:

- GCC
- LCC
- TurboC
- DMC
- Borland C Compiler

Gli eseguibili ottenuti sono stati testati con il medesimo scenario ed il compilatore che ha fornito il minor tempo di esecuzione è stato GCC. Esso ha consentito di incrementare notevolmente la velocità di esecuzione nella misura di circa il 30% rispetto alla versione generata senza alcuna ottimizzazione.

### 6.2.3 Organizzazione gerarchica dei file

Per ogni partecipante al concorso identificato come *Candidate-NN*, la gerarchia dei file è:

- Directory principale: *Candidate-NN*
- Sottodirectory:
  - *Candidate-NN/Team-description/*
  - *Candidate-NN/Method-description/*
  - *Candidate-NN/Result-synthesis/*
  - *Candidate-NN/Instances/*
  - *Candidate-NN/Solutions/*
  - *Candidate-NN/Results/*
  - *Candidate-NN/Program/*

Sia che sia fornito sia che sia generato, l'eseguibile deve essere utilizzabile digitando nella directory *Candidate-NN/Program/*:

***nome-dell'eseguibile nome\_dell'istanza -t tempo\_di\_CPU***

con:

*nome\_dell'istanza*: nome dello scenario

*tempo\_di\_CPU*: tempo di esecuzione allocato in secondi, limitato a 600

Il programma dovrebbe trovare i dati di input dello scenario *nome\_dell'istanza* nella directory *Candidate-NN/Instances/nome\_dell'istanza* e scrivere la soluzione ottenuta dopo *tempo\_di\_CPU* nella directory *Candidate-NN/Solutions/*. Il file soluzione deve avere nome *nome\_dell'istanza*.

Nel caso in cui non venga specificato il tempo di esecuzione, si assume come tempo di default 600 secondi. Questi vincoli sono stati considerati nell'implementazione del programma.

## 6.3 Procedure di valutazione e classificazione

Ci sono 2 categorie di valutazione:

- Senior: tutti i candidati appartengono a questa categoria
- Junior: limitato a progetti di studenti (un progetto è un progetto di studenti se la maggior parte dei membri della squadra sono studenti, possibilmente seguiti dai propri professori)

Nota:

Da notare che un progetto di studenti può vincere il premio della categoria senior se il progetto è il migliore tra tutti i progetti inviati. Al contrario un progetto senior non può vincere il premio della categoria junior.

### 6.3.1 Il programma di valutazione

Il programma di valutazione Renault, per ogni scenario, richiama il programma del candidato, lo lascia eseguire per 600 secondi, cerca il file soluzione ottenuto nella directory *Candidate-NN/Solutions/* , lo controlla e lo valuta e scrive la valutazione della soluzione nella directory *Candidate-NN/Results/*.

Per programmi non deterministici, sono richieste 10 esecuzioni per scenario per ottenere la soluzione media.

### 6.3.2 Metodo di valutazione di uno scenario

Il metodo di valutazione per le qualificazioni è indicato di seguito. Nel metodo di valutazione per le finali potranno essere apportati piccoli cambiamenti.

Per ogni scenario viene calcolato un punteggio, pesando i differenti obiettivi di ottimizzazione dello scenario.

Scenario con massima priorità ai ratio constraint

Posizione degli obiettivi di ottimizzazione:

1. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
2. Minimizzare il numero di cambi di colore
3. Minimizzare il numero di violazioni dei ratio constraint a bassa priorità

Punteggio dello scenario

$$\begin{aligned} &= (10000 * \text{numero di violazioni dei ratio constraint ad alta priorità}) \\ &+ (100 * \text{numero di cambi di colore}) \\ &+ (\text{numero di violazioni dei ratio constraint a bassa priorità}) \end{aligned}$$

Oppure

1. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
2. Minimizzare il numero di violazioni dei ratio constraint a bassa priorità
3. Minimizzare il numero di cambi di colore

Punteggio dello scenario

$$\begin{aligned} &= (10000 * \text{numero di violazioni dei ratio constraint ad alta priorità}) \\ &+ (100 * \text{numero di violazioni dei ratio constraint a bassa priorità}) \\ &+ (\text{numero di cambi di colore}) \end{aligned}$$

Oppure

1. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
2. Minimizzare il numero di cambi di colore

Punteggio dello scenario

$$\begin{aligned} &= (10000 * \text{numero di violazioni dei ratio constraint ad alta priorità}) \\ &+ (100 * \text{numero di cambi di colore}) \end{aligned}$$

### Scenario con massima priorità ai cambi di colore

Posizione degli obiettivi di ottimizzazione:

1. Minimizzare il numero di cambi di colore
2. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità
3. Minimizzare il numero di violazioni dei ratio constraint a bassa priorità

Punteggio dello scenario

= (10000 \* numero di cambi di colore)

+ (100 \* numero di violazioni dei ratio constraint ad alta priorità)

+ (numero di violazioni dei ratio constraint a bassa priorità)

Oppure

1. Minimizzare il numero di cambi di colore
2. Minimizzare il numero di violazioni dei ratio constraint ad alta priorità

Punteggio dello scenario

= (10000 \* numero di cambi di colore)

+ (100 \* numero di violazioni dei ratio constraint ad alta priorità)

### **6.3.3 Valutazione globale di un test set**

Per ogni test set ci sono 3 classi di scenari:

- scenari con massima priorità ai ratio constraint e ratio constraint ad alta priorità facili da soddisfare
- scenari con massima priorità ai ratio constraint e ratio constraint ad alta priorità difficili da soddisfare
- scenari con massima priorità ai cambi di colore

Per ogni classe di scenari è calcolata una media dei punteggi ottenuti su tutti gli scenari appartenenti alla stessa classe (il punteggio di ogni scenario è calcolato utilizzando il metodo discusso in precedenza).

Attraverso le 3 classi, viene calcolato un punteggio globale per valutare il candidato:

$$\sum_{classe} [ (\text{punteggio del candidato}(classe) - \text{punteggio peggiore}(classe)) / (\text{punteggio migliore}(classe) - \text{punteggio peggiore}(classe)) ]$$

Il punteggio peggiore ed il punteggio migliore sono il peggiore di tutti i punteggi ottenuti dal candidato, e il migliore di tutti i punteggi ottenuti dal candidato, per una data classe di scenari.

Il punteggio globale è l'unico criterio per classificare i candidati. Questo metodo di valutazione dovrebbe eliminare qualsiasi gap numerico tra i punteggi medi delle 3 classi di scenari.

Basandosi sui risultati del test set A, un massimo di 10 candidati saranno selezionati per le finali.

I finalisti saranno classificati sulla base dei risultati sul test set X. Il test set B è fornito solamente per mettere a punto i programmi dei candidati.

## 7. Conclusioni

### 7.1 Risultati del test set A

L'algoritmo è stato testato su tutti gli scenari appartenenti al Test set A, in quanto questi sono il mezzo con cui la giuria del concorso determina i concorrenti che avranno accesso alla fase finale. Come richiesto esplicitamente dall'organizzazione, tutti i risultati sono stati raccolti mediante un foglio elettronico ed inviati, unitamente alla documentazione generale, al responsabile della competizione.

Riportiamo in seguito la tabella riassuntiva sopraccitata:

Nome dello Scenario	Violazioni r. c. ad alta priorità	Violazioni r. c. a bassa priorità	Cambi di colore	Punteggio Scenario
022_3_4_EP_RAF_ENP	0	1	105	10501
022_3_4_RAF_EP_ENP	48	4	11	114804
024_38_3_EP_ENP_RAF	54	84	543	548943
024_38_3_EP_RAF_ENP	38	258	366	416859
024_38_5_EP_ENP_RAF	69	82	518	698718
025_38_1_EP_ENP_RAF	0	383	863	39163
024_38_5_EP_RAF_ENP	63	124	422	672323
025_38_1_EP_RAF_ENP	0	1890	282	30090
039_38_4_EP_RAF_ch1	47	0	196	489600
039_38_4_RAF_EP_ch1	364	0	69	726400
048_39_1_EP_ENP_RAF	26	75	399	267899
048_39_1_EP_RAF_ENP	25	765	189	269665
064_38_2_EP_RAF_ENP_ch1	0	756	180	18756
064_38_2_EP_RAF_ENP_ch2	0	58	49	4958
064_38_2_RAF_EP_ENP_ch1	440	799	64	684799
064_38_2_RAF_EP_ENP_ch2	411	97	27	311197

Per avere un'idea più specifica della bontà del software riportiamo un confronto tra i punteggi ottenuti utilizzando una soluzione corrispondente alla sequenza di vetture fornita in ingresso e quella finale generata dal programma.

Nome scenario	Punteggio Iniziale	Punteggio finale	Differenza
022_3_4_EP_RAF_ENP	27002	10501	16501
022_3_4_RAF_EP_ENP	114805	114804	1
024_38_3_EP_ENP_RAF	828164	548943	279221
024_38_3_EP_RAF_ENP	766505	416859	349646
024_38_5_EP_ENP_RAF	1073792	698718	375074
025_38_1_EP_ENP_RAF	207790	39163	168627
024_38_5_EP_RAF_ENP	1026899	672323	354576
025_38_1_EP_RAF_ENP	31075	30090	985
039_38_4_EP_RAF_ch1	1172900	489600	683300
039_38_4_RAF_EP_ch1	729200	726400	2800
048_39_1_EP_ENP_RAF	430134	267899	162235
048_39_1_EP_RAF_ENP	369061	269665	99396
064_38_2_EP_RAF_ENP_ch1	40287	18756	21531
064_38_2_EP_RAF_ENP_ch2	284650	4958	279692
064_38_2_RAF_EP_ENP_ch1	687043	684799	2244
064_38_2_RAF_EP_ENP_ch2	319761	311197	8564

Differenza media = **175274,6**

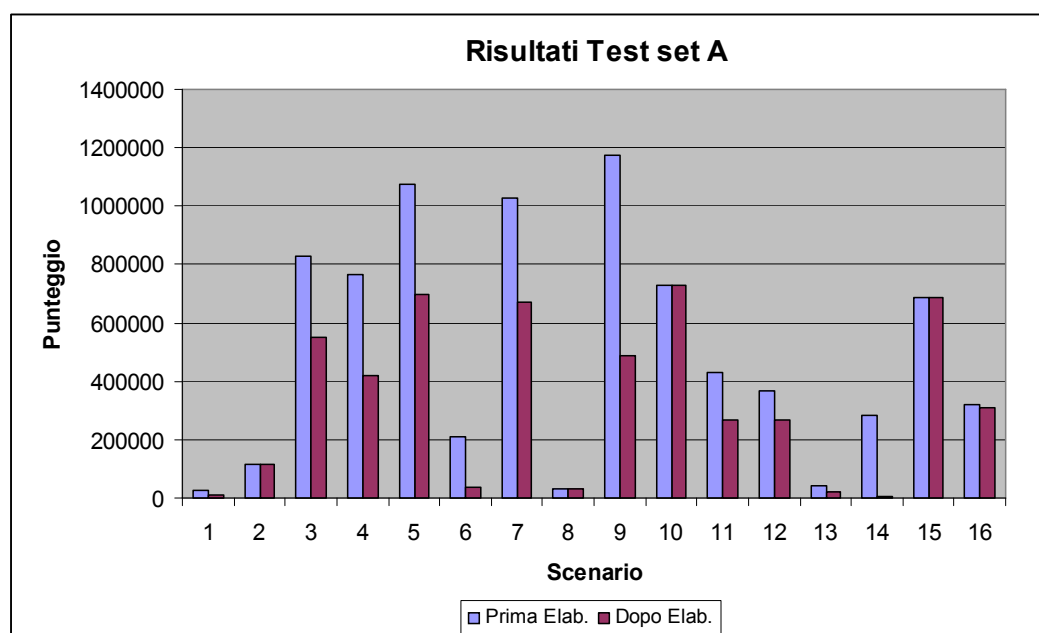
Il buon comportamento medio del programma è evidenziato dalla media dei miglioramenti ottenuti sui differenti scenari. Analizzando specificatamente i singoli dati d'ingresso, tuttavia, si possono notare notevoli differenza nei risultati. Ciò è dovuto alle differenti caratteristiche delle vetture da schedulare nel giorno di produzione D di ogni scenario ed al diverso ordine degli obiettivi.

Un riscontro dell'influenza dei dati sui risultati finali, lo possiamo ottenere analizzando gli scenari nei quali l'obiettivo a massima priorità è rappresentato dalla minimizzazione del numero di cambi di colore. In questi casi, spesso, la soluzione iniziale minimizza, di per sé, il numero di lavaggi delle pistole, quindi il miglioramento ottenibile attraverso l'elaborazione dei successivi obiettivi, non comporta evidenti variazioni di punteggio.



Quanto detto in precedenza è chiaramente visibile nel grafico sotto riportato, in cui le colonne azzurre rappresentano il punteggio della soluzione fornita in ingresso, le viola quello della sequenza generata dall'algoritmo.

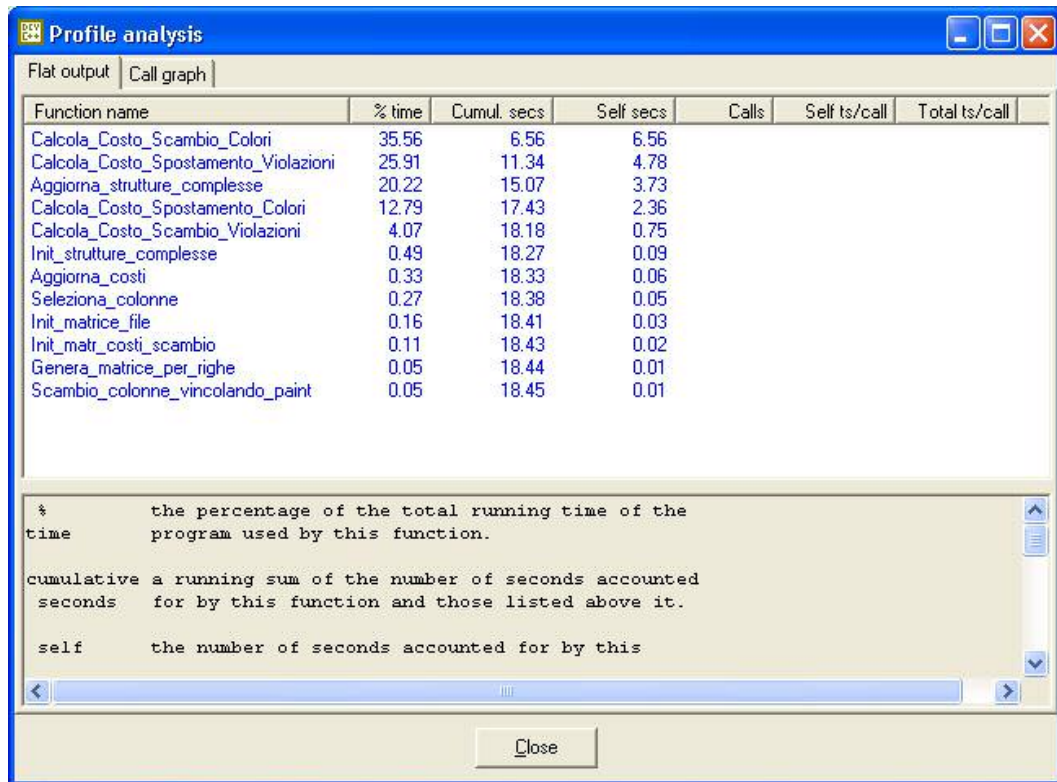
Vi è una corrispondenza diretta tra numero e nome degli scenari, basata sulla posizione di questi ultimi all'interno delle tabelle sopra riportate, ad esempio, lo scenario denominato "024\_38\_3\_EP\_ENP\_RAF" corrisponde, nel grafico, all'indice di colonna "3".



## 7.2 Risultati attività di profiling

L'esecuzione del software ha un limite alla durata massima (10 minuti), per questo si è resa utile l'attività di profiling che permette di conoscere quali sono le procedure più utilizzate, mostrando per ognuna di esse il tempo di CPU utilizzato, nonché la percentuale del tempo di esecuzione occupato rispetto al totale.

I risultati ottenuti, hanno mostrato come l'algoritmo si adatta all'ordine degli obiettivi di ottimizzazione, infatti, la graduatoria delle procedure maggiormente richiamate, varia di scenario in scenario. Ad esempio, nello scenario "064\_38\_2\_RAF\_EP\_ENP\_CHI", che come obiettivo primario richiede la minimizzazione dei cambi di colore, l'analisi ha fornito il seguente risultato:



The screenshot shows a window titled "Profile analysis" with two tabs: "Flat output" (selected) and "Call graph". Below the tabs is a table with the following data:

Function name	% time	Cumul. secs	Self secs	Calls	Self ts/call	Total ts/call
Calcola_Costo_Scambio_Colori	35.56	6.56	6.56			
Calcola_Costo_Spostamento_Violazioni	25.91	11.34	4.78			
Aggiorna_strutture_complesse	20.22	15.07	3.73			
Calcola_Costo_Spostamento_Colori	12.79	17.43	2.36			
Calcola_Costo_Scambio_Violazioni	4.07	18.18	0.75			
Init_strutture_complesse	0.49	18.27	0.09			
Aggiorna_costi	0.33	18.33	0.06			
Seleziona_colonne	0.27	18.38	0.05			
Init_matrice_file	0.16	18.41	0.03			
Init_matr_costi_scambio	0.11	18.43	0.02			
Genera_matrice_per_righe	0.05	18.44	0.01			
Scambio_colonne_vincolando_paint	0.05	18.45	0.01			

Below the table is a legend:

- % time: the percentage of the total running time of the program used by this function.
- cumulative: a running sum of the number of seconds accounted for by this function and those listed above it.
- self: the number of seconds accounted for by this

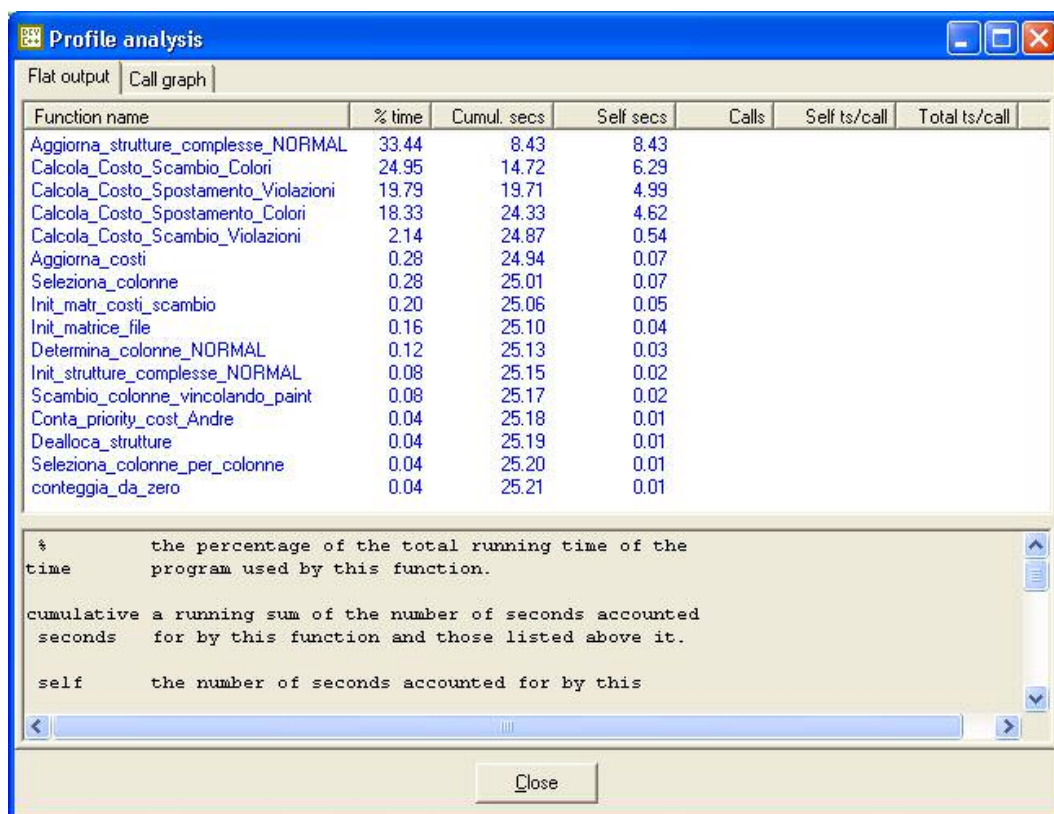
At the bottom of the window is a "Close" button.

**Profiling 1: Scenario 064\_38\_2\_RAF\_EP\_ENP\_CHI**

Dallo screen shot si evince che il maggior peso ricade sull'algoritmo di ricerca locale; infatti, le tecniche costruttive basate sui colori, hanno una durata limitata a causa di un numero di vetture esaminato in ogni posizione, ridotto alle sole auto del colore in esame.

Passando ad uno scenario caratterizzato da un differente ordine di obiettivi, in cui il più importante è quello sui ratio constraint ad alta priorità, la distribuzione dei tempi di CPU è differente. In questo caso la ricerca locale occupa ancora la maggior parte del tempo di elaborazione ma, al contrario dell'esempio precedente, la procedura più onerosa appartiene alla tecnica costruttiva. Il riscontro è

evidenziato nei dati appartenenti allo scenario “022\_3\_4\_EP\_RAF\_ENP” riportato in seguito:



The screenshot shows a window titled "Profile analysis" with two tabs: "Flat output" and "Call graph". The "Flat output" tab is active, displaying a table with the following data:

Function name	% time	Cumul. secs	Self secs	Calls	Self ts/call	Total ts/call
Aggiorna_strutture_complesse_NORMAL	33.44	8.43	8.43			
Calcola_Costo_Scambio_Colori	24.95	14.72	6.29			
Calcola_Costo_Spostamento_Violazioni	19.79	19.71	4.99			
Calcola_Costo_Spostamento_Colori	18.33	24.33	4.62			
Calcola_Costo_Scambio_Violazioni	2.14	24.87	0.54			
Aggiorna_costi	0.28	24.94	0.07			
Seleziona_colonne	0.28	25.01	0.07			
Init_matr_costi_scambio	0.20	25.06	0.05			
Init_matrice_file	0.16	25.10	0.04			
Determina_colonne_NORMAL	0.12	25.13	0.03			
Init_strutture_complesse_NORMAL	0.08	25.15	0.02			
Scambio_colonne_vincolando_paint	0.08	25.17	0.02			
Conta_priority_cost_Andre	0.04	25.18	0.01			
Dealloca_strutture	0.04	25.19	0.01			
Seleziona_colonne_per_colonne	0.04	25.20	0.01			
conteggia_da_zero	0.04	25.21	0.01			

Below the table, there is a legend explaining the columns:

- % time: the percentage of the total running time of the program used by this function.
- cumulative seconds: a running sum of the number of seconds accounted for by this function and those listed above it.
- self: the number of seconds accounted for by this

The window has a "Close" button at the bottom right.

**Profiling 2: Scenario 022\_3\_4\_EP\_RAF\_ENP**

Come ultimo esempio prendiamo in considerazione lo scenario “025\_38\_1\_EP\_ENP\_RAF” in cui le variazioni di colore risultano l’ultimo obiettivo in ordine di importanza. La risposta dell’analisi di profiling evidenzia un’elevata disparità tra i tempi di esecuzione di due procedure, rispetto a tutte le altre.

La prima appartiene alla ricerca locale e precisamente al calcolo dei costi; essendo l’obiettivo della minimizzazione dei cambi di colore l’ultimo da considerare, la funzione è costretta a considerare tutti i ratio constraints, anche quelli a bassa priorità. La seconda invece riguarda l’azzeramento della struttura “Albero” utilizzata dalla tecnica costruttiva. Una fase di inizializzazione così lenta è dovuta al fatto che sono presenti nello scenario un elevato numero di vincoli,

tutti da considerare, e questo aumenta in maniera esponenziale le dimensioni di questa struttura dati.

Profile analysis

Flat output | Call graph

Function name	% time	Cumul. secs	Self secs	Calls	Self ts/call	Total ts/call
Calcola_Costo_Scambio_Violazioni	62.74	185.06	185.06			
Azzerati_Albero	25.63	260.66	75.60			
Aggiorna_costi	5.13	275.78	15.12			
Aggiorna_strutture_complesse_NORMAL	3.09	284.89	9.11			
Riempi_Albero	1.12	288.19	3.30			
Crea_Albero	0.99	291.11	2.92			
Genera_matrice_per_colonne_NORMAL	0.29	291.98	0.87			
Seleziona_colonne	0.26	292.75	0.77			
Init_matrice_help	0.16	293.22	0.47			
pow	0.08	293.45	0.23			
Genera_matrice_per_righe_NORMAL	0.07	293.65	0.20			
Seleziona_colonne_per_colonne	0.07	293.85	0.20			
conteggia_da_zero	0.05	294.01	0.16			
Init_matr_costi_scambio	0.04	294.13	0.12			
Crea_file_soluzione	0.03	294.22	0.09			
Crea_matrice	0.03	294.31	0.09			
conteggia	0.03	294.39	0.08			
Converti_in_decimale	0.02	294.46	0.07			
Scambia_colonna	0.02	294.52	0.06			
main	0.02	294.58	0.06			
Determina_colonne_NORMAL	0.02	294.63	0.05			

% the percentage of the total running time of the program used by this function.  
 cumulative a running sum of the number of seconds accounted for by this function and those listed above it.  
 self the number of seconds accounted for by this

Close

### Profiling 3: Scenario 025\_38\_1\_EP\_ENP\_RAF

In generale l'analisi di profiling riporta i risultati attesi, ossia la tecnica greedy è in grado di fornire una soluzione in tempi ragionevoli, a discapito però della ottimalità di quest'ultima. Quanto prodotto dall'algoritmo costruttivo viene in seguito ottimizzato mediante la ricerca locale che risulta la parte più onerosa in termini di tempo, anche perché a priori non è possibile conoscere la durata delle elaborazioni che verranno effettuate.

## **7.3 Possibili miglioramenti**

L'algoritmo è in grado di fornire sequenze di vetture che rappresentano buone soluzioni per il problema trattato. Nonostante ciò sono stati individuati i possibili settori ai quali apportare delle modifiche per migliorare la qualità dei risultati proposti dall'algoritmo greedy ed incrementare la velocità di esecuzione della ricerca locale.

### **7.3.1 Migliorare la qualità dei risultati**

Le tecniche greedy forniscono delle soluzioni in tempo breve ma, purtroppo, queste non sempre risultano di buona qualità a causa dei meccanismi di decisione e di pesatura delle automobili da inserire in ogni posizione della sequenza. Migliorando i criteri di selezione e specificando ulteriormente le condizioni di scelta, mediante un miglior utilizzo delle informazioni relative a frequenza o densità dei ratio constraint, si potrebbe ottenere una soluzione che si avvicini maggiormente all'ottimo globale. Questo faciliterebbe i compiti della ricerca locale oltre a ridurre i tempi di esecuzione.

### **7.3.2 Incrementare la velocità di esecuzione**

L'attività di profiling ha evidenziato come le procedure che richiedono maggior tempo di CPU, siano quelle relative al calcolo dei costi, sia di scambio che di spostamento, e riferite sia ai ratio constraint che ai cambi di colore. Queste procedure sono già state implementate attraverso metodi che garantiscono una computazione rapida, in particolare per quel che riguarda i ratio constraint, tuttavia si potrebbero trovare soluzioni algoritmiche in grado di eseguire la computazione in un tempo inferiore.

### **7.3.3 Miglioramenti generali**

Entrambe le tecniche implementate hanno la caratteristica di non considerare come fondamentale l'ottimizzazione dell'ultimo obiettivo; quindi un miglioramento generale si avrebbe valutando in modo approfondito tutti gli obiettivi. Questo probabilmente andrà a discapito del tempo di elaborazione, ma, avendo a disposizione un calcolatore più potente e senza limiti al tempo di esecuzione, le soluzioni a cui si potrebbe giungere sarebbero senz'altro migliori di quelle attuali.

## **Appendice A.**

### **Indice dei diagrammi**

Diagramma 1 - Funzionamento generale . . . . .	32
Diagramma 2 - Inizializzazione e caricamento dati . . . . .	34
Diagramma 3 - Ricerca locale . . . . .	45
Diagramma 4 - Fase di ottimizzazione . . . . .	69
Diagramma 5 - Ottimizzazione con scambi . . . . .	70
Diagramma 6 - Ottimizzazione con spostamenti . . . . .	71