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 implementato in Java senza l'utilizzo librerie esterne eccetto `matplotlib4j` per la generazione dei grafici di convergenza e `CSVWriter` per la generazione del `.csv` dei ottenuti (non sono stati caricati tutti per evitare di appesantire troppo la repository).

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, soprattutto 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

È stato trovato opportuno implementare un algoritmo greedy per costruire la soluzione iniziale. Sono stati effettuate anche delle prove utilizzando come punto la soluzione *peggiore* (che contiene tutti i nodi dell'istanza) con la quale non si sono ottenuti buoni risultati

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 o swap di un vertice. 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 conseguenza in una prima fase si è deciso di implementare la seguente:

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
     return perturbed solution
  end if
```

Tuttavia è stato notato, durante le fasi di sviluppo, che questo operatore tendeva a perdere efficacia dopo qualche iterazione, 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
  return perturbed solution
```

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 soprattutto su istanze *medie* e *grandi*.

5 Criterio di accettazione

Per evitare che vengano accettate 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.

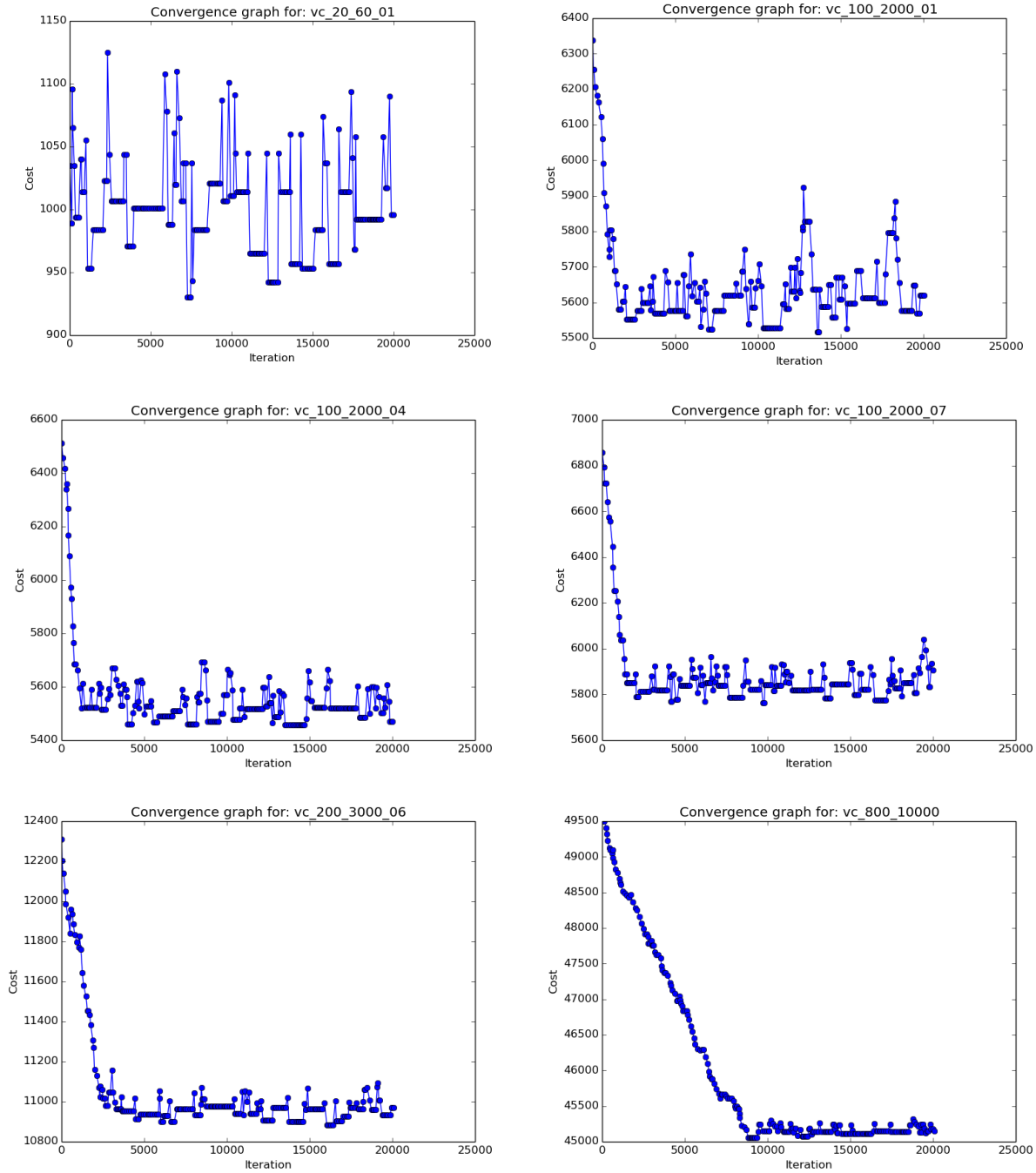

Figure 1: Sample grafici di convergenza

Per quanto riguarda l'istanza LPI, i valori si riferiscono ad stata inserita una media su 10 run indipendenti dell'istanza.

instance	best solution	best solution iter
vc_800_10000.txt	45058	8841
vc_25_150_10.txt	1144	14059
vc_25_150_09.txt	1364	17960
vc_25_150_08.txt	1246	12629
vc_25_150_07.txt	1465	6028
vc_25_150_06.txt	1211	12562
vc_25_150_05.txt	1337	18720
vc_25_150_04.txt	1435	10635
vc_25_150_03.txt	1324	16454
vc_25_150_02.txt	1144	13880
vc_25_150_01.txt	1335	10656
vc_20_60_10.txt	960	2251
vc_20_60_09.txt	1149	3083
vc_20_60_08.txt	1113	2001
vc_20_60_07.txt	1033	1926
vc_20_60_06.txt	980	944
vc_20_60_05.txt	934	11274
vc_20_60_04.txt	819	7002
vc_20_60_03.txt	750	2788
vc_20_60_02.txt	1077	893
vc_20_60_01.txt	930	7276
vc_20_120_10.txt	1054	2723
vc_20_120_09.txt	1160	20093
vc_20_120_08.txt	1099	17086
vc_20_120_07.txt	945	2289
vc_20_120_06.txt	919	2064
vc_20_120_05.txt	939	18957
vc_20_120_04.txt	1031	3640
vc_20_120_03.txt	960	648
vc_20_120_02.txt	1028	19134
vc_20_120_01.txt	898	16987

instance	best solution	best solution iter
vc_200_750_10.txt	9506	5725
vc_200_750_09.txt	10194	12447
vc_200_750_08.txt	9977	10051
vc_200_750_07.txt	8962	19990
vc_200_750_06.txt	9401	18489
vc_200_750_05.txt	10469	16295
vc_200_750_04.txt	9585	18489
vc_200_750_03.txt	9967	18263
vc_200_750_02.txt	10122	18489
vc_200_750_01.txt	9897	18489
vc_200_3000_10.txt	11027	12843
vc_200_3000_09.txt	11412	16213
vc_200_3000_08.txt	11265	4097
vc_200_3000_07.txt	11290	11685
vc_200_3000_06.txt	10885	16032
vc_200_3000_05.txt	11193	6571
vc_200_3000_04.txt	12054	12688
vc_200_3000_03.txt	11298	10718
vc_200_3000_02.txt	10938	17094
vc_200_3000_01.txt	11015	19143
vc_100_500_10.txt	5234	15582
vc_100_500_09.txt	5202	15612
vc_100_500_08.txt	5194	16926
vc_100_500_07.txt	5584	7796
vc_100_500_06.txt	5478	10191
vc_100_500_05.txt	5476	8037
vc_100_500_04.txt	5309	13155
vc_100_500_03.txt	5101	13155
vc_100_500_02.txt	5531	18855
vc_100_500_01.txt	5299	12145
vc_100_2000_10.txt	5676	16365
vc_100_2000_09.txt	5351	2727
vc_100_2000_08.txt	5633	14463
vc_100_2000_07.txt	5764	9729
vc_100_2000_06.txt	5394	15683
vc_100_2000_05.txt	5671	16801
vc_100_2000_04.txt	5458	13524
vc_100_2000_03.txt	5255	11275
vc_100_2000_02.txt	5388	2967
vc_100_2000_01.txt	5517	13586

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;
- implementazione e studio di altri criteri di accettazione;
- perfezionare l'algoritmo su istanze piccole;
- testare implementazione con 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