Local Search Strategies

Summary

- Greedy Local Search
- Random Walk Strategies
- Local Search on cycle cutset

Greedy Local Search... in short

G.B. Shaw

A life spent doing mistakes is not only more honorable, but more useful than a life spent doing nothing

Solution Techniques Up to Now

Solution Techniques

- Backtracking
  - Start from empty assignment
  - Build local consistent solutions
- Inference
  - force local consistency
  - make constraints explicit

Local Search

Basic Ideas

- start from a complete (inconsistent) assignment to all variables
- assignment can satify some constraints and violate others
- modify the value on a subset of variables (most cases one) such that number of violated constrained is reduced
- until all constraints are satisfied or we tried long enough

Example

Example (Local Search)

- Variables: x_1, x_2, x_3, x_4 Domains: $D_i = \{0, 1, 2, 3\}$
- Constraints: $x_1 < x_3, x_2 < x_3, x_3 < x_4$
- First assignment $\{1, 1, 0, 2\}$
- change $x_3 : 0 \rightarrow 2$
- change $x_4 : 2 \rightarrow 3$

Example

Example (Local Search for 4-Queens)

- $(1, 2, 3, 4)$ Conflicts: 6
- $(1, 2, 4, 4)$ Conflicts: 4
- $(1, 2, 4, 1)$ Conflicts: 2
- $(3, 2, 4, 1)$ Conflicts: 1 Local Minima

Why Study Local Search Methods then?

Features of Local Search

- In general can not guarantee completeness
- But are much better on average
- Extremely efficient in terms of memory and computation time
- Augmented with randomisation and heuristics for escaping local minima are also extremely effective

Local Search vs. Backtrack

	n-Queens	k-SAT
Backtrack based	600	few hundreds
Local Search	millions	many thousands

Local Search for TSP

TSP and local search

TSP Optimisation problem: try to find a tour through a set of cities that minimise travel cost visiting each city once.

- Start with a random sequence of cities
- Swap two cities in the sequence to improve solution quality
- Until no improvement possible
- Save the current solution and restart from a new random assignment
- Repeat this process for a given amount of time storing the best solution found

Stochastic Local Search

Procedure

Algorithm 1 SLS

Require: A constraint network $\mathcal{R}$, number of MAXTRIES, cost function
Ensure: A solution or notification that the algorithm could not find one

```
for all i = 1 to MAXTRIES do
    Initialisation: ā = {a_1, ··· , a_n}
    repeat
        if ā is consistent then
            return ā
        else
            Y = {< x_i, a'_i >} set of assignment that maximally improve current solution
            choose one pair < x_i, a'_i >
            ā ← {a_1, ··· , a'_i, ··· , a_n}
        end if
    until current assignment can not be improved
end for

return false
```

SLS for SAT

GSAT

- SLS algorithm for SAT
- Cost of assignment is number of unsatisfied clauses
- Very popular algorithm for SAT

Example

Example (GSAT)

- $\phi = \{(\neg C)(\neg A \vee \neg B \vee C)(\neg A \vee D \vee E)(\neg B \vee \neg C)\}$
- $(1,1,1,1,1)$ Initial assignment $\rightarrow$ cost $= 2$
- $Y = \{< C, 0 >< B, 0 >\}$ both new cost 1
- choose $< C, 0 >$
- $(1,1,0,1,1) \rightarrow$ cost $= 1$
- $Y = \{< B, 0 >< A, 0 >\}$ new cost 0
- $(1,0,0,1,1) \rightarrow$ cost $= 0 \rightarrow$ solution

Improvements to SLS

Improving SLS

Trying to avoid local minima

- Plateau Search
- Constraint Weighting
- Tabu Search

Plateau Search

Going Sideways

- Plateau: set of solutions that have same cost
- Local minima can be due to a plateau: SLS stops as soon as a plateau is found
- Keep on changing as long as solution is no worse than current one
- We can still have local minima

Example

Example (Plateau Search)

- Variables: x, y, z, k, s, r Domains: $D_i = \{0, 1, 2, 3\}$
- Constraints: $x < z, y < z, z < k, k < r, k < s$
- Initial Assignment $(0, 0, 1, 1, 2, 2) \rightarrow$ cost $= 1$ $z < k$
- No single variable change improves the cost but a solution exists $(0, 0, 1, 2, 3, 3)$
- Changing to a same cost assignment we can find the solution

Example: Cycles in Plateau Search

Example (Plateau Search and Cycles)

- Variables: x, y, z, k, m, r, s Domains: $D_i = \{0, 1, 2, 3\}$
- Constraints:
  $x = z, x = y, y = z, z < k, k < m, m = r, r = s, m = s$
- Initial Assignment $(0, 0, 0, 1, 1, 1, 1) \rightarrow$ cost $= 1$ $k < m$
- Modifying any variable in $\{x, y, z, m, r, s\}$ results in at least two violated constraints
- Setting $k = 0$ cost is constant but now $z < k$ is violated
- The only modification with constant cost is $k = 1 \rightarrow$ cycle

Constraint Weighting

Breakout Method

- Cost function: $F(\bar{a}) = \sum_i w_i C_i(\bar{a})$
- w_i current cost weight, $C_i(\bar{a}) = 1$ iff $\bar{a}$ violates constraint C_i
- Find local modification that maximise the decrement of F
- When local minima adjust weights increasing by one violated constraints
- Current assignment is no longer a local minima and we can progress towards a solution
- In general is not complete but extremely good empirical results [Morris 93]
- If no solution exists we can still cycle through inconsistent solutions

Example: Constraint Weighting

Example (Constraint Weighting)

- Variables: x, y, z, k, m, r, s Domains: $D_i = \{0, 1, 2, 3\}$
- Constraints:
  $x = z, x = y, y = z, z < k, k < m, m = r, r = s, m = s$
- Initial Assignment $(0, 0, 0, 1, 1, 1, 1) \rightarrow$ cost $= 1$ $k < m$
- Increasing constraints by 1 each time a local minima is reached we can find a solution

Example: Inconsistent Problem

Example (Constraint Weighting and Inconsistency)

- Variables: x, y, z Domains: $D_i = \{R, B\}$
- Constraints: $x! = y, x! = z, y! = z$
- Initial Assignment $(R, B, R) \rightarrow$ cost $= 1$
- Lifting weights results in cycling over inconsistent states

Tabu Search

Tabu Search

- Preventing backward moves
- Build a queue of last *n* assignments <variable,value>
- Assignments in the list are forbidden
- Forget the *oldest* assignment when the queue is full

Example: Tabu search

Example (Tabu Search)

- Variables: x, y, z, k, m, r, s Domains: $D_i = \{0, 1, 2, 3\}$
- Constraints:
  $x = z, x = y, y = z, z < k, k < m, m = r, r = s, m = s$
- Initial Assignment $(0, 0, 0, 1, 1, 1, 1) \rightarrow$ cost $= 1$ $k < m$
- Fix $n = 4$, first three moves $k = 0, k = 1, k = 2$ with constant cost
- Then k can not be changed anymore and we can get out of the cycle e.g. $m = 2$
- We could make a wrong choice $z = 1$
- If list is long enough at some point we will change m
- If list is too long we can make good paths longer

Random Walk Strategies

Random Walk for CP

- Include random moves in the search process
- Instead of making always greedy steps sometimes move at random
- Increase probability of escaping local minima
- Difference with greedy: sometimes modifying an unsatisfied assignment might cause more harm than doing nothing, greedy would not do such moves

Example (Pure Random Walk for SAT)

- Start from random assignment of all literals
- Randomly select an unsatisfied clause (constraint)
- Flip (e.g., T $\rightarrow$ F or F $\rightarrow$ T) the assignment of one random literal (variable) in the constraint [This might be harmful]

WalkSAT

WalkSAT

- Random walk variant for SAT, can be extended for generic problems, extremely succesful in practice
- Select a constraint violated by current assignment
- Make a random choice between:
  1. change the value in one of the variables in the violated constraint
  2. greedily minimise the total number of constraints when value of the variable is changed (break value)
- Value p give probability of taking choice 1 ($1 - p$ is probability of choice 2)
- In general, step 2 should consider also the selected constraint when minimising number of violated constraints
- For SAT, changing the value of a variable that appears in an unsatisfied clause will automatically satisfy the clause

WalkSAT Procedure

Algorithm

Algorithm 2 WalkSAT

Require: A constraint network $\mathcal{R}$, number of MAXTRIES, MAXFLIPS, probability p
Ensure: A solution or notification that the algorithm could not find one and the best assignment found

```
Initialisation: b̄ = {a_1, ··· , a_n} random assignment
for all i = 1 to MAXTRIES do
    Initialisation: ā = {a_1, ··· , a_n} random assignment
    for all i = 1 to MAXFLIPS do
        if ā is consistent then
            return true and ā
        else
            selecte a violated constraint C
            if randomNumber < p then
                select an arbitrary assignment x ← a′
            else
                select an assignment x ← a′ that minimises the number of new violated
                constraints (considering also C)
            end if
            ā ← ā with x ← a′
        end if
    end for
    b̄ ← best assignment between ā and b̄
end for

return false and b̄
```

Simulated Annealing

Main ideas

- Inspired by statistical mechanics
- Main idea: probability of making a random move is a function of the execution time steps
- Allow more random moves at the beginning
  - we can reach zones with better solutions
- Diminish probability of having a random move towards the end
  - refine search to find a complete solution

Simulated Annealing

More Details

- At each step select a variable $x = a$ and a new value a' and compute δ
- $\delta = F(x = a') - F(x = a)$ where F is the cost function
- If $\delta \leq 0$ change the value of x to a' (we are minimising)
- Otherwise change the value only with probability $e^{-\delta/T}$
- T is a parameter (called temperature) that is decreased with time (cooled)
- For given cooling schedules (i.e. if T is decreased gradually enough) the algorithm is guaranteed to converge to the exact solution

Properties of Local Search

Most important property of local search approaches

- always teminate at a local minima
- anytime: the longer they run the better solution they provide

General property of random walk

- consider a random walk on a line starting at the origin
- take a left or right move with 0.5 chances
- it can be shown that on average after L^2 steps the walk will reach a point distant L from origin.

Properties of random walk for SAT

Properties for 2-SAT

- For 2-SAT Random Walk is guaranteed to converge on formulas with n literals, with probability 1 after $O(n^2)$ steps
- A random assignment is on average $n/2$ flips away from a satisfying assignment
- There is $1/2$ prob. that a flip on a 2-Clause will reduce the distance by 1
- On average a random walk will reach a satisfying assignment in $O(n^2)$ steps
- This is not true for 3-SAT
  - probability of reaching a satisfying assignment will reach one after an exponential number of steps
  - other methods might never reach the assignment e.g. GSAT
- Empirical evaluations show very good performance when compared with complete algorithm

SLS with cycle cutset decomposition

Hybrid approach

1. Find a cycle cutset
2. Fix an assignment for the cutset variables (this leaves a forest of unassigned subproblems)
3. Force arc consistency on each tree and propagate constraints. If solution found stop.
4. Otherwise stochastic local search on cutset variables only.
5. If improvement go to step 3
6. Otherwise stop

Tree Decomposition based on cycle cutset

Decomposing the problem

Idea: given an assignment for cutset cycle variables find an assignment of other variables that minimises number of violated constraints

- Partition X in $\{Z, Y\}$
- Y cutset variables, $Z = X \setminus Y$ tree variables
- $\bar{a}_y$ current assignment of cycle cutset variables
- Divide the problem in rooted sub-trees
- Duplicate cutset variables for each neighbour and assign current value ($\bar{a}_y[y_i]$)

Tree Inference based on cycle cutset

Propagating constraints

- $C_{z_i}(a_i, \bar{a}_y)$ number of violated conflicts in the tree rooted at the tree variable z_i
- $C(\bar{a}_z, \bar{a}_y)$ number of violated constraint for overall problem with assignments $\bar{z}$ and $\bar{y}$
- Want to compute $C_{min} = \min_{Y=y} \min_{Z=z} C(z, y)$
- General Idea
  - compute number of violated constraint from leaves to root
  - choose assignment at root
  - propagate assignment from root to leaves

Tree Algorithm

Tree Algorithm

Algorithm 3 Tree

Require: An arc consistent network $\mathcal{R}$, a partition of $X = \{Z, Y\}$, an assignment for cutset variables $\bar{a}_y$

Ensure: An assignment $\bar{a}_z$ such that $C(\bar{a}_z, \bar{a}_y) = C^T_{min}(\bar{a}_y)$

Initialisation: $C_{y_i}(\bar{a}_y[y_i], \bar{a}_y) = 0$ for all $y_i \in Y$

going from leaves to root on the tree compute:

for all variable z_i and every value $a_i \in D_{z_i}$ **do**

  compute $C_{z_i}(a_i, \bar{a}_y)$

end for

going from root to leaves on the tree compute:

for all every tree variable $z_i \in Z$ let D_{z_i} its consistent values with its parent's value v_{p_j} **do**

  compute best a_i^*

end for

return $\{< z_1, a_1 >, \cdots, < z_k, a_k >\}$

Main Computation for Tree Algorithm

Main Computation

Computations performed in the algorithm

- $C_{z_i}(a_i, \bar{a}_y) = \sum_{z_j | z_j \text{ child of } z_i} \min_{a_j \in D_{z_j}} (C_{z_j}(a_j, \bar{a}_y) + R_{z_i, z_j}(a_i, a_j))$
- $R_{z_i, z_j}(a_i, a_j) = 0$ if $< a_i, a_j > \in R_{z_i, z_j}$; 1 Otherwise
- $a_i^* = \arg\min_{a_i \in D_{z_i}} C_{z_i}(a_i, \bar{a}_y) + R_{z_i, p_j}(a_i, v_{p_j})$
- p_j parent of z_i

Example for Tree Algorithm

Example (Execution of Tree Algorithm)

- Variables: x_1, x_2, x_3, x_4, x_5 Domain: R, B, Y
- Constraints: $x_1 != x_2, x_1 != x_3, x_1 != x_4, x_1 != x_5, x_2 != x_3, x_3 != x_5, x_4 != x_5$
- Cycle Cutset variables $Y = \{x_3, x_5\}$
- Assignment for cycle cutset variables $\bar{a}_y = x_3 = R, x_5 = B$

SLS with cycle cutset

SLS + CC

- Replace backtracking with local search
- Start random initial assignment
- Perform a given number of TRY
- within each TRY alternate between SLS and TREE:
  - Fix one assignment for CC variables, perform TREE
  - Fix the given assignment for tree variables and perform SLS on CC

SLS with cycle cutset: performance

Behaviour of SLS and CC

- Empirical result on randomly generated instances: SLS + CC Not always better than SLS
- Empirical evidence show that behavior depends on ratio of CC variables
- Crossing point around 36%
- Not completely clear how general these results are

Properties of stochastic Local Search

General Features

- Anytime: the longer they run the better the solution
- Local Minima: guaranteed to terminate at local minima
- Not complete: can not be used to prove inconsistency