5. Simulated Annealing
5.2 Advanced Concepts

Fall 2010

Instructor: Dr. Masoud Yaghini

Outline

- Acceptance Function
- Initial Temperature
- Equilibrium State
- Cooling Schedule
- Stopping Condition
- Handling Constraints
- An Exercise
- References

Acceptance Function

Acceptance Function

- SA can escape from local optima due to the probabilistic acceptance of a nonimproving neighbor.
- The probability of accepting is dependent of:
  - The temperature T
  - The change of the objective function ΔE

Acceptance Function

- The acceptance probability of a nonimproving move is:

$$P(\Delta E, T) = e^{\frac{-\Delta E}{T}} > R$$

  - where E is the change in the evaluation function,
  - T is the current temperature, and
  - R is a uniform random number between 0 and 1.

To accept or not to accept?

Change	Temperature	exp(-ΔE/T)
0.2	0.95	0.8101
0.4	0.95	0.6563
0.6	0.95	0.5317
0.8	0.95	0.4308
0.2	0.10	0.1353
0.4	0.10	0.0183
0.6	0.10	0.0024
0.8	0.10	0.0003

Acceptance Function

- At high temperatures,
  - the probability of accepting worse moves is high.
  - If $T = \infty$, all moves are accepted
  - It corresponds to a random local walk in the landscape.
- At low temperatures,
  - The probability of accepting worse moves decreases.
  - If $T = 0$, no worse moves are accepted
  - The search is equivalent to **local search**.
- Moreover, the probability of accepting a large deterioration in solution quality decreases exponentially toward 0.

Initial Temperature

Initial Temperature

- If the starting temperature is very high,
  - the search will be a random local search for a period of time
  - accepting all neighbors during the initial phase of the algorithm.
  - The main drawback of this strategy is its high computational cost.
- If the initial temperature is very low,
  - the search will be a local search algorithm.
- Temperature must be high enough to allow moves to almost neighborhood state.
- Problem is finding a suitable starting temperature

Initial Temperature

- **Finding a suitable starting temperature methods:**
  - Acceptance deviation method
  - Tuning for initial temperature method

Initial Temperature

- **Acceptance deviation method**
  - The starting temperature is computed using preliminary experimentations by: $k\sigma$
  - where
    - σ represents the standard deviation of difference between values of objective functions and
    - $k = -3/\ln(p)$ with the acceptance probability of p, which is greater than 3σ

Initial Temperature

- **Tuning for initial temperature method:**
  - Start high, reduce quickly until about 60% of worse moves are accepted.
  - Use this as the starting temperature

Equilibrium State

Equilibrium State

- Once an equilibrium state is reached, the temperature is decreased.
- To reach an equilibrium state at each temperature, a number of sufficient transitions (moves) must be applied.
- The number of iterations must be set according to:
  - The size of the problem instance and
  - Particularly proportional to the neighborhood size |N(s)|

Equilibrium State

- The strategies that can be used to determine the number of transitions visited:
  - **Static strategy**
  - **Adaptive strategy**

Equilibrium State

- **Static strategy**
  - The number of transitions is determined before the search starts.
  - For instance, a given proportion y of the neighborhood N(s) is explored.
  - Hence, the number of generated neighbors from a solution s is $y \cdot |N(s)|$.
  - The more significant the ratio y, the higher the computational cost and the better the results.

Equilibrium State

- **Adaptive strategy**
  - The number of generated neighbors will depend on the characteristics of the search.
  - One adaptive approach is an improving neighbor solution is generated.
  - This feature may result in the reduction of the computational time without compromising the quality of the obtained solutions
  - Another approach is achieving a predetermined number of iterations without improvement of the best found solution in the inner loop with the same temperature

Cooling Schedule

Cooling Schedule

- In the SA algorithm, the temperature is decreased gradually such that $T_i > 0, \forall i$
- There is always a compromise between the quality of the obtained solutions and the speed of the cooling schedule.
- If the temperature is decreased slowly, better solutions are obtained but with a more significant computation time.

Cooling Schedule

- The temperature can be updated in different ways:
  - **Static Strategy**
    - **Linear**
  - **Dynamic Strategy**
    - **Geometric**
    - **Logarithmic**
  - **Adaptive Strategy**

Cooling Schedule

- **Linear**
  - In the trivial linear schedule, the temperature T is updated as $T = T - \beta$, where β is a specified constant value.
  - Hence, we have

    $$T_i = T_0 - i \times \beta$$

  - where T_i represents the temperature at iteration i.
  - β is a specified constant value
  - T_0 is the initial temperature

Cooling Schedule

- **Geometric**
  - In the geometric schedule, the temperature is updated using the formula

    $T_{i+1} = \alpha.T_i$
  - where $\alpha \in]0, 1[$.
  - It is the most popular cooling function.
  - Experience has shown that α should be between 0.5 and 0.99.

Cooling Schedule

- **Logarithmic**
  - The following formula is used:

$$T_i = \frac{T_0}{\log(i+10)}$$

  - This schedule is too slow to be applied in practice but has the property of the convergence proof to a global optimum.

Cooling Schedule

- **Adaptive Strategy**
  - Most of the cooling schedules are static or dynamic in the sense that the cooling schedule is defined completely a **priori**.
  - In this case, the cooling schedule is "blind" to the characteristics of the search landscape.
  - In an adaptive cooling schedule, the decreasing rate is depends on some information obtained during the search.

Stopping Condition

Stopping Condition

- Concerning the stopping condition, theory suggests a final temperature equal to 0.
- In practice, one can stop the search when the probability of accepting a move is negligible.

Stopping Condition

- The following stopping criteria may be used:
  1. Reaching a final temperature T_F is the most popular stopping criteria.
     - This temperature must be low (e.g., $T_{min} = 0.01$).
  2. Achieving a predetermined number for successive temperature values no improvement in solution quality
  3. After a fixed amount of CPU time
  4. When the objective reaches a pre-specified threshold value

Handling Constraints

Handling Constraints

- Constraints cannot handled implicitly
  - Penalty function approach should be used
- **Constraints**
  - **Hard Constraints**: these constraints cannot be violated in a feasible solution
  - **Soft Constraints**: these constraints should, ideally, not be violated but, if they are, the solution is still feasible

Handling Constraints

- Hard constraints are given a large weighting.
  - The solutions which violate those constraints have a high cost function
- Soft constraints are weighted depending on their importance
- Weightings can be dynamically changed as the algorithm progresses.
  - This allows hard constraints to be accepted at the start of the algorithm but rejected later

An Exercise

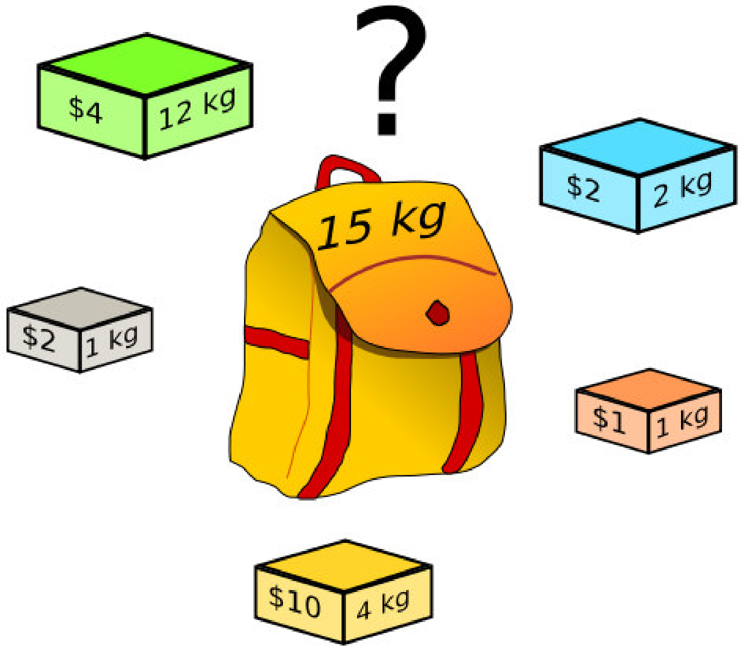
The Knapsack problem

- There are *n* items:
  - Each item *i* has a weight w_i
  - Each item *i* has a value v_i
- The knapsack has a limited capacity of W units.
- We can take one of each item at most

$$\max \sum_i v_i * x_i \qquad i = 1,2,\ldots,n$$

$$\textit{subject to} \quad \sum_i w_i * x_i \le W$$

$$x_i \in \{0,1\}$$



The Knapsack problem

- **Example:**
  - **Item:** **1 2 3 4 5 6 7**
  - **Benefit:** **5 8 3 2 7 9 4**
  - **Weight:** **7 8 4 10 4 6 4**
- **Knapsack holds a maximum of 22 pounds**
- **Fill it to get the maximum benefit**
- The problem description:
  - Maximize $\sum_i v_i$
  - While $\sum_i w_i \le W$

Solution 1

Item	1	2	3	4	5	6	7
Solution	**1**	**1**	**0**	**0**	**1**	**0**	**0**
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Objective: 5 + 8 + 7 = 20
- Weight: 7 + 8 + 4 = 19 <= 22

Solution 2 overweighted

Item	1	2	3	4	5	6	7
Solution	**0**	**1**	**0**	**1**	**0**	**1**	**0**
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Weight: $8 + 10 + 6 = 24 > 22$

References

References

- El-Ghazali Talbi, **Metaheuristics : From Design to Implementation, John Wiley & Sons, 2009.**
- J. Dreo A. Petrowski, P. Siarry E. Taillard, **Metaheuristics for Hard Optimization**, Springer-Verlag, 2006.

The End