Università di Catania

A solution for MWVC using Iterated Local Search

Laboratorio Intelligenza Artificiale (LM-18)
Università degli Studi di Catania - A.A 2021/2022

Danilo Leocata
Docente: Mario Pavone

March 22, 2022

1 Introduzione

Si propone una soluzione per il Weight Vertex Cover problem utilizzando l'Iterated Local Search: l'obbiettivo proposto è trovare la migliore soluzione, data un istanza, con il minimo numero di iterazioni. I Il codice è stato interamente in Java senza utilizzare librerie esterne eccetto `matplotlib4j` per la generazione dei grafici di convergenza e `CSVWriter` per la generazione del `.csv` dei benchmarks.

Il codice è interamente disponibile al seguente repository GitHub https://github.com/khalld/mwvc-using-ils-java, nel quale sono stati caricati i risultati di benchmark e parte dei grafici di convergenza realizzati.

Prima di procedere con l'implementazione e la scelta dell'algoritmo, sono state consultate e prese in considerazioni varie pubblicazioni riguardanti la soluzione del problema (indicate a fine relazione). È stata trovata sin da subito interessante implementare una soluzione utilizzando l'Iterated Local Search rispetto all'algoritmo genetico con il quale è computazionalmente più oneroso trovare una buona soluzione iniziale. Se si riuscisse a trovare una buona soluzione iniziale ed implementando un operatore di perturbazione sarebbe più semplice ottenere soluzioni vicine da valutare.

2 Generazione della soluzione iniziale

Banalmente è stato trovato opportuno implementare un algoritmo greedy per costruire la soluzione iniziale. Sono stati effettuati dei test in cui il punto di partenza è la soluzione *peggiore* (che contiene tutti i nodi dell'istanza) le cui performance sono sempre state peggiori rispetto all'algoritmo greedy

Algorithm 1 `GetInitialSolution`

```
Require: graph, allVertex
  totalWeight = 0
  selectedVertex = initialize empty list of vertex
  selectedEdges = initialize empty list of edges
  for edge in allEdges do
     get source and destination of current edge
     if is already explored, skip
     if sourceVertexWeight < destVertexWeight then
        set sourceVertex explored
        totalWeight+= sourceVertexWeight
        add sourceVertex to selectedVertex
        add current edge to selectedEdges
     else
        set destVertex explored
        totalWeight+= destVertexWeight
        add destVertex to selectedVertex
        add current edge to selectedEdges
     end if
  end for
      return Initial Solution
```

3 Validità della soluzione

La soluzione è da considerarsi completa e valida se tutti gli archi del grafo sono esplorati: di conseguenza sarà necessario controllare la validità della soluzione dopo la rimozione di un nodo. Una generica soluzione viene dunque validata scorrendo la lista di adiacenza di tutti i nodi selezionati: se dunque non saranno presenti tutti gli archi del grafo allora sarà considerata non valida.

4 Operatore di perturbazione

L'idea generale della perturbazione è quella di modificare i parametri ad ogni iterazione applicando una perturbazione sui nodi selezionati dalla soluzione. È stato inoltre già dimostrato da una delle referenze citate che una perturbazione troppo forte potrebbe portare allo stallo dell'ottimo locale: di è conseguenza in una prima fase si è deciso di implementare la seguente:

Un buon algoritmo deve evitare di far cadere sempre nello stesso minimo locale, in generale:

Algorithm 2 `WeakPerturbation`

```
Require: Solution, availableVertex
  if there aren't nodes not selected then
     remove random vertex from already seleceted
  else
     remove random vertex from already seleceted
     add random vertex from list of not seleceted
  end if
```

Tra le migliorie che la perturbazione potrebbe apportare alla soluzione vi è:

- l'eliminazione automaticamente di cicli se questa contiene dei nodi ridondanti;
- potrebbe rendere la soluzione non completa: di conseguenza applicando nuovamente la LocalSearch è possibile trovare un nodo candidato migliore rispetto a quello rimosso.

Tuttavia è stato notato, durante fasi di sviluppo, che questo operatore tendeva a perdere efficacia in quanto tendeva a bloccare la soluzione sull'ottimo locale. Di conseguenza, è trovato opportuno introdurre un parametro ϵ ed una variante della `Weakperturbation`:

Algorithm 3 `SecondChoicePerturbation`

```
Require: Solution, availableVertex
  remove random vertex from already seleceted
```

Il parametro ϵ rappresenta banalmente la percentuale di *risparmio* sul costo tra la soluzione corrente e quella peggiore. La `SecondChoicePerturbation` verrà utilizzata solo quando ϵ sarà maggiore di 25: in questo modo la perturbazione non contribuirà al *lock* sull'ottimo locale: basandoci su questa percentuale vi è la certezza che vi saranno alcuni vertici non selezionati. Quest'ultima, funziona bene principalmente su istanze *medie* e *grandi*.

5 Criterio di accettazione

Per evitare che il criterio accetti solo soluzioni migliori rispetto alla precedente è stata introdotta una componente randomica per accettare anche soluzioni *non migliori* in modo da esplorarne il vicinato e cercare di trovare una soluzione ottima.

Algorithm 4 AcceptanceCriteria

Require: `prevSolution, newSolution`

```
if cost of prev solution > cost of new solution then
   return new solution
end if
extract random number between 1 and 2

if extractedNumber is equal to 1 then
   return new solution
end if
return prev solution
```

6 Local Search e criterio di selezione

La local search implementata permette di completare la soluzione perturbata, cercando tra i nodi che hanno gli archi non selezionati. Un vertice può essere *preferito* rispetto ad un altro se:

- ha un peso minore rispetto agli altri candidati e dunque può potenzialmente far abbassare il costo della soluzione;
- ha un numero di archi maggiore rispetto agli altri candidati e dunque può potenzialmente contribuire di più alla soluzione;

Una versione randomizzata che utilizza entrambi ha portato i risultati migliori

7 Benchmarks e conclusione

È stato implementato uno script che permette di salvare i benchmark le soluzioni ottenute dalle istanze su file `.csv`, oltre alla creazione dei grafici di convergenza. Con una buona dose di fortuna l'algoritmo riesce a trovare una soluzione buona sin dalle prime iterazioni, l'introduzione del parametro *epsilon* ha contribuito ad ottenere una scoperta del vicinato piuttosto esaustiva e dei benchmark piuttosto buoni.

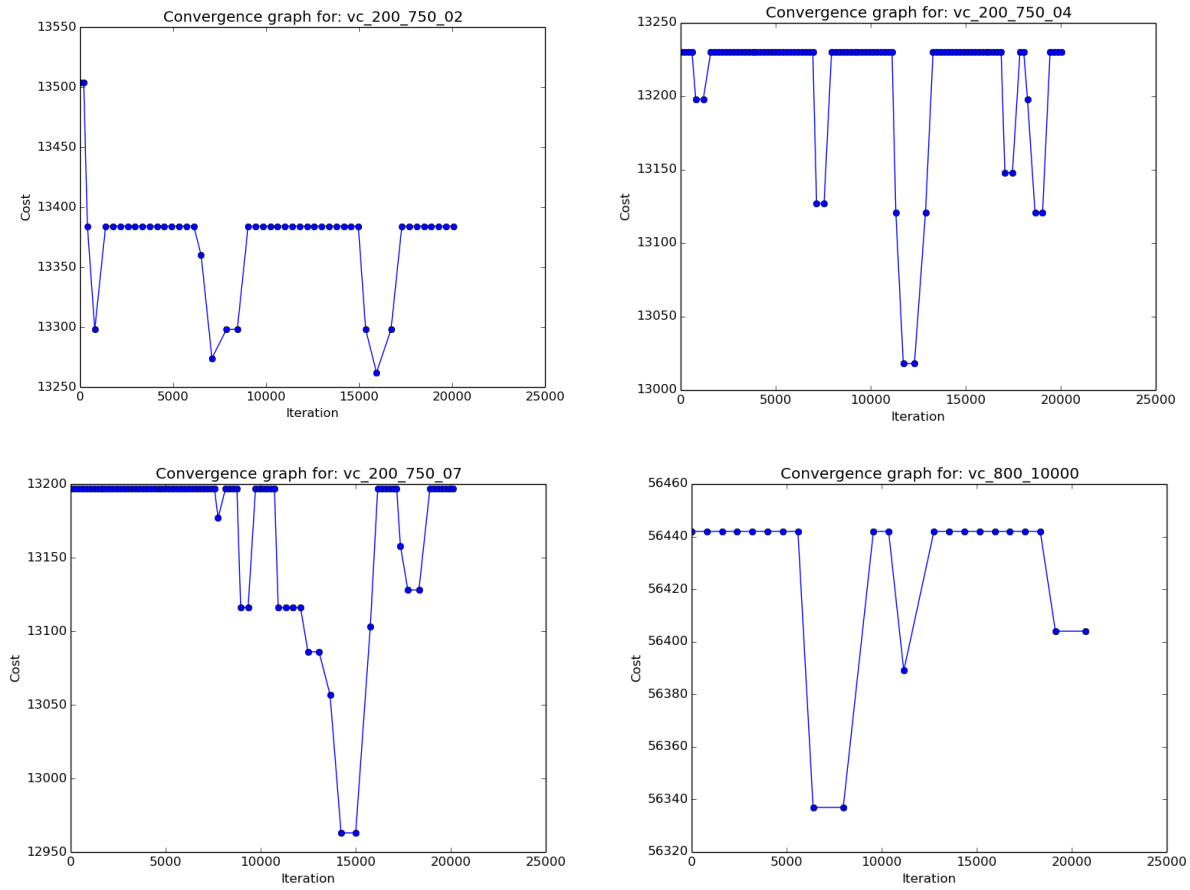

Figure 1: Sample grafici di convergenza

instance	best solution	best solution iter	elapsed ms
vc_20_120_01	1253	77	99
vc_20_120_02	1348	191	94
vc_20_120_03	1319	58	92
vc_20_120_04	1414	1179	99
vc_20_120_05	1293	761	92
vc_20_120_06	1271	77	102
vc_20_120_07	1265	1806	98
vc_20_120_08	1520	1654	98
vc_20_120_09	1587	229	97
vc_20_120_10	1452	381	95
vc_20_60_01	1105	405	33
vc_20_60_02	1585	476	37
vc_20_60_03	1313	58	34
vc_20_60_04	1177	115	37
vc_20_60_05	1352	324	37
vc_20_60_06	1426	343	33
vc_20_60_07	1573	210	36
vc_20_60_08	1526	39	36
vc_20_60_09	1595	324	36
vc_20_60_10	1416	552	35
vc_25_150_01	1785	265	108
vc_25_150_02	1472	241	103
vc_25_150_03	1658	20000	105
vc_25_150_04	1923	97	103
vc_25_150_05	1792	1849	114
vc_25_150_06	1651	505	99
vc_25_150_07	1977	841	104
vc_25_150_08	1693	361	107
vc_25_150_09	1739	841	111
vc_25_150_10	1547	553	102
vc_200_750_01	14160	1991	280
vc_200_750_02	13384	399	201
vc_200_750_03	14509	6767	297
vc_200_750_04	13230	797	306
vc_200_750_05	14495	399	211
vc_200_750_06	13586	1792	325
vc_200_750_07	13197	7762	286
vc_200_750_08	13676	9155	298
vc_200_750_09	13710	200	208
vc_200_750_10	13794	399	211

instance	best solution	best solution iter	elapsed ms
vc_100_2000_01	7027	1189	3158
vc_100_2000_02	6853	20000	2970
vc_100_2000_03	6680	20000	3032
vc_100_2000_04	6890	6535	2984
vc_100_2000_05	7236	8812	3093
vc_100_2000_06	6876	2080	2873
vc_100_2000_07	7322	4753	3103
vc_100_2000_08	7163	1090	2930
vc_100_2000_09	6829	793	3073
vc_100_2000_10	7238	1882	2977
vc_100_500_01	6739	5446	252
vc_100_500_02	7205	496	225
vc_100_500_03	6764	10331	220
vc_100_500_04	6818	9802	242
vc_100_500_05	7185	2278	254
vc_100_500_06	7228	3961	237
vc_100_500_07	6763	194	81
vc_100_500_08	6788	397	250
vc_100_500_09	6894	193	226
vc_100_500_10	6670	2773	244
vc_200_3000_01	13795	200	2648
vc_200_3000_02	13790	20000	4038
vc_200_3000_03	14245	8956	4144
vc_200_3000_04	15082	598	2681
vc_200_3000_05	14095	9354	4136
vc_200_3000_06	13731	2588	4196
vc_200_3000_07	14247	16717	4083
vc_200_3000_08	14197	200	3903
vc_200_3000_09	14294	19901	4141
vc_200_3000_10	13907	4976	3982
vc_800_10000	56404	6393	52037

Una delle difficoltà più grandi trovate durante l'implementazione è stato quello di determinare se i criteri di scelta fossero effettivamente robusti o no. Sarebbe interessante proseguire con lo studio del problema continuando sui seguenti punti:

- implementare altri criteri per la selezione del nodo;
- implementare altri criteri di accettazione;
- perfezionare l'algoritmo su istanze piccole;
- implementare altre soluzioni di partenza.

References

[1] An Effective Algorithm for Minimum Weighted Vertex Cover Problem

[2] Two approximation algorithm for Vertex Cover

[3] A fast heuristic for the minimum weight vertex cover problem

[4] An Effective Algorithm for Minimum Weighted Vertex Cover Problem

[5] A memory-based iterated local search algorithm for the multi-depot open vehicle routing problem

[6] A fast euristic for the minimum weight vertex cover problem