Design Techniques for Approximation Algorithms and Approximation Classes

Summary

- Local search technique
- Linear programming based algorithms
  - Relaxation and rounding
  - Primal-dual

Local search technique

- Local search technique consists in the improvement of a initial solution (found using some other algorithm) by moving to a better “neighbour” solution.
- It's suitable for problem where an initial feasible solution can be determined in efficient way and where the neighbourhood structure of a feasible solution is known.

Local search technique

- Local search scheme

```
Input: Instance x;
begin
    y = initial feasible solution;
    while there exists a neighbour solution z of y better than y do
        y = z;
end
Output: Locally optimal solution z
```

- For problem NP-hard, we do not expect to find neighbourhood structure that allow us to find an optimal solution in polynomial time (unless P=NP)

Local search technique

- The most famous local search algorithm is the Simplex Method of Dantzig (1947) for LINEAR PROGRAMMING problem.
- Other example of local search algorithm is the Augmenting Path algorithm (and its variants) for MAX FLOW problem.

MAXIMUM CUT

- INSTANCE: Graph $G=(V,E)$

- SOLUTION: Partition of V into disjoint sets V_1 and V_2

- MEASURE: Cardinality of the cut, i.e., the number of edges with one endpoint in V_1 and one endpoint in V_2

Local search technique

- Initial solution: $V_1 = \varnothing$, $V_2 = V$
- Neighbourhood structure *N* of (V_1, V_2) consists of ALL partition (V_{1k}, V_{2k}) for $k=1,\ldots,|V|$ s.t.:
  - If $v_k \in V_1$ then $V_{1k} = V_1 - \{v_k\}$ and $V_{2k} = V_2 \cup \{v_k\}$
  - If $v_k \notin V_1$ then $V_{1k} = V_1 \cup \{v_k\}$ and $V_{2k} = V_2 - \{v_k\}$

Local search technique

- Polynomial-time 2-approximation algorithm for MAXIMUM CUT

```
begin
        V1:=ø;
        repeat
        if exchanging one node between V1 and V2 =V- V1 improves the cut then
        perform the exchange;
        until a local optimum is reached;
        return f
end.
```

Local search technique

- Proof
  - We prove that any local optimum contains at least half of the *m* edges
  - Notation:
    - c= # of edges of the cut
    - i= # of edges inside V_1
    - o= # of edges outside V_1
    - $m=c+i+o$, that is, $i+o=m-c$
  - For any node v,
    - $i(v)$= # of edges between v and a node in V_1
    - $o(v)$= # of edges between v and a node not in V_1

Local search technique

- Proof (continued)
  - V_1 is a local optimum: for any $v \in V_1$, $i(v)$-$o(v) \leq 0$ and,for any v not in V_1,$o(v)$-$i(v) \leq 0$
  - Summing over all nodes in V_1, we have

    $$2i\text{-}c \leq 0$$

  - Summing over all nodes not in V_1, we have

    $$2o\text{-}c \leq 0$$

  - That is, $i+o\text{-}c \leq 0$
  - That is, $m\text{-}2c \leq 0$
  - That is, $c \geq m/2$

Linear programming based algorithms

- A linear program can be solved in polynomial time
- We do not expect that NP-hard problem can be formulated as a linear programming problem with polynomial number of constrains

but

- some NP-hard problems can be formulated as
  - INTEGER LINEAR PROGRAMMING problems or
  - LINEAR PRGRAMMING problems with exponential number of constrains

Linear programming based algorithms

- INTEGER LINEAR PROGRAMMING problems
  - can be solved in approximate way rounding the solution of a linear program
- LINEAR PRGRAMMING problems with exponential number of constrains
  - Can be solved in approximate way by primal-dual algorithms

Relaxation and rounding technique

- Polynomial-time 2-approximation algorithm for weighted version of MINIMUM VERTEX COVER
  - formulate the problem as linear integer programming
  - solve the relaxation
  - select nodes that have been chosen *enough*

Relaxation and rounding technique

- IPL formulation

$$min \sum_{v_i \in V} w_i x_i$$

$$x_i + x_j \geqslant 1 \quad \forall (v_i, v_j) \in E$$

$$x_i \in \{0,1\} \quad \forall v_i \in V$$

Relaxation and rounding technique

- LP relaxation and rounding

$$\min \sum_{v_i \in V} w_i x_i$$

$$x_i + x_j \geqslant 1 \quad \forall (v_i, v_j) \in E$$

$$0 \leqslant x_i \leqslant 1 \quad \forall v_i \in V$$

- Final solution: $U = \{i : x_i \geq 0.5\}$

Relaxation and rounding technique

- Proof
  - The solution is feasible
    - Otherwise, one edge (i,j) is not filled (that is, $x(i)+x(j)<1$)
  - The solution has measure at most twice the optimum of LP relaxation
    - $m^*_{LP}(G) \leq m^*_{IPL}(G)$
    - $w(U) \leq 2\, m^*_{LP}(G) \leq 2\, m^*_{IPL}(G)$
  - The set of feasible solutions is larger

Primal-dual technique

- Classical approach for solving exactly combinatorial optimization problems
  - Weighted combinatorial problems are reduced to purely combinatorial, unweighted problems
- Examples: Dijkstra (shortest path), Ford and Fulkerson (maximum flow), Edmonds (maximum matching)
- Polynomial-time 2-approximation algorithm for weighted version of MINIMUM VERTEX COVER

Primal-dual technique

- LP formulation

$$\text{minimize} \quad \sum_{i \in V} w(i)x(i)$$

$$\text{subject to} \quad x(i) + x(j) \geq 1 \qquad (i,j) \in E$$

$$0 \leq x(i) \leq 1 \qquad i \in V$$

Primal-dual technique

- Dual problem

$$\text{maximize} \sum_{e \in E} y(e)$$

$$\text{subject to} \quad \sum_{j \in N(i)} y(i,j) \leq w(i) \qquad i \in V$$

$$y(e) \geq 0 \qquad e \in E$$

where $N(i)$ denotes the neighborhood of i

Primal-dual technique

- 2-approximation algorithm
  - Simultaneously maintains a (possibly unfeasible) integer solution of LP formulation and a (not necessarily optimal) feasible solution of dual problem
    - At each step integer solution becomes *more* feasible and dual solution has better measure
    - Ends when integer solution becomes feasible

begin
    y=0; U=∅;
    while a not covered edge (i,j) exists
  increase y(i,j) until either i or j is filled
  if i (resp. j) is filled then put i (resp. j) in U
end.

Primal-dual technique

- Proof
  - Feasibility: trivial
  - Performance ratio:
    - For any $i \in U$, the ith constraint is tight.
    - Sum C of the weights of the nodes in U is equal to the sum P of the profit of the incident edges
    - P is at most twice the sum of the profit of all edges which is at most equal to the maximum profit
    - By duality, maximum profit is equal to minimum weight
- Time complexity:
  - At most n iterations, where n is the number of nodes

Dynamic programming technique

- It is an algorithm technique that can make possible to reduce the size of the search space
- It can be applied to all combinatorial problems where optimal solution can be derived by composing optimal solutions of a limited set of subproblems (not always disjoints)

Dynamic programming technique

- For efficiency reasons, it is implemented in a bottom-up way
  - Subproblems are defined with just a few indices (usually 2,3)
  - Subsolutions are optimally extended by means of iterations over this indices
    - Subolutions are stored in a matrix

MINIMUM PARTITION

- INSTANCE: Finite set X of items, for each $x_i \in X$ a positive integer weight a_i

- SOLUTION: A partition of X into 2 disjoint sets Y_1, Y_2

- MEASURE: Maximum between the sum of the weights of elements in V_1 and the sum of the weights of elements in V_2

Dynamic programming technique

- Pseudo-polynomial time algorithm for MINIMUM PARTITION:
  - T: n x b-matrix (b=sum of the weights of all n elements)
  - $T(i,j)$=TRUE if a subset of $\{a_1,\ldots,a_i\}$ exists whose sum is j
  - Construction of T: $T(i+1,j)=T(i,j)$ **or** $T(i,j-a_i+1)$
  - Final answer to the evaluation problem:
    - select true element of nth row of T that minimizes $\max(j,b-j)$
- Complexity: $O(nb)=O(n^2a_{\max})$, where $a_{\max}$ is the maximum weight
  - Can be modified to obtain a feasible solution

Dynamic programming technique

- The approximation algorithm
  - Ignore the last *t* digits of the numbers
  - Apply the pseudo-polynomial time algorithm
  - Return the corresponding solution in the original instance

Dynamic programming technique

- Performance ratio:
  - $m(x, y^*(x')) - m^*(x) \leq 10^t n$
    where $y^*(x')$ denotes an optimal solution for scaled instance x'
  - Performance ratio is at most $1+10^t n/m^*(x)$
  - $a_{max} \leq m^*(x) \leq n\, a_{max}$ ($a_{\max}$ = max value of items)
  - $m^*(x) \leq a_{max}/(a_{max} - n\, 10^t)\, m(x, y^*(x'))$
  - For any r, if we choose $t=\log_{10}(a_{\max}(r-1)/\mathrm{r}\, n)$, then the performance ratio is at most r
- Time complexity:
  - $O(n^2 a'_{\max}) = O(\mathrm{r}n^3/(r-1))$