Trajectory-based Optimization: Simulated Annealing - II

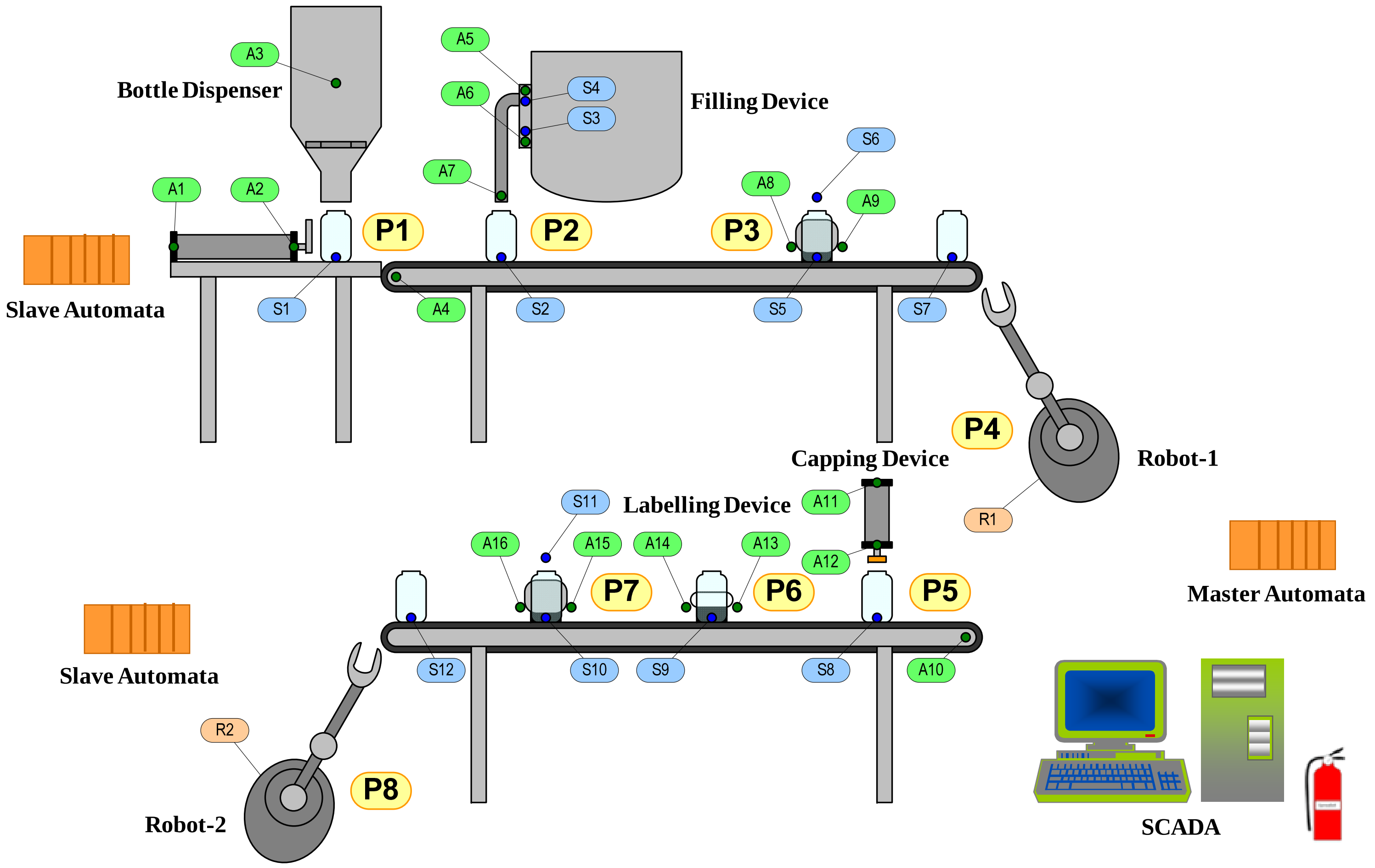
Lecture 8 – Tuesday June 3, 2014

Outline

- Physical Annealing
- Simulated Annealing
- SA Cooling Schedule
- SA for TSP
- SA for PLP
- **SA for Scheduling problems**
- SA for Function Optimization
- Adaptive SA
- Cooperative SA
- Summary

SA for Scheduling problems

- **Job shop scheduling (or job-shop problem)** addresses how to schedule jobs to be processed on machine(s).



Alaa Khamis, (56-10569): Industrial Automation II, 2005-2006, Department of Systems Engineering and Automation, Charles III University of Madrid.

SA for Scheduling problems

- In **Job shop scheduling problem (JSSP)**, n jobs J_1, J_2, …, J_n of varying sizes are given. These jobs need to be scheduled on m machines, while trying to minimize the **makespan**.
- The **makespan** is the total length of the schedule (that is, when all the jobs have finished processing). It is the **completion time of the last job**.

SA for Scheduling problems

- **Two Machine - Three Jobs Example**

  There are three parts, P_1, P_2, and P_3. Each needs to be processed first on Machine M_1, then on Machine M_2.

Machines	Parts		
	P1	P2	P3
M1	6	2	8
M2	4	4	4

  There are 6 possible sequences for the parts:

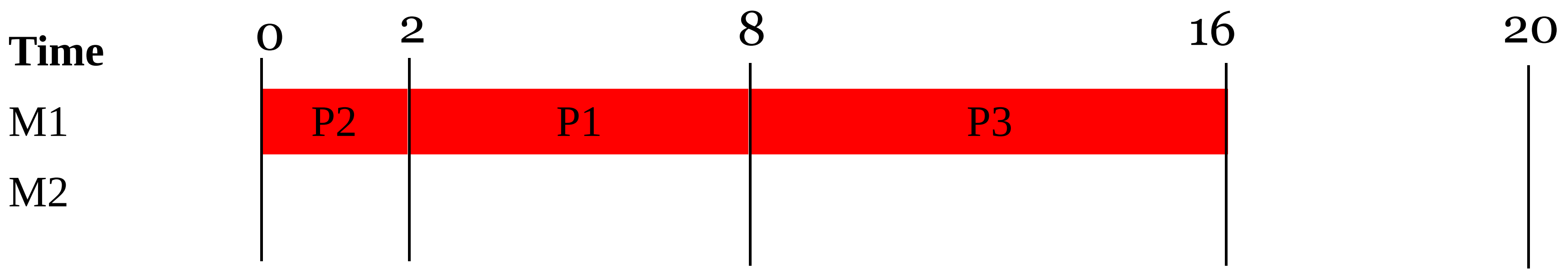
  1-2-3, 1-3-2, 2-1-3, 2-3-1, 3-1-2, and 3-2-1

  For any sequence, **Machine 1** will start working at **T=0**, and work till **T= 6+2+8 = 16**.

[1]

SA for Scheduling problems

- **Two Machine - Three Jobs Example**



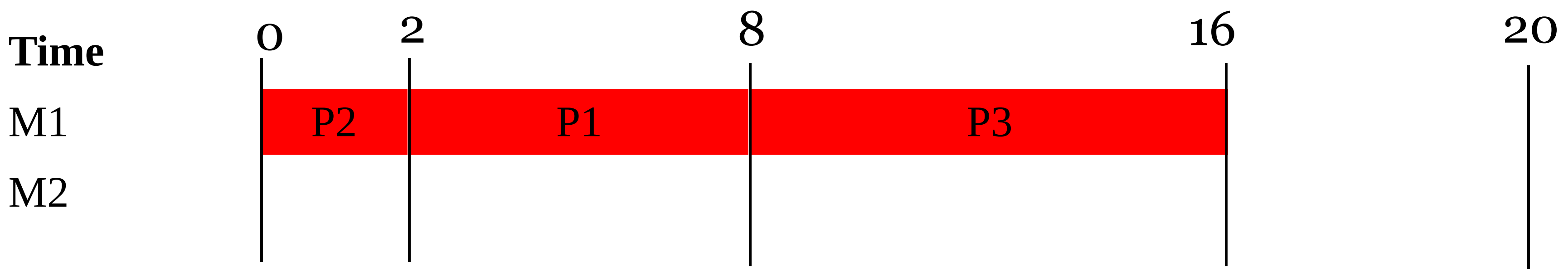
Utilization of M2:

◇ It will not work at the initial period when M1 is doing its first scheduled job. This hints that we should try to do the **shortest job on M1 first**!

[1]

SA for Scheduling problems

- **Two Machine - Three Jobs Example**



Utilization of M2 (cont'd):

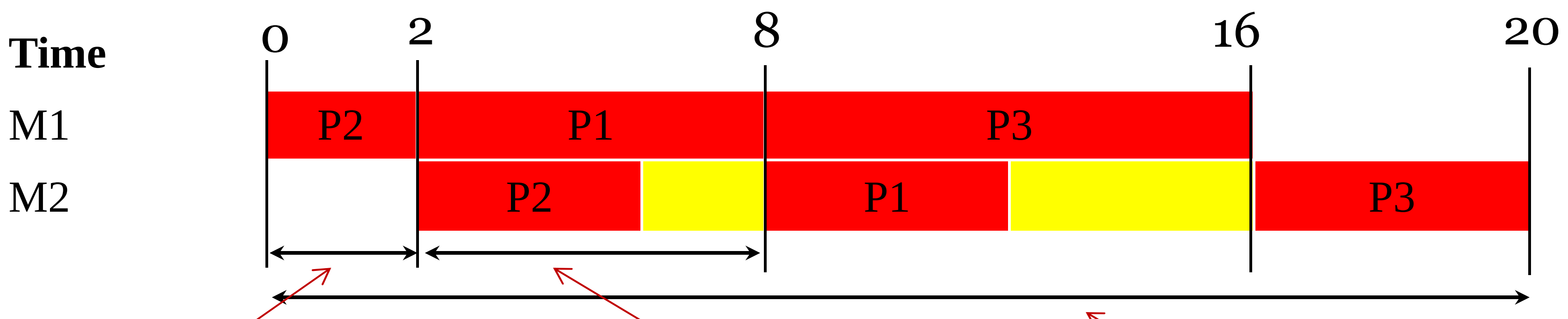
◇ What happens if the Second operation (operation on M2) for a part is **very short** ? In this case, M2 will finish this part, while M1 is still working on the first operation of the next part. This will make **M2 idle** for some time. This hints that we should try to place jobs that have **LONG 2nd operations in the beginning**.

[1]

SA for Scheduling problems

- **Two Machine - Three Jobs Example**

Gantt-chart



M2 will always be idle for this duration

If Operation 2 of P2 (first job) is shorter than this duration, M2 will be idle by the duration shown in yellow

Makespan

(the time between the moment you start the first Job, till the time you end the last Job)

[1]

SA for Scheduling problems

- **Machine environments**
  - ◊ **Single Machine:** Only one machine is available to process jobs. Each job has a single task. Every job is performed on the same machine.
  - ◊ **Parallel Machines:** Multiple machines are available to process jobs. The machines can be identical, of different speeds, or specialized to only processing specific jobs. Each job has a single task.

[3]

SA for Scheduling problems

- **Objective Functions**
  - ◊ Makespan
  - ◊ Machine Utilization
  - ◊ Total (Weighted) Completion Time
  - ◊ Lateness
  - ◊ Tardiness

SA for Scheduling problems

- **Objective Functions: Makespan**

Let C_i denote the **completion time** of the ith job in a batch of n jobs given. The makespan, C_{max}, is defined as,

$$\mathbf{C_{max} = max\ (C_1, C_2, \ldots, C_n)}$$

The makespan is the completion time of the last job.

A common problem of interest is to **minimize C_{max}**,

This criterion is usually used to measure the **level of utilization of the machine**.

[3]

SA for Scheduling problems

- **Objective Functions: Machine Utilization**

Machine utilization reflects the **ratio between available and required machine capacity**.

For a manufacturing company, it is usually desirable that machine **idle times** are kept as **small as possible**. The average machine utilization MU can be computed as

$$MU = \frac{\sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}}{m.C_{\max}}$$

where n=number of jobs, m=number of machines and p_{ik}=processing time.

[9]

SA for Scheduling problems

- **Objective Functions: Total Completion Time**

Let C_i denote the completion time of the ith job in a batch of n jobs given. The **total completion time** is defined as,

$$\sum_{i=1}^{n} C_i$$

The total completion time is the sum of all the completion times of the jobs. It is commonly referred to as the **flow time**.

[3]

SA for Scheduling problems

- **Objective Functions: Total (Weighted) Completion Time**

It may be advisable to rank jobs according to their **importance or order value**. For this purpose, a non-negative weight value w_i has to be assigned to each job. In the default case, all weights w_i are chosen equal to 1.

Let w_i denote the weight assigned to the ith job in the a batch of n jobs given. The **total weighted completion time** is defined as,

$$\sum_{i=1}^{n} w_i C_i$$

[3]

SA for Scheduling problems

- **Objective Functions: Total (Weighted) Completion Time**

  A common problem is in **minimizing** the total (weighted) completion time. This problem allows one to find an indication to the **total holding or inventory** caused by the schedule.

[3]

SA for Scheduling problems

- **Objective Functions: Lateness**

Difference between **completion time and the due date**.

$L_i=C_i-d_i$

where C_i is the completion of job *i* and d_i is the due date of job *i*.

Some of the problems involving lateness are:

1. Minimizing the **total lateness**. The total lateness is defined as,

$$\sum_{i=1}^{n} L_i$$

[3]

SA for Scheduling problems

- **Objective Functions: Lateness**

2. Minimizing the **total weighted lateness**. Let w_i denote the weight assigned to the ith job. The total weighted lateness is defined as,

$$\sum_{i=1}^{n} w_i L_i$$

3. Minimizing the **maximum lateness**, L_{max}. L_{max} is defined as,

$$L_{\max} = \max(L_1, L_2, \ldots, L_n)$$

[3]

SA for Scheduling problems

- **Objective Functions: Tardiness**

The tardiness of job *i*, T_i, is defined as

T_i=Max (0, C_i-d_i)

where

C_i is the completion of job *i* and

d_i is the due date of job i.

Some of the problems involving tardiness are:

1. Minimizing the **maximum tardiness**

$Min(\max(T_i))$

[3]

SA for Scheduling problems

- **Objective Functions: Tardiness**

  2. Minimizing the **number of tardy jobs**. The number of tardy jobs is defined as,

  $$N_T = \sum_{i=1}^{n} \delta(T_i)$$

  where $\delta(x) = 1$ if $x > 0$

  $\delta(x) = 0$ otherwise

  3. Minimizing the **total weighted tardiness**. Let w_i denote the weight assigned to the ith job. The total weighted tardiness is defined as,

  $$\sum_{i=1}^{n} w_i T_i$$

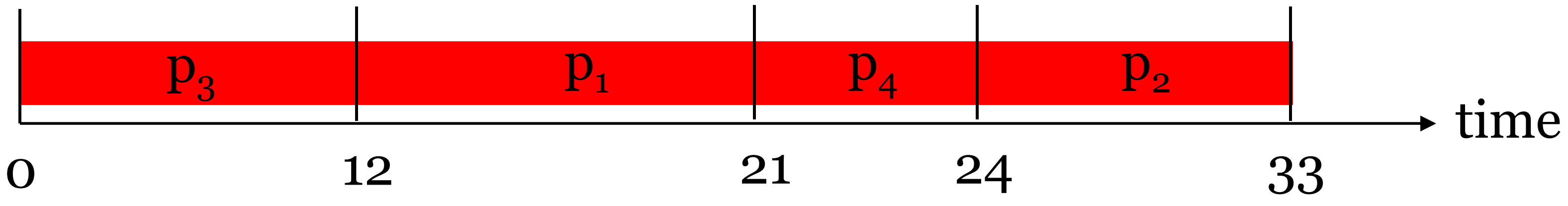
[3]

SA for Scheduling problems

- **Example-1: Single machine, minimize total weighted tardiness**

	Jobs	**1**	**2**	**3**	**4**
Processing	p_i	9	9	12	3
Due	d_j	10	8	5	28
Weight	w_j	14	12	1	12

Let an initial solution be **3, 1, 4, 2**



$$\sum w_j T_j = 1\cdot\max(0, (12-5)) + 14\cdot\max(0, (21-10)) +$$

$$12\cdot\max(0,(24-28)) + 12\cdot\max(0,(33-8) = 461$$

[2]

SA for Scheduling problems

- **Example-1: SA**

	Jobs	**1**	**2**	**3**	**4**
Processing	p_i	9	9	12	3
Due	d_j	10	8	5	28
Weight	w_j	14	12	1	12

Neighborhood by **swapping the order of jobs**. Select one at random

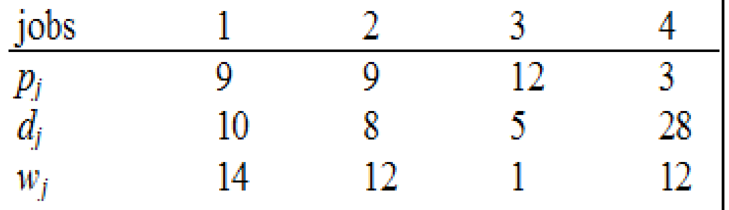
Use **geometric temperature reduction:** $\alpha=0.9$, $T_o=0.9$

Set **number of iterations** at the same temperature=2

[2]

SA for Scheduling problems

- **Example-1: SA – Iteration 1**



jobs	1	2	3	4
p_j	9	9	12	3
d_j	10	8	5	28
w_j	14	12	1	12

Initial solution: **3, 1, 4, 2** *f*=**461**

Consider the following neighbor: **1, 3, 4, 2**

$\sum w_j T_j$ = 14.max(0,(9-10))+1.max(0,(21-5))+

12.max(0,(24-28)+12.max(0,(33-8))=**316**

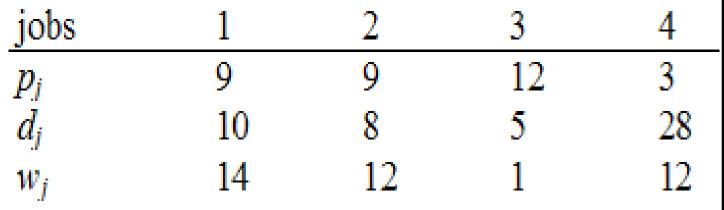
Δf = **-145**

then we accept the new solution as it is improving.

[2]

SA for Scheduling problems

- **Example-1: SA – Iteration 2**



jobs	1	2	3	4
p_j	9	9	12	3
d_j	10	8	5	28
w_j	14	12	1	12

Current solution: **1, 3, 4, 2** f= **316**

Generate a neighbor (say): **1, 4, 3, 2**

$\sum w_j T_j$ = 14.max(0,(9-10))+12.max(0,(12-28))+

1.max(0,(24-5)+12.max(0,(33-8)) = **319**

Δf **= +3** i.e not-improving, apply the acceptance criteria.

Calculate $p = e^{-\Delta f/T} = e^{-(319-316)/0.9} = 0.035$

Generate a random number r between (0,1), say it was r=0.01

p>r then we accept the solution.

[2]

SA for Scheduling problems

- **Example-1: SA – Iteration 2**

Current solution: **1, 3, 4, 2** *f*= **316**

Generate a neighbor (say): **1, 4, 3, 2**

Update the temperature after these two iterations using geometric cooling:

$T=T_o\alpha^i$, T= 0.9x0.9=0.81

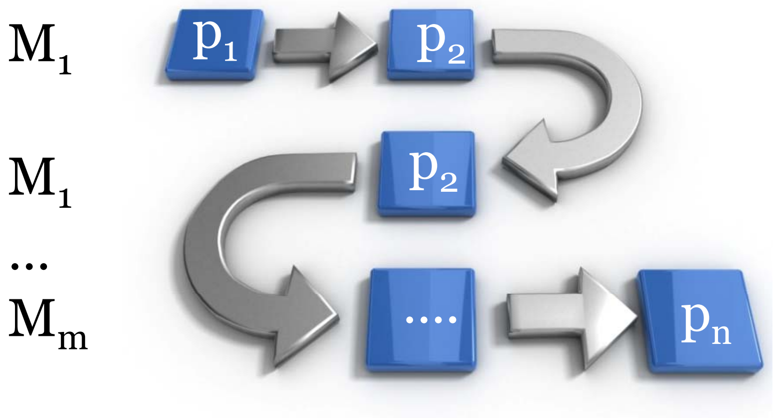
search continues…

[2]

SA for Scheduling problems

- **Example-2: Multiple Machines Minimum Makespan Problem**

Given processing times for **n jobs**, $p_1, p_2, \ldots, p_n$, and an integer m, the **Minimum Makespan Scheduling problem** is to find an assignment of the jobs to **m machines**, $M_1, M_2, \ldots, M_m$ so that the final completion time (the **makespan**) is minimized.



SA for Scheduling problems

- **Example-2: 3 Machines – 6 Jobs Problem**
  - ◇ 6 jobs with processing times:p_1,p_2,p_3,p_4,p_5,p_6= (2,3,4,6,2,2)
  - ◇ 3 machines m_1,m_2, m_3.
  - ◇ All machines should be assigned jobs.

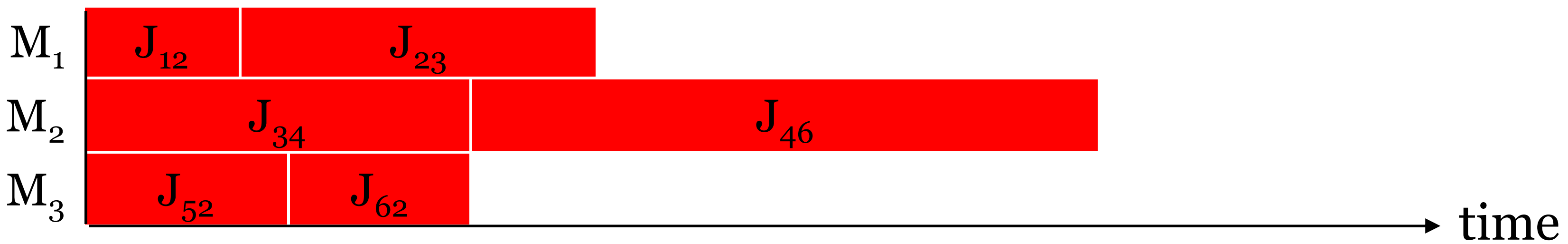
[2]

SA for Scheduling problems

- **Example-2: 3 Machines – 6 Jobs Problem**

Processing time: P=(2,3,4,6,2,2)

Consider the following schedule, we can code this schedule as **(1,1,2,2,3,3)** indicating machine assigned to the corresponding jobs.



Gantt-chart

Makespan=10

[2]

SA for Scheduling problems

- **Example-2: Generating Neighbors**
  - ◇ We can consider permutation of the machines numbers for the 6 jobs, e.g **(1,1,1,1,2,3)**, **(1,1,1,2,1,3)**,……
  - ◇ There are a bit less than 3^6 as we excluding unassigned machines.
  - ◇ Notice that swap operation from the initial solution doesn’t cover feasible solutions.

[2]

SA for Scheduling problems

- **Example-2: SA**
  - ◇ Select a permutation at random
  - ◇ Use geometric temperature reduction α=.9, T_0=0.9
  - ◇ Set no. of iterations at the same temperature=2

[2]

SA for Scheduling problems

- **Example-2: SA – Iteration 1**

  Initial Solution: **(1,1,2,2,3,3)** , **Makespan=10**

  Consider the following neighbor: **(1,1,1,1,2,3)** for p(2,3,4,6,2,2) will produce a **makespan of 15** which is worse than the initial solution.

  Calculate $p = e^{-\Delta f/T} = e^{-(15-10)/0.9} = 0.0004$

  Generate a random number r between (0,1), say it was r=0.2

  P<r so we reject the solution.

[2]

SA for Scheduling problems

- **Example-2: SA – Iteration 2**

  Current Solution: **(1,1,2,2,3,3)** , **Makespan=10**

  Generate another neighbor, say **(1,2,3,1,2,3)** with **makespan of 8**.

  This is an **improvement** of the current solution so we accept it.

  And so on.

  Optimal makespan for this problem is 7.

[2]

Outline

- Physical Annealing
- Simulated Annealing
- SA Cooling Schedule
- SA for TSP
- SA for PLP
- SA for Scheduling problems
- **SA for Function Optimization**
- Adaptive SA
- Cooperative SA
- Summary

SA for Function Optimization

Let us **maximize the continuous** function:

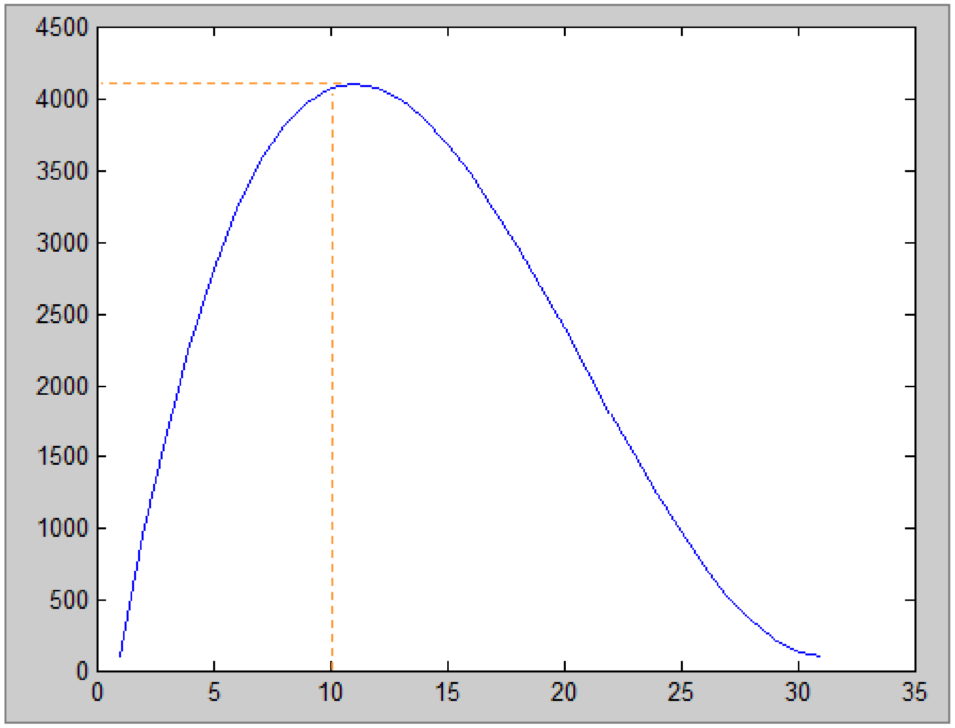
$$f(x) = x^3 - 60x^2 + 900x + 100.$$

- **Solution Representation:**

  A solution x is represented as a string of **5 bits**.

  The **global maximum** of this function is: 01010

  (x = 10, f (x) = 4100).



[4]

SA for Function Optimization

- **Neighboring Solution:**

  The neighborhood consists in **flipping randomly a bit**.

Solutions
00011
00111
00110
01110
01100
01000
01011
11011

[4]

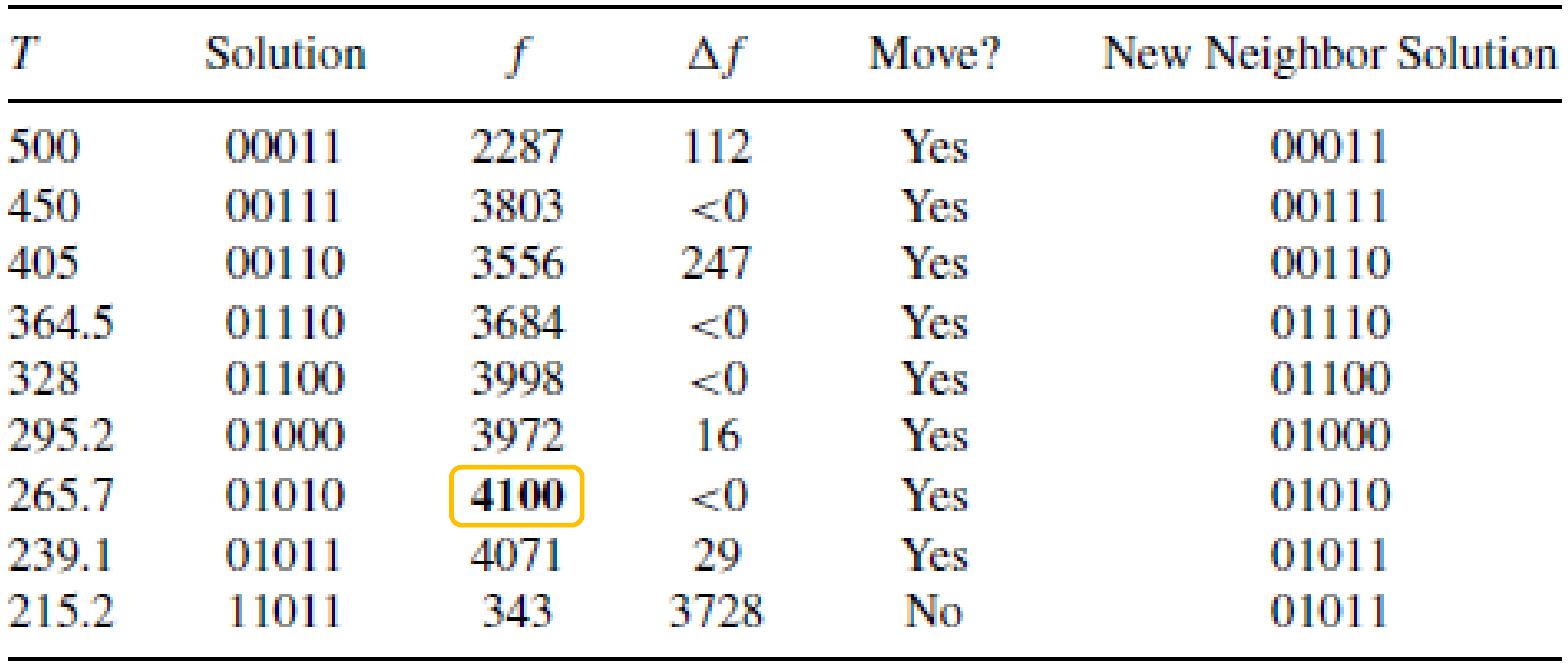
SA for Function Optimization

- **First Scenario:**

  Initial solution: 10011 (x = 19, f (x) = 2399)

  Initial temperature $T_o = 500$

First Scenario T = 500 and Initial Solution (10011)



T	Solution	f	Δf	Move?	New Neighbor Solution
500	00011	2287	112	Yes	00011
450	00111	3803	<0	Yes	00111
405	00110	3556	247	Yes	00110
364.5	01110	3684	<0	Yes	01110
328	01100	3998	<0	Yes	01100
295.2	01000	3972	16	Yes	01000
265.7	01010	$\boxed{\mathbf{4100}}$	<0	Yes	01010
239.1	01011	4071	29	Yes	01011
215.2	11011	343	3728	No	01011

[4]

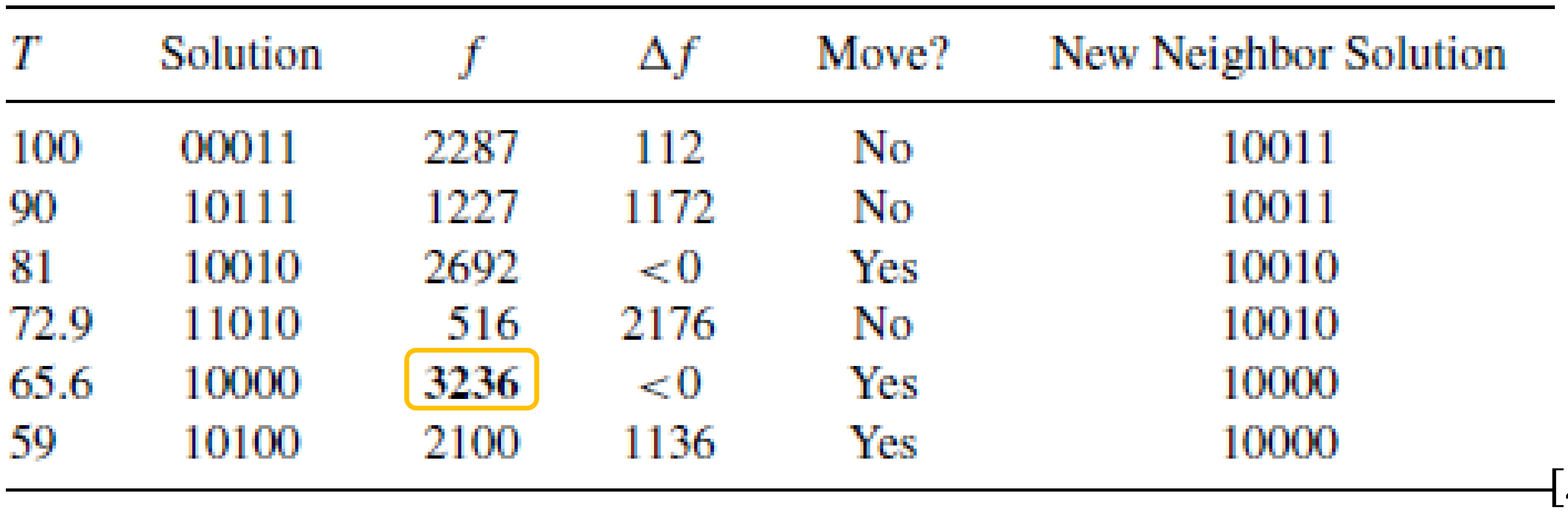
SA for Function Optimization

- **Second Scenario:**

  Initial solution: 10011

  Initial temperature T_o = 100. The initial temperature is not high enough and the algorithm gets stuck by local optima.

Second Scenario: T = 100 and Initial Solution (10011). When Temperature is not High Enough, Algorithm **Gets Stuck**

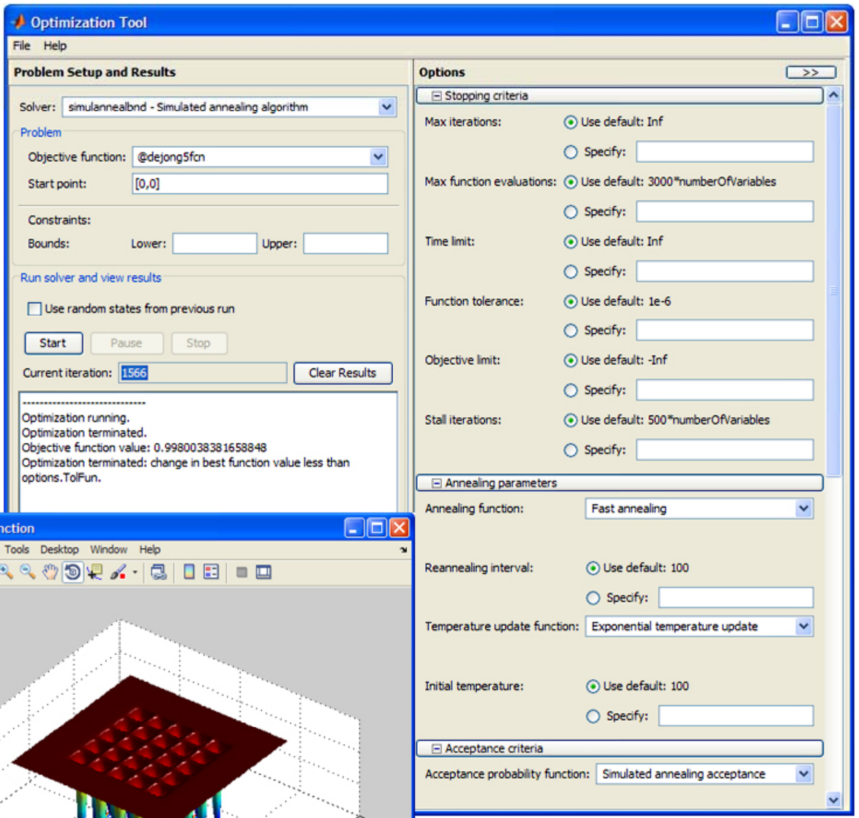


T	Solution	f	Δf	Move?	New Neighbor Solution
100	00011	2287	112	No	10011
90	10111	1227	1172	No	10011
81	10010	2692	<0	Yes	10010
72.9	11010	516	2176	No	10010
65.6	10000	$\boxed{3236}$	<0	Yes	10000
59	10100	2100	1136	Yes	10000

[4]

SA for Function Optimization

- **Simulated Annealing Solver**



Optimization Tool

File Help

Problem Setup and Results

Solver: simulannealbnd - Simulated annealing algorithm

Problem

Objective function: @dejong5fcn

Start point: [0,0]

Constraints:

Bounds: Lower: Upper:

Run solver and view results

☐ Use random states from previous run

Start Pause Stop

Current iteration: 1566 Clear Results

```
--------------------------
Optimization running.
Optimization terminated.
Objective function value: 0.9980038381658848
Optimization terminated: change in best function value less than
options.TolFun.
```

Options

Stopping criteria

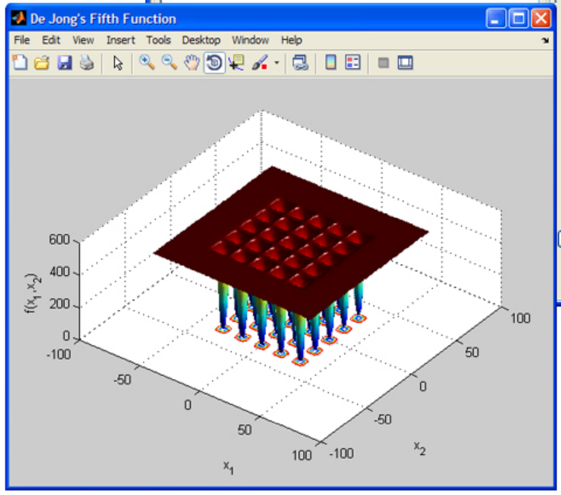
Max iterations:	[✓ Use default: Inf] Specify:
Max function evaluations:	[✓ Use default: 3000*numberOfVariables] Specify:
Time limit:	[✓ Use default: Inf] Specify:
Function tolerance:	[✓ Use default: 1e-6] Specify:
Objective limit:	[✓ Use default: -Inf] Specify:
Stall iterations:	[✓ Use default: 500*numberOfVariables] Specify:

Annealing parameters

Annealing function:	Fast annealing
Reannealing interval:	[✓ Use default: 100] Specify:
Temperature update function:	Exponential temperature update
Initial temperature:	[✓ Use default: 100] Specify:

Acceptance criteria

Acceptance probability function:	Simulated annealing acceptance

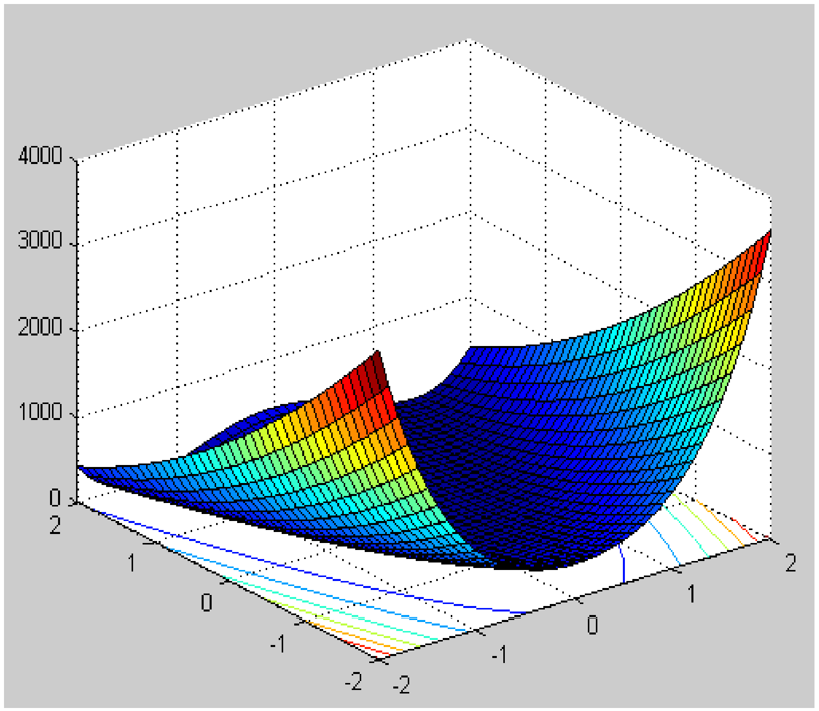


Resources page of the Course website

SA for Function Optimization

- **Assignment-2**

  Rosenbrock's banana function is a standard test function and quite tough for most conventional algorithms. This banana function is still relatively simple as it has a curved narrow valley. Other functions such as the egg crate function are strongly multimodal and highly nonlinear.



Rosenbrock's banana function

$$f(x,y)=(1-x)^2+100(y-x^2)^2$$

Egg crate function: $f(x,y)=x^2+y^2+25[\sin^2(x)+\sin^2(y)]$

Outline

- Physical Annealing
- Simulated Annealing
- SA Cooling Schedule
- SA for TSP
- SA for PLP
- SA for Scheduling problems
- SA for Function Optimization
- **Adaptive SA**
- Cooperative SA
- Summary

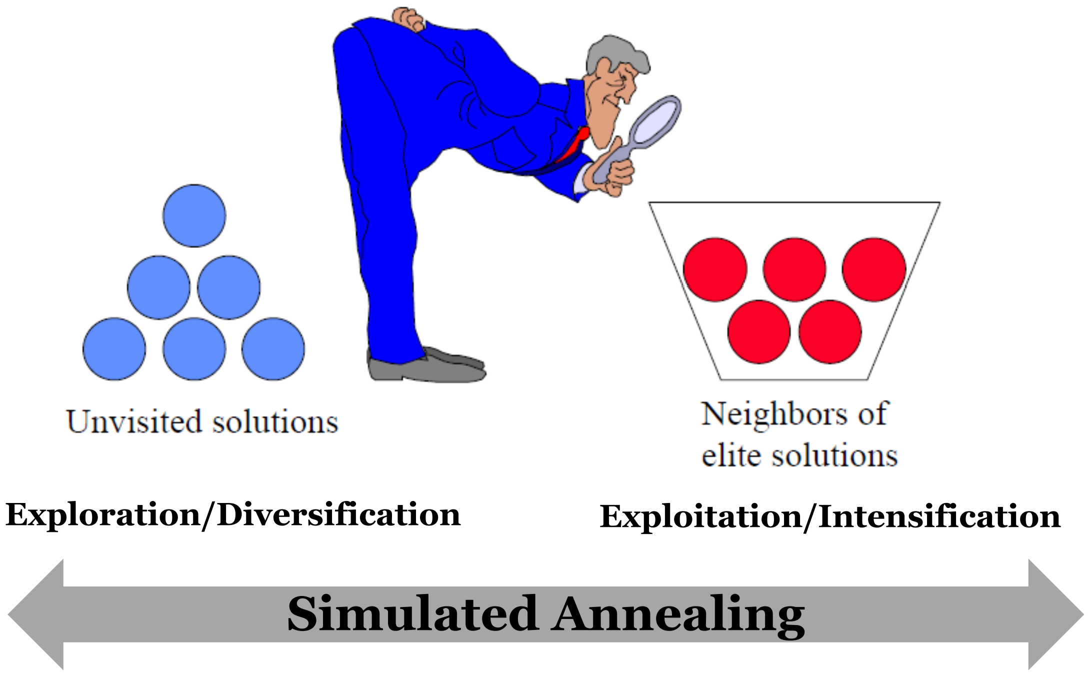
Adaptive SA

- **Adaptation in SA is in the parameters used by the algorithm**
  - ◇ The **initial temperature**, **the cooling schedule** and **no. of iterations per temperature** being the most critical.
  - ◇ Other components such the **cost function** and **method of generating neighborhood solutions** and **acceptance probability** affect the computational costs.

[2]

Adaptive SA

- **Initial Temperature**


Unvisited solutions
Neighbors of elite solutions
Exploration/Diversification
Exploitation/Intensification
Simulated Annealing

High Temperature

Low Temperature

Adaptive SA

- **Initial Temperature**
  - ◇ Finding the **right temperature** depends on the type of problem and sometime the instance of the problem.
  - ◇ One can try **different values** and see which leads to better solutions.
  - ◇ Some researchers suggested doing this adaptively using a search method such as **Genetic Algorithm** [5].

[2]

Adaptive SA

- **Cooling schedule**

Linear Cooling: $T = T_o - \beta i, \qquad \beta = \frac{(T_o - T_f)}{i_f}$

Geometric Cooling: $T(t) = T_o \alpha^i, \qquad i = 1,2,\ldots,i_f$

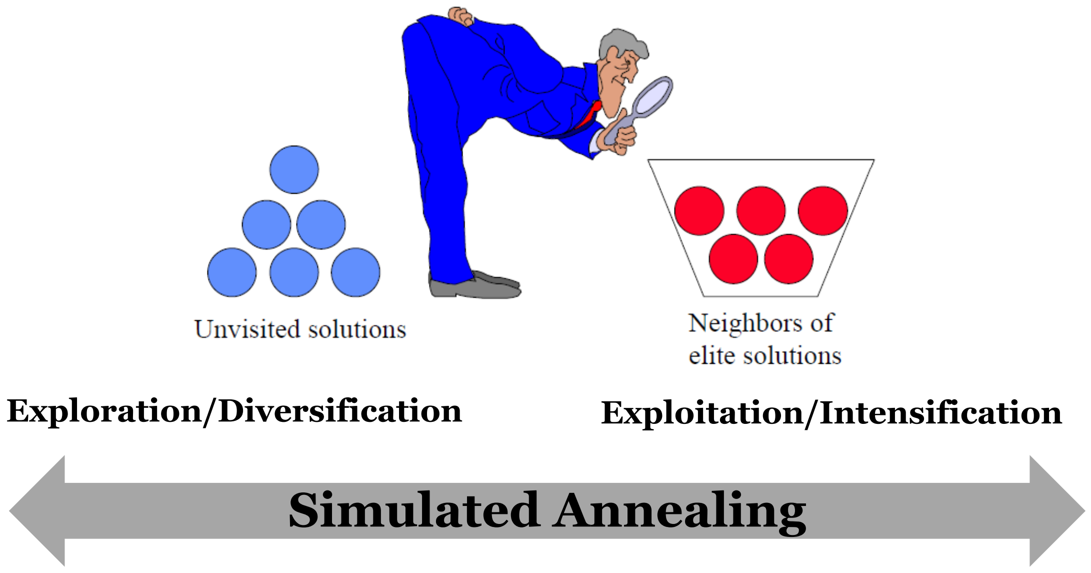
◇ **Different cooling schedules** can be used in different phases (most useful work is done in the middle of the schedule).

◇ **Reheating** may be tried if no progress is observed.

◇ **Cooling** may take place every time a move (or so many moves) is accepted.

Adaptive SA

- **Number of iterations per temperature**
  - ◇ Allowing a small number of iterations at high temperatures
  - ◇ Allowing a large number of iterations at low temperature to fully explore the local optimum.


Unvisited solutions
Neighbors of elite solutions
Exploration/Diversification
Exploitation/Intensification
Simulated Annealing

High Temperature
(allow small number of iterations)

Low Temperature
(allow large number of iterations)

Adaptive SA

- **Probability of acceptance**
  - ◇ The Boltzmann-based acceptance probability takes significant computational time (**~1/3 of the SA computations**).

    $$p = e^{-\Delta f/T} > r$$

  - ◇ The idea of using a **lookup table** where the exponential calculation are done once for a range of values for change in *f* and T has been suggested.
  - ◇ Other non-exponential probability formulas have been suggested, for example Johnson et.al.[8] suggested

    $$\mathbf{P(\Delta \mathit{f}) = 1 - \Delta \mathit{f}/T}$$

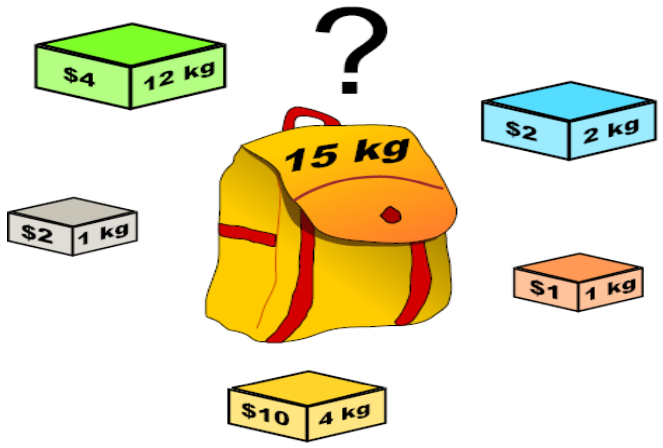
[2]

Adaptive SA

- **Cost Function**
  - ◇ Avoid cost functions that return the **same value** for many states (e.g number of edges included in a route in TSP). This type of functions doesn't lead the search.
  - ◇ Many problems have some constraints that can be represented in the cost function using penalty terms.

Example: f_1 = the difference (in absolute value) between the **total weight** of the selected items and the **knapsack capacity** + the **penalty** (50) in case the capacity is overloaded (and this criterion is to be minimized).

$$f_1 = \left|\sum_{i=1}^{m} w_i - C\right| + 50\Big|_{if \sum_{i=1}^{m} w_i \succ C}$$ m is number of selected items



Knapsack Problem

Adaptive SA

- **Cost Function**
  - ◇ One way to make the algorithm more adaptive is to have a **dynamically changing weighting of the penalty** terms. So in the initial phase the constraints can be relaxed more than in advanced phases.
  - ◇ To reduce the computational time involved, update functions have been suggested.

[2]

Adaptive SA

Handling constraints

Generate only feasible neighbourhood

- ◇ Elegant
- ◇ Inflexible
- ◇ Most suitable for hard constraints

Penalty function methods

- ◇ Flexible
- ◇ Huge search space includes infeasible solutions.
- ◇ Hard to define penalty function
- ◇ Most suitable for soft constraints

[7]

Adaptive SA

- **Totally adaptive SA**
  - ◇ There have been some attempts at making the selection and control of SA totally adaptive.
  - ◇ One such attempt proposed by Ingber in 1993 [6]. ASA **automatically adjusts the algorithm parameters that control the temperature schedule**.
  - ◇ It requires the user to only specify the **cooling rate** α. All the other parameter are automatically determined.

[2]

Adaptive SA

- **Totally adaptive SA**
  - ◇ The method uses **linear random combination of previously accepted steps and parameters** to estimate new steps and parameters.
  - ◇ More information including source code available at:

    http://www.ingber.com/ Last accessed: May 31, 2014.

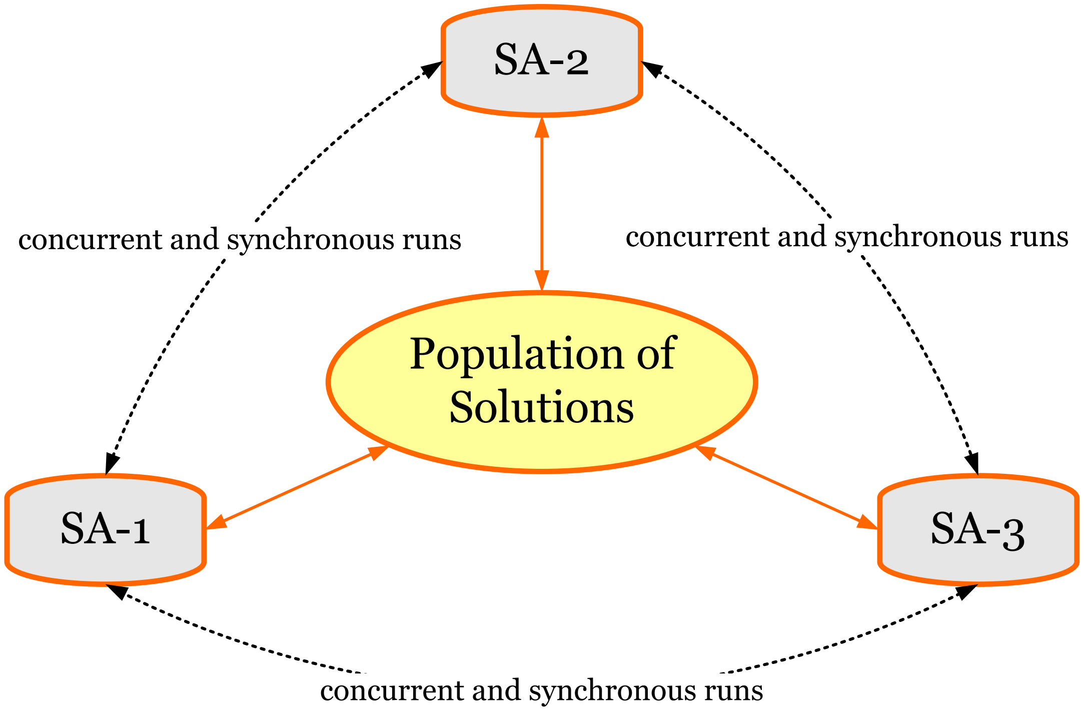
[2]

Outline

- Physical Annealing
- Simulated Annealing
- SA Cooling Schedule
- SA for TSP
- SA for PLP
- SA for Scheduling problems
- SA for Function Optimization
- Adaptive SA
- **Cooperative SA**
- Summary

Cooperative SA

- **Cooperative SA (COSA)** was proposed by Wendt et al. in 1997 [8],
- COSA implements **concurrent and synchronous runs of multiple SA processes**,
- The concurrent processes are coupled through the cooperative transitions,
- The cooperative transition replaces the **uniform distribution used to select the neighbours**.


SA-2
concurrent and synchronous runs
concurrent and synchronous runs
Population of Solutions
SA-1
SA-3
concurrent and synchronous runs

[2]

Cooperative SA

- **Cooperative SA (COSA)**
  - ◇ The algorithm manipulates multiple solutions (a population of solutions) at once,
  - ◇ Assuming we have five solutions, we start by a randomly chosen population: $Pop_0 = [S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5]$
  - ◇ The new population is **iteratively produced**,
  - ◇ Any new solution is **cooperatively produced** by:
    - The **previous value of that solution**,
    - The previous value of a **randomly selected solution**,

[2]

Cooperative SA

- **Cooperative SA (COSA)**

$$\begin{tikzcd}
 & & & & \text{A randomly selected solution} \arrow[d] & \\
Pop_{k-1} = \Big[& S_1 \arrow[d] & S_2 & S_3 & S_4 \arrow[dlll] & S_5 \Big] \\
Pop_k = \Big[& S_{1new} \Big] & & & &
\end{tikzcd}$$

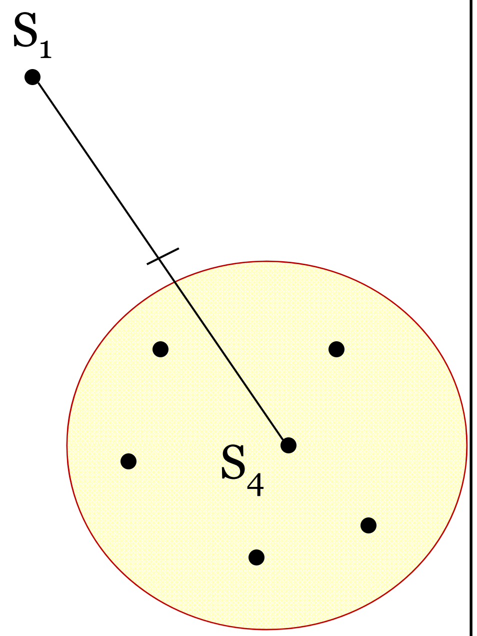
$$\begin{tikzcd}
Pop_{k-1} = \Big[& S_1 & S_2 \arrow[dr] & S_3 \arrow[d] & S_4 & S_5 \Big] \\
Pop_k = \Big[& S_{1new} & & S_{2new} \Big] & &
\end{tikzcd}$$

[2]

Cooperative SA

- **Cooperative SA (COSA)**
  - ◇ **How do we find S_{1new} from S_1 and S_4?**
  - ◇ We find the neighbours of S_1 that are closer to S_4 than S_1 itself (using some distance measure),
  - ◇ These neighbours constitute what is known as the ***CLOSER*** set.
  - ◇ The ***CLOSER*** set is defined as:

$$CLOSER = \{s_k \in N(S_1) \mid d(s_k, S_4) < d(s_k, S_1)\}$$



CLOSER set

[2]

Cooperative SA

- **Cooperative SA (COSA)**
  - ◇ **If** the $\boldsymbol{CLOSER}$ set is **not empty**, the new solution is **randomly selected from it**,
  - ◇ **Otherwise,** the new solution is **randomly selected from the neighbourhood of $\mathbf{S_1}$**.

[2]

Cooperative SA

- **Cooperative SA (COSA)**
  - ◇ The **temperature** is updated based on the **difference of the mean fitness of the new and old populations**,

$$\Delta f = f\left(Pop_k\right) - f\left(Pop_{k-1}\right)$$

$$T = \begin{cases} T & , if \;\; \Delta f < 0 \\ \alpha T & , otherwise \end{cases}$$

[2]

Cooperative SA

- **Cooperative SA (COSA)**

```
Set population size $Pop$,
Initialize $Pop_0$,
For a determined number of transitions K
    For $i$ = 1 to $Pop$
        Select a random cooperator $S_j \in Pop_{k-1}$,
        Generate $S_{inew}$ = cotrans($S_i$ ,$S_j$),
        Accept the new solution if it's better,
        Otherwise, probabilistically determine if a move to be taken to
          the new solution and update the solution accordingly.
    end
    Update Temperature,
end
```

[2]

Cooperative SA

- **Cooperative SA (COSA)**
  - ◇ COSA inherits the idea of **population and information exchange form Genetic Algorithms**.
  - ◇ It uses **cooperative transitions** instead of GA crossover.
  - ◇ In [8] they showed that using **increased cooperation** had **positive effect on getting better solutions** (TSP, Job Shop Scheduling, select of comm. protocols).

[2]

Outline

- Physical Annealing
- Simulated Annealing
- SA Cooling Schedule
- SA for TSP
- SA for PLP
- SA for Scheduling problems
- SA for Function Optimization
- Adaptive SA
- Cooperative SA
- **Summary**

Summary

- **Simulated annealing (SA)** is a random search technique for **global optimization problems**, and it mimics the annealing process in material processing when a metal cools and freezes into a crystalline state with the minimum energy and larger crystal size so as to reduce the defects in metallic structures. The annealing process involves the careful control of temperature and cooling rate, often called annealing schedule.
- The basic idea of the SA algorithm is to use random search in terms of a **Markov chain**, which not only accepts changes that improve the objective function, but also keeps some changes that are not ideal. In a **minimization problem**, for example, any better moves or changes that decrease the value of the objective function *f* will be accepted; however, some changes that increase *f* will also be accepted with a certain **probability**.

Summary

- **To fine tune SA:**
  - ◇ Experiment for best parameter settings (replications are needed)
  - ◇ Monitor evolution of the SA algorithm, graphically if possible, for different parameter settings
  - ◇ Monitor temperature, cost, ratio of accepted moves to rejected moves and other statistics (plot these against the iteration number).

References

1. Johnson's Algorithm For Scheduling: http://www.ielm.ust.hk/dfaculty/ajay/courses/ieem513/GT/johnson.html. Last accessed June 1, 2014.
2. M. Kamel. ECE457A: Cooperative and Adaptive Algorithms. University of Waterloo, 2010-2011.
3. The Scheduling Problem: http://riot.ieor.berkeley.edu/riot/Applications/Scheduling/objectives.html. Last accessed June 1, 2014.
4. El-Ghazali Talbi. Metaheuristics: From Design to Implementation. John Wiley & Sons, Inc., 2009.
5. Miki, Hiroyasu, Wako and Yoshida Adaptive Temperature Schedule Determined by Genetic Algorithm for Parallel Simulated Annealing, CEC 2003.

References

6. L. Ingber. “Adaptive Simulated Annealing: Lessons Learned”. invited paper to a special issue of the Polish Journal Control and Cybernetics on “Simulated Annealing Applied to Combinatorial Optimization.”, Vol.25, No.1, pp.33-54 1996.
7. Mahmut Ali GÖKÇE. ISE 410: Heuristics in Optimization. Izmir Ekonomi Üniversitesi.
8. O. Wendt and W. Konig.”Cooperative Simulated Annealing: How much Cooperation is Enough?”. Research Report 97-19, Institute of Information Systems, GoetheUniversity Frankfurt, 1997.
9. Günther Zäpfel, Roland Braune and Michael Bögl. *Metaheuristic Search Concepts to Production and Logistics A Tutorial with Applications*. Springer.