L'obiettivo principale della tesi consiste nello studio e l'implementazione di alcuni algoritmi per partizionare in modo ottimale un cammino pesato sui vertici in modo da ridurre il numero di colori in un'immagine digitale senza, per questo, ridurne il contrasto e la riconoscibilità. Per fare questo abbiamo utilizzato algoritmi di ottimizzazione per il partizionamento di garfi e, in particolare, di alberi e cammini.

Abbiamo impostato la tesi partendo dal caso più generale dei reticoli o grafi a griglia dimostrando come in questi casi, e in generale per quasi tutti i grafi bi-dimensionali, non sia possibile trovare la soluzione al problema in tempo polinomiale. Tali problemi appartengono, infatti, all'insieme dei problemi NP-completi, una classe che contiene problemi di cui ancora non si conosce un algoritmo risolutore di complessità polinomiale e caratterizzati da interessanti ed importanti proprietà che saranno approfondite nel primo capitolo.

Considerando il fatto che lo scopo del lavoro e le nostre applicazioni non sono direttamente connesse alla partizione di grafi generici o di quelli bi-dimensionali, abbiamo deciso di parlare solo brevemente del problema del partizionamento di tali grafi, cercando di sottolineare in modo particolare sia la loro NP-completezza, sia la possibilità di ottenere alcune risposte sulla loro complessità sfruttando alcuni particolari algoritmi che restituiscono soluzioni *quasi* ottime e che vanno sotto il nome di *algoritmi approssimanti*. Questi sono algoritmi il cui costo C della soluzione prodotta si discosta per un certo fattore dal costo C^* della soluzione ottima. Quanto più questo fattore è piccolo, tanto migliore risulta l'algoritmo appossimante.

Abbiamo introdotto, allora, il problema del partizionamento di grafi a griglia prendendo spunto da un articolo di Conti, Malucelli, Nicoloso e Simeone [4] per dimostrare la possibilità di ridurre il problema ad altri simili, ma meno complicati, sfruttando delle euristiche. Alcune di queste sono molto semplici

sia da definire che da implementare poiché che sfruttano delle imposizioni paricolari sui tagli da effettuare (fissando ad esempio i tagli orizzontali o verticali). Altre sono un po' più complicate anche perché sfruttano, all'interno dell'algoritmo stesso, anche le euristiche di cui abbiamo parlato sopra rendendo lo studio e l'implementazione più articolati.

Abbiamo considerato, poi, il caso del p-partizionamento di grafi di tipo Max-Min studiato da Becker, Lari, Lucertini e Simeone [2], un particolare tipo di partizionamento che abbiamo studiato anche nel capitolo sugli alberi, e che divide il grafo a griglia in p componenti connesse con lo scopo di massimizzare il più possibile la componente di peso minore. Per la prima parte di questa sezione il risultato più interessante, oltre a quelli per dimostrare la correttezza degli algoritmi, è la dimostrazione del fatto che anche il problema della bi-partizione di un grafo G a griglia di dimensione $M \times 3$, tale che il peso di entrambe le componenti sia maggiore di un intero K prefissato, è NP-completo

Per concludere il capitolo dedicato ai grafi bi-dimensionali, abbiamo considerato due algoritmi approssimanti denominati SHIFT e SHIFTA che approssimano la soluzione ottima del problema e che hanno una complessità di $O(n^3p)$ e $O(n^3p^2)$ rispettivamente.

Una delle strategie adottate nell'implementazione degli algorimi per la risuzione dei colori nelle immagini digitali è stata quella di considerare anche gli algoritmi sviluppati per partizionare alberi. Come vedremo, per la soluzione del problema è sufficiente considerare dei cammini, ma, sfruttando la considerazione banale che un cammino è anche un albero, abbiamo applicato un algoritmo per il partizionamento di alberi al caso dei cammini. Abbiamo studiato, allora, in modo approfondito il problema del partizionamento di alberi ed in particolare due algoritmi denominati MIN-MAX e MAX-MIN che risolvono il problema in tempo polinomiale. Il primo dei due algoritmi,

dovuto a Becker, Shach e Perl [3], sfrutta due tipi di spostamenti dei tagli (che all'inizio vengono posti tutti sull'arco incidente la radice dell'albero): il primo verso il basso (*down shift*) e il secondo lateralmente (*side shift*). L'algoritmo termina non appena la componente che contiene la radice diventa la componente più pesante tra tutte quelle dell'albero e trova una soluzione con complessità computazionale dell'ordine $O(k^3h(T)+kn)$ (in cui con $h(T)$ abbiamo indicato l'altezza dell'albero, con k il numero dei tagli e con n i nodi che lo compongono). Il secondo, di Perl e Shach [11], sfrutta solo gli spostamenti dei tagli verso il basso e si blocca quando il peso della componente generata dallo spostamento di un taglio diventa minore del peso della componente più leggera del passo precedente; trova una soluzione con complessità computazionale dell'ordine $O(k^2h(T)+kn)$. Abbiamo deciso, anche perché ha una complessità inferiore, che fosse quest'ultimo l'algoritmo da implementare nel caso dei cammini con il risultato (visibile anche dalle figure riportate nelle ultime pagine della tesi) che l'algoritmo produce un output di buona qualità; tuttavia, non essendo specializzato nel partizionamento di cammini, la sua complessità computazionale è decisamente superiore a quella dell'algoritmo PATH SHIFTING che vedremo in seguito.

L'ultima parte della tesi è dedicata allo studio approfondito degli algoritmi per il partizionamento dei cammini. Il problema del partizionamento in questo caso trova applicazioni in svariati campi tra i quali uno dei più interessanti riguarda, come abbiamo già accennato, l'elaborazione di immagini digitali. Le immagini importate su un computer tramite scanner o fotocamere digitali sono rappresentate con una matrice di punti ognuno dei quali può essere codificato con un intero in cui il valore minore, lo 0, rappresenta un pixel nero, mentre il più grande indica un pixel completamente bianco; i valori intermedi indicano quantità crescenti di luminosità in una "scala di grigi".[1] Per vari problemi che possono sorgere e che vanno dal-

[1]Naturalmente, per semplicità, stiamo riportando il caso di immagini monocromatiche, che può essere esteso facilmente al caso delle immagini a colori.

la bassa risoluzione delle macchine adottate, all'occupazione di memoria di questo tipo di rappresentazioni, è spesso utile ridurre la scala di livelli di grigio fino ad un numero di fumature di gran lunga inferiore all'immagine originale. Questo problema può essere ben rappresentato tramite i cammini in questo modo: ogni vertice rappresenta un colore e lo scopo è quello di suddividere il cammino in un numero di componenti fissato pari alnumero di sfumature che si vuole ottnere. Il peso che associamo ad ogni vertice i rappresenta il numero di pixel aventi l'i-esimo livello di luminosità nell'immagine digitale. È possibile partizionare il cammino, pesato sui vertici nel modo che abbiamo appena detto, assumendo come funzione obiettivo da minimizzare una tra quelle associate alle varie norme: L_1, L_2 e L_∞. Indicando con π la partizione su cui viene calcolata la funzione obiettivo, con n il numero dei vertici, con p il numero di componenti in cui si vuole suddividere il cammino, possiamo definire le funzioni obiettivo come segue:

- L_1, la funzione obiettivo è : $f(\pi) = \sum_{k=1}^{n} |W(C_k) - \mu|$
- L_2, la funzione obiettivo è : $f(\pi) = \sum_{k=1}^{n} (W(C_k) - \mu)^2$
- L_∞, la funzione obiettivo è : $f(\pi) = \max_k |W(C_k) - \mu|$

dove con μ si è indicato il peso medio di $\pi = \{C_1, \ldots, C_k\}$

$$\mu = \frac{W(C_1) + W(C_2) + \ldots W(C_p)}{p} = \frac{W(V)}{p}$$

A questo proposito abbiamo ritenuto importante studiare alcuni algoritmi per il bilanciamento delle immagini digitali che tengono conto del minimo (L) e il massimo (U) numero di pixel all'interno di ogni componente e che cercano di risolvere il problema del partizionamento di un cammino pesato sui nodi trovando il più piccolo valore di $U - L$, con la condizione che il peso di ogni componente debba rimanere nel range $[L, U]$. Lucertini, Perl

e Simeone [10] hanno dimostrato che il miglior contrasto ottico si ottiene tanto più quanto questa differenza è piccola. Il problema sopra descritto va sotto il nome di MUP (*Most Uniform Partitioning*).

Oltre a questo caso particolare abbiamo considerato anche degli esempi meno restrittivi sulle condizioni, sui quali vengono implementati degli algoritmi che sfruttano una tecnica detta di *preprocessing*. Per la parte più importante del lavoro, quella su cui abbiamo costruito le applicazioni presentate nell'ultima parte, abbiamo reputato interessante implementare l'algoritmo di PROG DINAMICA [1] con la norma L_1 e l'algoritmo PATH SHIFTING [9] con la norma L_∞. Se tra tutte le partizioni che possono essere generate su un cammino dato in input, non consideriamo quelle improprie (ovvero tutte quelle partizioni che ammettono qualche componente nulla), le partizioni possibili le possiamo rappresentare con un grafo simile a quello di figura 1, in cui la partizione (a) rappresenta una partizione in cui le prime $p-1$ componenti contengono un unico vertice, mentre l'ultima componente C_p contiene i rimanenti $n-p-1$ vertici e la (b) rappresenta la partizione "opposta" (in cui la prima componente C_1 contiene $n-p-1$ vertici e gli altri vertici sono equamente distribuiti tra le rimanenti componenti).

Il primo algoritmo sfrutta una tecnica di programmazione detto *programmazione dinamica* che risolve il problema combinando la soluzione dei sottoproblemi comuni che vengono calcoli una sola volta memorizzandone la soluzione per poterla riutilizzare in seguito.

Il secondo (PATH SHIFTING), invece, fornisce un nuovo algoritmo per l'equipartizione di un cammino che migliora in modo consistente la complessità della soluzione del problema, essendo caratterizzato da un'efficienza superiore a quella di altri algoritmi analoghi presenti in letteratura. Il metodo che sfrutta si basa su una tecnica detta *simulated annealing*, tecnica che differisce da molte altre per il criterio di accettazione della nuova partizione

Figura 1: Grafo di tutte le possibili p-partizioni di un cammino di n vertici

generata: vengono accettate infatti, anche partizioni che peggiorano il valore della funzione obiettivo, nella speranza che questo peggioramento permetta di scavalcare eventuali minimi locali.

Se osserviamo ancora la figura 1 possiamo sintetizzare i due algoritmi dicendo che, mentre l'algoritmo di programmazione dinamica calcola tutte le possibili partizioni (proprie) scegliendo quelle con valore ottimo (la programmazione dinamica, infatti, non genera *una* soluzione con valore ottimo, ma diverse soluzioni ottime), l'altro algoritmo ne "salta" alcune scegliendo di volta in volta (sotto alcune ipotesi) la soluzione che risulta essere la migliore.

Riportiamo, infine, in figura , una elaborazione ottenuta con l'implementazione degli algoritmi presentati nella tesi. La prima figura (a) è l'immagine originale, in (b) abbiamo riportato l'immagine ottenuta con l'algoritmo MAX-MIN, (c) riporta l'immagine elaborata con l'algoritmo di programmazione dinamica mentre in (d) abbiamo riportato quella ottenuta tramite l'algoritmo PATH SHIFTING. Abbiamo scelto l'elaborazione della raffigu-

razione de “Il bacio” di Francesco Hayez (1859) perché ci sembra l’immagine che più mette in risalto la differenza tra gli algoritmi. In realtà, come è facile vedere, le immagini in (b) e (d) sono molto simili (anche se PATH SHIFTING sembra restituire un’immagine leggermente più definita soprattutto sul muro in secondo piano) mentre invece l’immagine prodotta dall’algoritmo di programmazione dinamica è molto più “solarizzata” e meno definita rispetto alle altre due.

Figura B.3: Elaborazione de “Il bacio”

Bibliografia

[1] Enzo L. Aparo, B. Simeone, *Un algoritmo di equipartizione e il suo impiego in un problema di contrasto ottico*, Ricerca operativa n.6, 1973.

[2] R. I. Becker, I. Lari, M. Lucertini, B. Simeone, *Max-Min partitioning of grid graphs into connected components*, Networks 32, 1998, pp. 115-125.

[3] R. I. Becker, S. R. Scach, Y. Perl *A shifting algorithm for Min-Max tree partitioning*, Journal of the Association for Computing Machinery, Vol. 29, No. 1, gennaio 1982, pp. 58-67.

[4] F. Conti, F. Malucelli, S. Nicoloso, B. Simeone, *On a 2-dimensional equipartition problem*, preprint.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Introduzione agli algoritmi*, Jackson Libri, 1998, Vol. 1, (cap. 5), Vol. 2 (cap. 16-17), Vol. 3 (cap. 36-37).

[6] C. De Simone, M. Lucertini, S. Pallottino, B. Simeone, *Fair dissections of Spider, Worms and Caterpillars*, Networks, Vol. 20, 1990, pp. 323-344.

[7] M. R. Garey, D. S. Johnson, *Computers and Intractability. A guide to the theory of NP-completeness*, BT Laboratories, 1979.

[8] S. Kundu, J. Misra, *A linear tree partitioningalgorithm*, SIAM J. Computer, 1977, pp.151-154.

[9] M. Liverani, A. Morgana, B. Simeone, G. Storchi, *Path equipartition in the Chebyshev norm*, European Journal of Operational Research, 123, 2000, pp. 428-436.

[10] M. Lucertini, Y. Perl, B. Simeone, *Most uniform path partitioning and its use in image processing*, Discrete Applied Mathematic 42, 1993, pp. 227-256.

[11] Y. Perl, S. R. Schach, *Max-Min tree partitioning*, Journal of the Association for Computing Machinery, Vol. 28, No. 1, gennaio 1981, pp. 5-15.

[12] M. Schroeder, *Balanced tree partitioning*, preprint, 2000.