Optimization Techniques for Design Space Exploration

Zebo Peng
Embedded Systems Laboratory (ESLAB)
Linköping University

Outline

- Optimization problems in ERT system design
- Heuristic techniques
- Simulated annealing
- Tabu search
Design Space of ERT Systems

- Very large due to many solution parameters:
  - architectures and components
  - hardware/software partitioning
  - mapping and scheduling
  - operating systems and global control
  - communication synthesis

S3 Hardware Software
Embedded memory
Microprocessor
ASIC
Analog circuit
Sensor
High-speed electronics

Design Space of ERT Systems

- Very bumpy due to the interdependence between the different parameters and the many design constraints (time, power, partial structure, ...).

Many embedded RT systems have a very complex design space.
Design Space Exploration

What are needed in order to explore the complex design space to find a good solution:

- Exploration in the higher level of abstractions.
- Development of high-level analysis and estimation techniques.
- Employment of very fast exploration algorithms.
- Memory-less algorithms.
- Each solution needs a huge data structure to store, so we can’t afford to keep track of all visited solutions.

The Optimization Problem

The majority of design space exploration tasks can be viewed as optimization problems:

To find

- the architecture (type and number of processors, memory modules, and communication mechanism, as well as their interconnections),
- the mapping of functionality onto the architecture components, and
- the schedules of basic functions and communications,

such that a cost function (in terms of implementation cost, performance, power, etc.) is minimized and a set of constraints is satisfied.
Mathematical Optimization

- The design optimization problems can be formulated as to

  Minimize \( f(x) \)

  Subject to \( g_i(x) \geq b_i; \; i = 1, 2, ..., m \)

  where

  - \( x \) is a vector of decision variables \( (x \geq 0) \);
  - \( f \) is the cost (objective) function;
  - \( g_i \)'s are a set of constraints.

- If \( f \) and \( g_i \) are linear functions, we have a linear programming (LP) problem.
- LP can be solved by the simplex algorithm, which is an exact method.
  - It will always identify the optimal solution if it exists.

Combinational Optimization (CO)

- In many design problems, the decision variables are restricted to integer values.
  - The solution is a set, or a sequence, of integers or other discrete objects.
  - We have an Integer Linear Programming (ILP) problem, which turns out to be more difficult to solve than the LP problems.

- Ex. System partitioning can be formulated as:
  - Given a graph with costs on its edges, partition the nodes into \( k \) subsets no larger than a given maximum size, to minimize the total cost of the cut edges.
  - A feasible solution (with \( n \) nodes) is represented as
    \[ x_i = j; \quad j \in \{1, 2, ..., k\}, \quad i = 1, 2, ..., n. \]
- They are called combinatorial optimization problems.
Features of CO Problems

- Most CO problems, e.g., system partitioning with constraints, for ERT system designs are NP-complete.
- The time needed to solve an NP-complete problem grows exponentially with respect to the problem size \( n \).
- Approaches for solving such problems are usually based on implicit enumeration of the feasible solutions.
- For example, to enumerate all feasible solutions for a scheduling problem (all possible permutation), we have:
  - 20 tasks in 1 hour (assumption);
  - 21 tasks in 20 hour;
  - 22 tasks in 17.5 days;
  - ...
  - 25 tasks in 6 centuries.

Outline

- Optimization problems in ERT system design
  - Heuristic techniques
  - Simulated annealing
  - Tabu search
Heuristics

- A heuristic seeks near-optimal solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality.

- Motivations:
  - Many exact algorithms require a huge computation effort.
  - The decision variables have complicated interdependencies.
  - We have often nonlinear cost functions and constraints, even no mathematical functions.
    - Ex. The cost function $f$ can, for example, be defined by a computer program (e.g., for power estimation).
  - Approximation of the model for optimization.
    - A near optimal solution is usually good enough and could be even better than the theoretical optimum.

Heuristic Approaches to CO

<table>
<thead>
<tr>
<th>Problem specific</th>
<th>Generic methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructive</td>
<td></td>
</tr>
<tr>
<td>List scheduling</td>
<td>Divide and conquer</td>
</tr>
<tr>
<td>Left-edge algorithm</td>
<td>Branch and bound</td>
</tr>
<tr>
<td>Clustering</td>
<td></td>
</tr>
<tr>
<td>Transformational</td>
<td></td>
</tr>
<tr>
<td>Kernighan-Lin algorithm</td>
<td>Neighborhood search</td>
</tr>
<tr>
<td>for graph partitioning</td>
<td>Simulated annealing</td>
</tr>
<tr>
<td></td>
<td>Tabu search</td>
</tr>
<tr>
<td></td>
<td>Genetic algorithms</td>
</tr>
</tbody>
</table>
Clustering for System Partitioning

- Each node initially belongs to its own cluster, and clusters are gradually merged until the desired partitioning is found.
- Merge operations are selected based on local information (closeness metrics), rather than global view of the system.

Branch-and-Bound

- Traverse an implicit tree to find the best leaf (solution).

4-City TSP

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & 0 & 3 & 6 & 41 \\
1 & 0 & 40 & 5 \\
2 & 0 & 4 \\
3 & 0 \\
\end{array}
\]

Total cost of this solution = 88
### Branch-and-Bound Ex

- Low-bound on the cost function.
- Search strategy

### Neighborhood Search Method

- **Step 1** (Initialization)
  
  (A) Select a starting solution $x^\text{now} \in X$.
  
  (B) $x^\text{best} = x^\text{now}$, $\text{best_cost} = c(x^\text{best})$.

- **Step 2** (Choice and termination)
  
  Choose a solution $x^\text{next} \in N(x^\text{now})$, in the neighborhood of $x^\text{now}$.
  
  If no solution can be selected, or the terminating criteria apply, then the algorithm terminates.

- **Step 3** (Update)
  
  Re-set $x^\text{now} = x^\text{next}$.
  
  If $c(x^\text{now}) < \text{best_cost}$, perform Step 1(B).
  
  Goto Step 2.
Neighborhood Search Method

- Very attractive for many CO problems as they have a natural neighborhood structure, which can be easily defined and evaluated.
  - Ex. Graph partitioning: swapping two nodes.

The Descent Method

- Step 1 (Initialization)
- Step 2 (Choice and termination)
  Choose $x_{\text{next}} \in N(x_{\text{now}})$ such that $c(x_{\text{next}}) < c(x_{\text{now}})$, and terminate if no such $x_{\text{next}}$ can be found.
- Step 3 (Update)

The descent method can easily be stuck at a local optimum.
Dealing with Local Optimality

- Enlarge the neighborhood.
- Start with different initial solutions.
- To allow “uphill moves”:
  - Simulated annealing
  - Tabu search

Outline

- Optimization problems in ERT system design
- Heuristic techniques
  - Simulated annealing
- Tabu search
Annealing

● Annealing is the slow cooling of metallic material after heating:
  - A solid material is heated past its melting point.
  - Cooling is then slowly done.
  - When cooling stops, the material settles usually into a low energy state (e.g., a stable structure).
  - In this way, the properties of the materials are improved.

Annealing

● The annealing process can be viewed intuitively as:
  - At high temperature, the atoms are randomly oriented due to their high energy caused by heat.
  - When the temperature reduces, the atoms tend to line up with their neighbors, but different regions may have different directions.
  - If cooling is done slowly, the final frozen state will have a near-minimal energy state.
Simulated Annealing

Annealing can be simulated using computer simulation:

- Generate a random perturbation of the atom orientations and calculates the resulting energy change.
- If the energy has decreased, the system moves to this new state.
- If energy has increased, the new state is accepted according to the laws of thermodynamics:
  
  At temperature $t$, the probability of an increase in energy of magnitude $\Delta E$ is given by
  
  $$p(\Delta E) = \frac{1}{e^{\left(\frac{\Delta E}{k \cdot t}\right)}}$$

  where $k$ is called the Boltzmann’s constant.

Probability of Accepting Higher-Energy States

$$p(\Delta E) = \frac{1}{e^{\left(\frac{\Delta E}{k \cdot t}\right)}}$$

At high temperatures, the probability of accepting higher-energy states is high, but as the temperature decreases, the probability decreases exponentially, making it less likely to accept higher-energy states.
Simulated Annealing for CO

- The SA algorithm could be applied to combinatorial optimization:

<table>
<thead>
<tr>
<th>Thermodynamic simulation</th>
<th>Combinatorial optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>System states</td>
<td>Feasible solutions</td>
</tr>
<tr>
<td>Energy</td>
<td>Cost</td>
</tr>
<tr>
<td>Change of state</td>
<td>Moving to a neighboring solution</td>
</tr>
<tr>
<td>Temperature</td>
<td>“Control parameter”</td>
</tr>
<tr>
<td>Frozen state</td>
<td>Final solution</td>
</tr>
</tbody>
</table>

The SA Algorithm

Select an initial solution $x^{\text{now}} \in X$;
Select an initial temperature $t > 0$;
Select a temperature reduction function $\alpha$;

Repeat

Repeat

Randomly select $x^{\text{next}} \in N(x^{\text{now}})$;
\[ \delta = \text{cost}(x^{\text{next}}) - \text{cost}(x^{\text{now}}); \]
If $\delta < 0$ then $x^{\text{now}} = x^{\text{next}}$
else generate randomly $p$ in the range $(0, 1)$ uniformly;
\[ \text{If } p < \exp(-\delta/t) \text{ then } x^{\text{now}} = x^{\text{next}}; \]
Until iteration_count = nrep;
Set $t = \alpha(t)$;
Until stopping condition = true.
Return $x^{\text{now}}$ as the approximation to the optimal solution.
The SA Algorithm

- Simulated annealing is really an optimization strategy rather than an algorithm — a meta heuristic.

- To have a working algorithm, the following must be done:
  - Selection of generic parameters.
  - Problem-specific decisions must also be made.

A HW/SW Partitioning Example

Cost Function value optimum at iteration 1006
The Cooling Schedule I

- **Initial temperature (IT):**
  - IT must be "hot" enough to allow an almost free exchange of neighborhood solutions, if the final solution is to be independent of the starting one.
  - Simulating the heating process:
    - A system can be first heated rapidly until the proportion of accepted moves to rejected moves reaches a given value; e.g., when 80% of moves leading to higher costs will be accepted.
    - Cooling will then start.

The Cooling Schedule II

- **Temperature reduction scheme:**
  - A large number of iterations at few temperatures or a small number of iterations at many temperatures.
  - Typically $\alpha(t) = a \times t$, where $a < 1$; $a$ should be large, usually between 0.8 and 0.99.
  - For better results, the reduction rate should be slower in middle temperature ranges.

- **Stopping conditions:**
  - Zero temperature - the *theoretical* requirement.
  - A number of iterations or temperatures has passed without any acceptance of moves.
  - A given total number of iterations have been completed (or a fixed amount of execution time).
Problem-Specific Decisions

- The neighborhood structure should be defined such that:
  - All solutions should be reachable from each other.
  - Easy to generate randomly a neighboring feasible solution.
  - Penalty for infeasible solutions, if the solution space is strongly constrained.
  - The cost difference between $s$ and $s_0$ should be able to be efficiently calculated.
  - The size of the neighborhood should be kept reasonably small.

- Many decision parameters must be fine-tuned based on experimentation on typical data.

Outline

- Optimization problems in ERT system design
  - Heuristic techniques
    - Simulated annealing
      - Tabu search
Introduction to Tabu Search

- Tabu search (TS) is a neighborhood search method which employs "intelligent" search and flexible memories to avoid being trapped at local optimum.
  - To de-emphasize randomization.
  - Moves are selected intelligently in each iteration (the best admissible move is selected).
  - Use tabus to restrict the search space and avoid cyclic behavior (dead loop).
  - The classification of tabus is based on the history of the search.
- Taking advantage of history.
- It emulates human problem solving process.

An Illustrative Example

- A set of tests is to be scheduled to check the correctness and other features of a given system.
- To find an ordering of the tests that maximizes the test performance (fault coverage, time, and power):
- A feasible solution can be simply represented by a permutation of the given set of tests.
- A neighborhood move can be defined by swapping two tests.
- The best move will be selected in each step.
- To avoid repeating or reversing swaps done recently, we classify as tabu all most recent swaps.
Let a paired test tabu to be valid only for three iterations (tabu tenure):

- When a tabu move would result in a solution better than any visited so far, its tabu classification may be overridden (an aspiration criterion).
A TS Process

Current solution

Performance = 10

Performance = 16

Current solution

Performance = 18

Top 5 candidates

Swap Value

5, 4  6
7, 4  4
3, 6  2
2, 3  0
4, 1  -1

Swap Value

3, 1  2
2, 3  1
3, 6  -1
7, 1  -2
6, 1  -4

Swap Value

1, 3  -2
2, 4  -4
7, 6  -6
4, 5  -7
5, 3  -9
A TS Process

Current solution

Performance = 14

Uphill moves are allowed!

Performance = 20

Aspiration criterion applies!

Best so far!

Current solution

Performance = 18

Tabu structure

Top 5 candidates

Swap   Value
1, 3   -2
2, 4   -4
7, 6   -6
4, 5   -7
5, 3   -9

Swap   Value
4, 5   6
5, 3   -2
7, 1   0
1, 3   -3
2, 6   -6

Best so far!

Current solution

Performance = 16

Tabu structure

Top 5 candidates

Swap   Value
4, 5   6
5, 3   2
7, 1   0
1, 3   -3
2, 6   -6

Swap   Value
7, 1   0
4, 3   -3
6, 3   -5
5, 4   -6
2, 6   -8
The paired test tabu makes use of recency-based memory (short-term memory).

It should be complemented by frequency-based memory (long-term memory) to diversify the search into new regions.

Diversification is restricted to operate only on particular occasions.

For example, we can select those occasions where no admissible improving moves exist.

**Tabu Memory Structure**

**Iteration 26**

**Current solution**

```
1 3 6 2 7 5 4
```

**Performance = 12**

**Top 5 candidates**

<table>
<thead>
<tr>
<th>Swap</th>
<th>Value</th>
<th>Penalized Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2,4</td>
<td>-1</td>
<td>-6</td>
</tr>
<tr>
<td>3,7</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>1,6</td>
<td>-5</td>
<td>-5</td>
</tr>
<tr>
<td>6,5</td>
<td>-4</td>
<td>-6</td>
</tr>
</tbody>
</table>

P.V. = Value – Frequency_count

**Recency-based**

**Frequency-based**
Effects of Random Diversifications

The Basic TS Algorithm

Step 1  (Initialization)
(A) Select a starting solution \( x^{\text{now}} \in X \).
(B) \( x^{\text{best}} = x^{\text{now}}, \ best_{\text{cost}} = c(x^{\text{best}}) \).
(C) Set the history record \( H \) empty.

Step 2  (Choice and termination)
Determine \( \text{Candidate}_N(x^{\text{now}}) \) as a subset of \( N(H, x^{\text{now}}) \).
Select \( x^{\text{next}} \) from \( \text{Candidate}_N(x^{\text{now}}) \) to minimize \( c(H, x) \).
Terminate by a chosen iteration cut-off rule.

Step 3  (Update)
Re-set \( x^{\text{now}} = x^{\text{next}} \).
If \( c(x^{\text{now}}) < best_{\text{cost}} \), perform Step 1(B).
Update the history record \( H \).
Return to Step 2.
Tabu and Tabu Status

- A tabu is usually specified by some attributes of the moves.
- Typically when a move is performed that contains an attribute, a record is maintained for its reverse attribute. => Preventing reversals or repetitions!
- A tabu restriction is typically activated only under certain condition:
  - Recency-based restriction: its attributes occurred within a limited number of iterations prior to the present iteration;
  - Frequency-based restriction: occurred with a certain frequency over a longer span of iterations.
- The tabu restrictions and tenure should be selected to achieve cycle prevention and induce vigor into the search.

Tabu Tenure Decision

- The tabu tenure, $t$, must be carefully selected:
  - For highly restrictive tabus, $t$ should be smaller than for lesser restrictive tabus.
  - It should be long enough to prevent cycling, but short enough to avoid driving the search away from the global optimum.
- $t$ can be determined using static rules or dynamic rules:
- Static rule choose a value for $t$ that remains fixed:
  - $t = \text{constant}$ (typically between 7 and 20).
  - $t = f(n)$, where $n$ is the problem size (typically between $0.5 \, n^{1/2}$ and $2 \, n^{1/2}$).
- Experimentation must be carried out to choose the best tenure!
Aspiration Criteria (AC)

- Used to determine when tabu restrictions can be overridden.
- They contribute significantly to the quality of the algorithm.

Examples of Aspiration Criteria:

- Aspiration by Default: If all available moves are classified as tabu, and are not rendered admissible by some other AC, then a "least tabu" move is selected.
  - This is always implemented, e.g., by selecting the tabu with the shortest time to become inactive.

- Aspiration by Objective:
  - \( c(x_{\text{trial}}) < \text{best\_cost}. \)
  - Subdivide the search space into regions \( R \in \mathbb{R} \), and let \( \text{best\_cost}(R) \) denote the minimum \( c(x) \) for \( x \) found in \( R \). If \( c(x_{\text{trial}}) < \text{best\_cost}(R) \), a move aspiration is satisfied.

Stopping Conditions

TS does not converge naturally.

- A fixed number of iterations has elapsed in total.
- A fixed number of iterations has elapsed since the last best solution was found.
- A given amount of CPU time has been used.
TS vs. SA

- Neighborhood space exploration:
  - TS emphasizes complete neighborhood evaluation to identify moves of high quality.
  - SA samples the neighborhood solutions randomly.

- Move evaluation:
  - TS evaluates the relative attractiveness of moves in relation not only to objective function change, but also to factors of influence.
  - SA evaluates moves only in terms of their objective function change.

TS vs. SA (Cont’d)

- Search guidance:
  - TS uses multiple thresholds, reflected in the tabu tenures and aspiration criteria, which varies also non-monotonically.
  - SA is based on a single threshold implicit in the temperature parameter that only changes monotonically.

- Use of memory:
  - SA is memoryless.
  - TS makes heavily and intelligently use of both short-term and long-term memory.
  - TS can also use the mid-term memory for intensification.
Summary

- Design space exploration is basically an optimization problem.
- Due to the complexity of the optimization problem, heuristic algorithms are widely used.
- Many general heuristics are based on neighborhood search principles.
- SA is applicable to almost any combinatorial optimization problem, and very simple to implement.
- TS has a natural rationale: it emulates intelligent uses of memory.
- When properly implemented, TS often outperforms SA (the execution time is often one order of magnitude smaller).