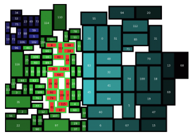
Embedded System Design and Synthesis

Robert Dick

http://robertdick.org/esds-two-week
http://ziyang.eecs.northwestern.edu/~dickrp/esds-two-week
Department of Electrical Engineering and Computer Science
Northwestern University

Office at Tsinghua University: 9–310 East Main Building

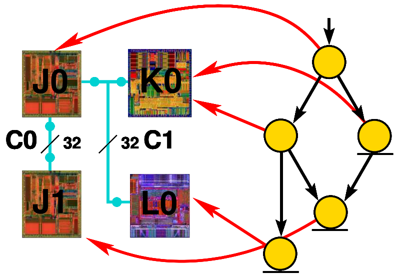




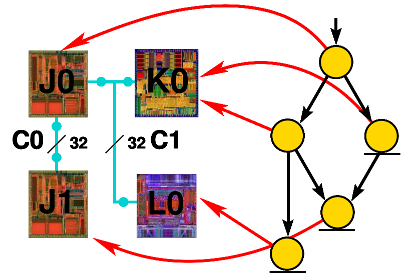
Synthesis motivation

- Embedded systems are found everywhere: cars, houses, games, phones, hospitals, etc.
- Designers need tools to deal with increasing complexity, increase product quality, and guarantee correct operation.
- Software or hardware errors are not acceptable. Anti-lock brake systems aren't allowed to crash.
- Embedded systems should not require bug fixes or upgrades.
- Price competition can be intense.
- Power consumption should be low.

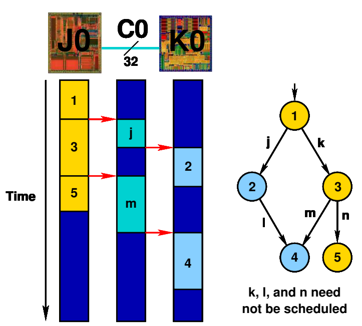
Allocation



Assignment

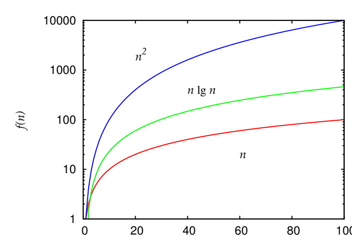


Scheduling



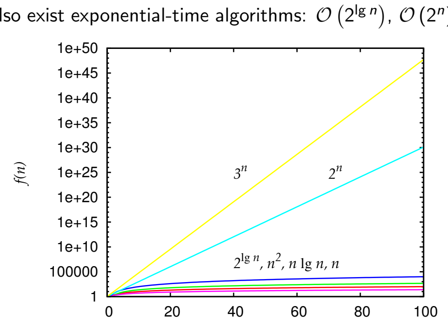
Polynomial time complexities

- Recall that sorting may be done in $\mathcal{O}(n \lg n)$ time
- DFS $\in \mathcal{O}(|V| + |E|)$, BFS $\in \mathcal{O}(|V|)$
- Topological sort $\in \mathcal{O}(|V| + |E|)$



Exponential time complexities

There also exist exponential-time algorithms: $\mathcal{O}(2^{\lg n})$, $\mathcal{O}(2^n)$, $\mathcal{O}(3^n)$



Implications of exponential time complexity

For $t(n) = 2^n$ seconds

$t(1) = 2$ seconds
$t(10) = 17$ minutes
$t(20) = 12$ days
$t(50) = 35,702,052$ years
$t(100) = 40,196,936,841,331,500,000,000$ years

$\mathcal{NP}$-complete problems

- Digital design and synthesis is full of NP-complete problems
- Graph coloring
- Allocation/assignment
- Scheduling
- Graph partitioning
- Satisfiability (and 3SAT)
- *Covering*
- ... and many more

Conjecture on hardness of problems

- There is a class of problems, $\mathcal{NP}$-complete, for which nobody has found polynomial time solutions
- It is possible to convert between these problems in polynomial time
- Thus, if it is possible to solve any problem in $\mathcal{NP}$-complete in polynomial time, all can be solved in polynomial time
- $\mathcal{P} \subseteq \mathcal{NP}$
- Unproven conjecture: $\mathcal{P} \neq \mathcal{NP}$

$\mathcal{NP}$

- What is $\mathcal{NP}$? Nondeterministic polynomial time.
- A computer that can simultaneously follow multiple paths in a solution space exploration tree is nondeterministic. Such a computer can solve $\mathcal{NP}$ problems in polynomial time.
- Nobody has been able to prove either

$$\mathcal{P} \neq \mathcal{NP}$$

or

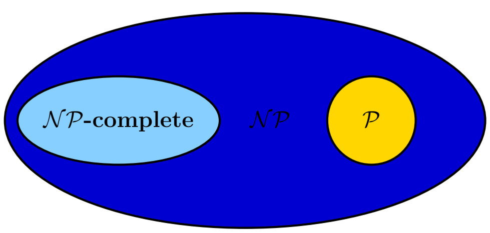
$$\mathcal{P} = \mathcal{NP}$$

$\mathcal{NP}$-completeness

If we define $\mathcal{NP}$-complete to be a set of problems in $\mathcal{NP}$ for which any problem's instance may be converted to an instance of another problem in $\mathcal{NP}$-complete in polynomial time, then

$$\mathcal{P} \subsetneq \mathcal{NP} \Rightarrow \mathcal{NP}\text{-complete} \cap \mathcal{P} = \varnothing$$

Basic complexity classes



- $\mathcal{P}$ solvable in polynomial time by a computer (Turing Machine)
- $\mathcal{NP}$ solvable in polynomial time by a nondeterministic computer
- $\mathcal{NP}$-complete converted to other $\mathcal{NP}$-complete problems in polynomial time

How to deal with hard problems

- What should you do when you encounter an apparently hard problem?
- Is it in $\mathcal{NP}$-complete?
- If not, solve it
- If so, then what?

Despair. Solve it! Resort to a suboptimal heuristic.
Bad, but sometimes the only choice. Develop an approximation algorithm.
Better. Determine whether all encountered problem instances are constrained.
Wonderful when it works.

One example

O. Coudert. Exact coloring of real-life graphs is easy. *Design Automation*, pages 121–126, June 1997

Properties of complete optimization techniques

- If a solution exists, will be found
- Very slow for some problems
- Good formal understanding of complexity

Example complete algorithms

- Enumeration
- Branch and bound
- Dynamic programming
- Integer-linear programming
- Backtracking iterative improvement

Enumeration

- Considers all possible solutions
- Extremely slow for large n
- Potentially has low constant factor, may be O.K. for small n

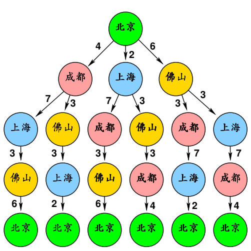
Example problem

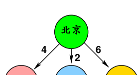
Traveling salesman problem

Find shortest path visiting all cities.

Traveling salesman problem

Enumeration





Branch and bound

- Keep track of minimal encountered cost
- When a path has a higher cost, terminate

Branch and bound

Branch and bound

- Better average-case complexity
- Still worst-case exponential

Linear programming

- In $\mathcal{P}$– Ellipsoid Algorithm / internal point methods
- However, in practice WC exponential Simplex Algorithm better
- Goal: Maximize a linear weighted sum under constraints on variables

Linear programming

Maximize

$$c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n$$

where

$$\forall c_i \in c, c_i \in R$$

subject to the following constraints:

$$\begin{array}{ll} a_{11} \cdot x_1 + a_{12} \cdot x_2 + \cdots + a_{1n} \cdot x_n \leq, =, \geq & b_1 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + \cdots + a_{2n} \cdot x_n \leq, =, \geq & b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{n1} \cdot x_1 + a_{1n} \cdot x_2 + \cdots + a_{nn} \cdot x_n \leq, =, \geq & b_n \\ \forall x_i \in x, x_i \geq 0 \qquad \forall a_{jk} \in A, a_{jk} \in R & \end{array}$$

Linear programming

- Can be formulated as a linear algebra problem
  - Vector x of variables
  - Vector c of cost
  - Matrix A of constraints
  - Vector b of constraints
- Maximize or minimize $c^T x$
- Satisfy $Ax \leq b$
- Satisfy $x \geq 0$

Integer-linear programming (ILP)

- ILP is $\mathcal{NP}$-complete
- LP with some variables restricted to integer values
- Formulate problem as ILP problem
  - Excellent understanding of problem
  - Good solvers exist
- Variants – both NP-complete
  - Mixed ILP has some continuous variables
  - Zero-one ILP

Example – ILP formulation for the travaling salesman problem

Let T be a tentative solution, or tour
$\forall e \in E$ let there be a variable

$$t_e = \begin{cases} 1 & \text{if } e \in T \\ 0 & \text{if } e \notin T \end{cases}$$

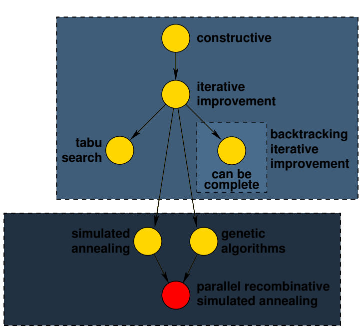
Constraint: Given that S is a set of vertices, $\mathbf{con}(S)$ is the set of edges connecting $v \in S$ to $v \notin S$, and $\{v_i\}$ is the vertex set containing only v_i, every vertex, v_i must be connected to two edges of the tour

$$\forall v_i \in V, \sum_{e \in \mathbf{con}(\{v_i\})} = 2$$

Backtracking iterative improvement

- Allows B steps of backtracking
- Can be incomplete
- Complete if B = the problem decision depth
- Allows use of problem-speficic heuristics for ordering
- Incomplete if $B <$ decision depth
- More on this later

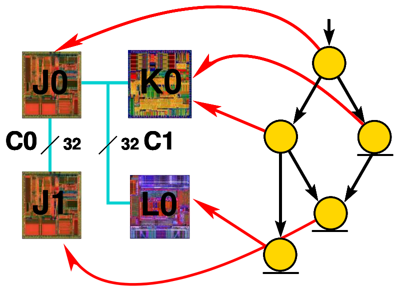
Optimization techniques



Constructive algorithms

- Build solution piece by piece
- Once complete solution is generated, don't change
- Typically fast
- Easy to use problem-specific information
- Easy to implement
- Prone to becomming trapped in poor search space

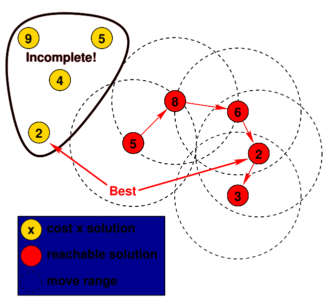
Constructive algorithms example



Iterative improvement

- Starts with complete but poor solution
  - therefore contains constructive algorithm
  - superset of constructive
- Makes changes to solution to improve it
- Typically fast
- Easy to use problem-specific information
- Easy to implement
- Prone to becomming trapped in local minima

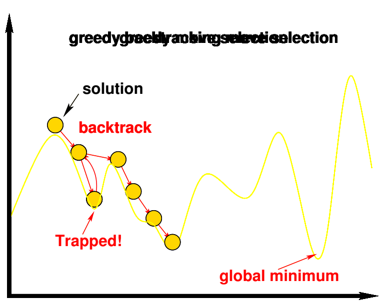
Local minima move size



Local minima

- Even if all solutions reachable, may not get best solution
- Depends on move selection

Trapped in local minima



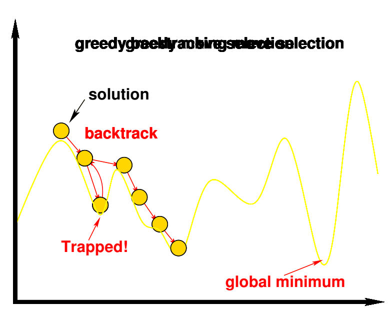
Local mimina

- Being trapped in local minima is a big problem
- Numerous probabilistic optimization techniques designed
  - avoid local minima
  - find global minima
  - do so efficiently

Backtracking iterative improvement

- Backtracking iterative improvement is complete if
  - all solutions are reachable
  - the backtracking depth $\geq$ search depth
  - . . . however, this can be slow
- Even if incomplete, backtracking can improve quality
- Can trade optimization time for solution quality
- Greedy iterative improvement if backtracking depth is zero

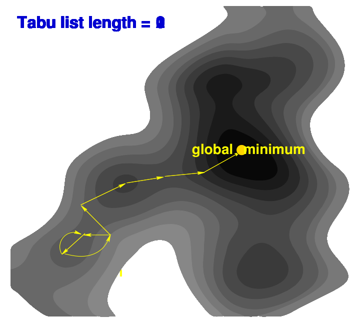
Backtracking



Tabu search

- Similar to interative improvement
- Iterative improvement can cycle
  - chooses largest cost decrease move. . .
  - . . . then chooses smallest cost increase move
- Tabu search has a *tabu list*
  - solutions to avoid
  - solution characteristics to avoid
- Prevents iterative cycles

Tabu search example



Simulated annealing

- Inspired by annealing of metals
- Start from high temperature and gradually lower
- Avoids local minima traps
- Generate trial solutions
- Conduct Boltzmann trials between old and new solution

Simulated annealing

- Easy to implement
- Can trade optimization time for solutions quality
- Greedy iterative improvement if temperature is zero
- Famous for solving difficult physical problems, e.g., placement

Boltzmann trials

Solution are selected for survival by conducting Boltzmann trials between parents and children.
Given a global temperature T, a solution with cost K beats a solution with cost J with probability:

$$\frac{1}{1 + e^{(J\text{-}K)/T}}$$

Boltzmann trials

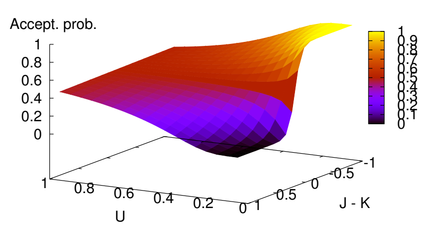
Introduce convenience variable U

$$U(T) = 1 - \frac{1}{T+1}$$

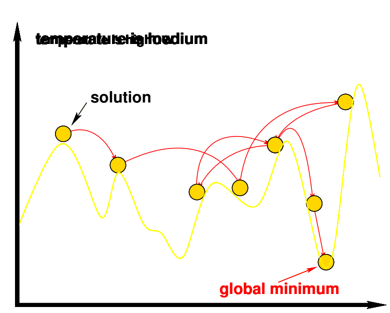
$$U(0) = 0$$

$$T \to 1 \Rightarrow U(T) \to \infty$$

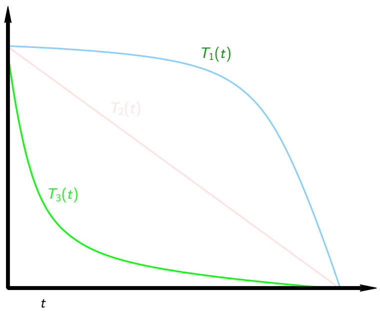
Boltzmann trials



Simulated annealing example



Cooling schedule often not important



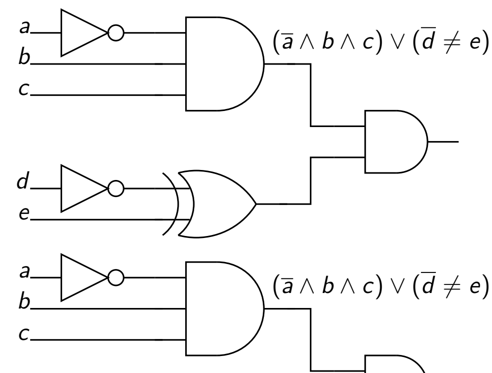
Simulated annealing notes

- Time complexity extremely difficult to analyze
- Given a slow enough cooling schedule, will get optimum
  - This schedule sometimes makes simulated anealing slower than exhaustive search
  - Determining optimal schedule requires detailed knowledge of problem's Markov chains

Genetic algorithms

- Multiple solutions
- Local randomized changes to solutions
- Solutions share information with each other
- Can trade optimization time for solution quality
- Good at escaping sub-optimal local minima
- Greedy iterative improvement if no information sharing
- Difficult to implement and analyze
- Researchers have applied in testing, system synthesis

Solution representation



Mutation

- Choose an element of the solution
- Change it to another value
- Local modification, similar to that in iterative improvement

Solution representation

a	d	b	e	c
0	0	1	0	1

a	d	b	e	c
0	0	1	0	1

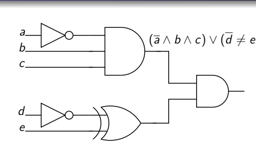
a	d	b	e	c
0	0	1	1	1

Crossover

a	d	b	e	c
0	0	1	1	1

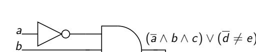
a	d	b	e	c

Locality not preserved

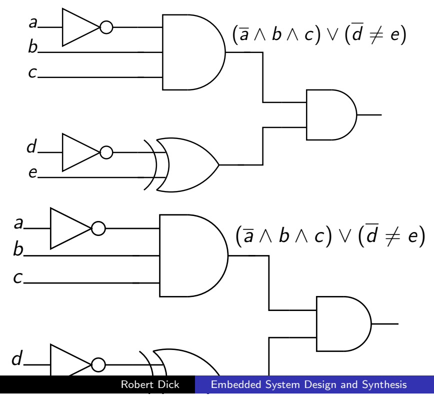


a	d	b	e	c
0	0	0	0	1

0	1	1	1	1



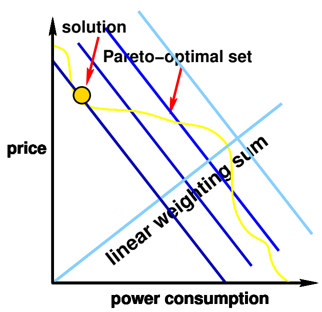
Locality preserved



Multidimensional optimization

- Real-world problems often have multiple costs
  - Price
  - Power consumption
  - Speed
  - Temperature
  - Reliability
  - etc.
- Necessary to simultaneously minimize all costs

Linear weighting sum



Pareto-ranking

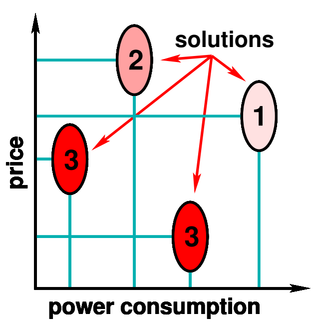
A solution dominates another if all its costs are lower, i.e.,

$$\mathbf{dom}_{a,b} = \forall_{i=1}^{n} cost_{a,i} < cost_{b,i} \wedge a \neq b$$

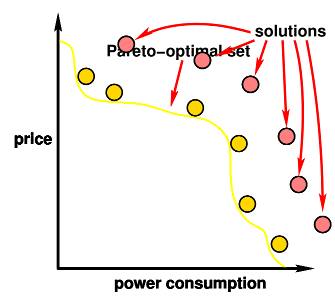
A solution's rank is the number of other solutions which do not dominate it, i.e.,

$$\mathbf{rank}_{s'} = \sum_{i=1}^{n} \mathbf{not\ dom}_{s_i,s'}$$

Pareto-ranking



Pareto-rank based multiobjective optimization



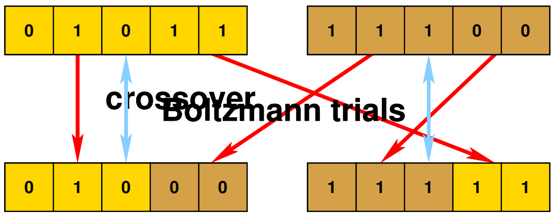
Genetic algorithm selection

- Solutions are selected for survival by cost or rank
- Resistant to becoming trapped in local minima
  - mutation
  - crossover
- Possible to do better?

PRSA

- Genetic algorithm where Boltzmann trials are used for solution selection
- Genetic algorithm if temperature is set to zero
- Simulated annealing if only one solution
- Easily parallelizable
- Has strengths of genetic algorithms and simulated annealing
- Difficult to implement but not more difficult than genetic algorithms

PRSA example



Multiobjective GAs required reading

Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proc. Int. Conf. Genetic Algorithms*, pages 416–423, July 1993

- Explains importance of multiobjective optimization
- Shows simple way to use Pareto-rank in parallel optimization meta-heuristics

Optimization reference

Robert P. Dick. *Multiobjective Synthesis of Low-Power Real-Time Distributed Embedded Systems*. PhD thesis, Dept. of Electrical Engineering, Princeton University, July 2002

- Chapter 4 contains an overview of some of the popular probabilistic optimization techniques used in CAD
- Chapters 5 and 6 describe a PRSA for system synthesis.

Genetic algorithms reference

David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, MA, 1989

- The most basic and complete book on genetic algorithms
- Weak on multiobjective potential this meta-heuristic

Evolutionary algorithms required reading

D. Graham-Rowe. Radio emerges from the electronic soup. *New Scientist*, August 2002

- Interesting short article on a phyical application on evolutionary algorithms
- Similar results for FPGA-based filter

PRSA reference

Samir W. Mahfoud and David E. Goldberg. Parallel recombinative simulated annealing: A genetic algorithm. *Parallel Computing*, 21:1–28, January 1995

Definitions

Let Γ_{j_1,j_2} represents the dependency between tasks j_1 and j_2 where

$$\Gamma_{j_1,j_2} = \begin{cases} 1 & \text{if task } j_1 \text{ is an immediate predecessor of } j_2 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$\delta(j,m) = \begin{cases} 1 & \text{if task } j \text{ is assigned to core } m \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Credit to Tam Chantem and Xiaobo Sharon Hu for formulation.

Definitions

$$\alpha(j,k) = \begin{cases} 1 & \text{if task } j \text{ starts at time instant } k \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$\sigma(j,k) = \begin{cases} 1 & \text{if task } j \text{ ends at time instant } k \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Constraints

- Every task j is assigned to exactly one core m:

$$\forall j \in J \qquad \sum_{m \in M} \delta(j, m) = 1 \qquad (5)$$

- Every task j meets its deadline:

$$\forall j \in J \qquad ts(j) + te(j) \leq d(j) \qquad (6)$$

- Precedence constraints are honored:

$$\forall j \in J \qquad ts(j_2) \geq tf(j_1) \cdot \Gamma_{j_1, j_2} \qquad (7)$$

$$\forall k, \forall j_1, j_2 \in J \qquad \sum_{k'=0}^{k} \left(\sigma(j_1, k') - \alpha(j_2, k')\right) \cdot \Gamma_{j_1, j_2} \geq 0 \qquad (8)$$

Constraints

- Every task has only one start time instant:

$$\forall j \in J \qquad \sum_{k=0}^{2|J|-1} \alpha(j, k) = 1 \qquad (9)$$

- Every task has only one finish time instant:

$$\forall j \in J \qquad \sum_{k=0}^{2|J|-1} \sigma(j, k) = 1 \qquad (10)$$

- The start time and the finish time instants of a task must be different:

$$\forall j \in J, \forall k \qquad \alpha(j, k) + \sigma(j, k) \leq 1 \qquad (11)$$

Constraints

- At each time instant, at most one task can be active on a core:

$$\forall k, \forall m \in M \qquad \sum_{j \in J} \beta(k, j, m) \leq 1 \qquad (12)$$

- A task j must start before it ends:

$$\forall j \in J, \forall k \qquad \sum_{k'=0}^{k} \alpha(j, k') \geq \sum_{k'=0}^{k} \sigma(j, k') \qquad (13)$$

Constraints

If tasks j_1 and j_2 both execute on core m, they must not overlap:

$$\forall j_1, j_2 \in J : j_1 \neq j_2, \forall m \in M, \forall k$$

$$tf(j_1) \leq (2 - \delta(j_1, m) - \delta(j_2, m)) \cdot \Lambda + ts(j_2) + \left(1 - \sum_{k'=0}^{k} \left(\sigma(j_1, k') - \alpha(j_2, k')\right)\right) \cdot \Lambda \qquad (14)$$

$$tf(j_2) \leq (2 - \delta(j_1, m) - \delta(j_2, m)) \cdot \Lambda + ts(j_1) + \left(1 - \sum_{k'=0}^{k} \left(\alpha(j_2, k') - \sigma(j_1, k')\right)\right) \cdot \Lambda \qquad (15)$$

Past work of others

- **1992**: Optimal MILP co-synthesis of small systems [Prakash & Parker], [Bender], [Schwiegershausen & Pirsch]
- **1993**: One CPU-One ASIC [Ernst, Henkel & Benner], [D'Ambrosio & Hu], [Barros, Rosenstiel, & Xiong], others. . .
- **1994**: Software generation and delay estimation [Gupta & De Micheli], [Li, Malik, & Wolfe]
- **1997**: Iterative improvement algorithm for co-synthesis of distributed embedded systems [Wolf]

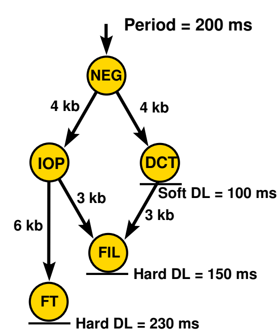
Past work of others

- **1997**: Constraint logic programming for embedded system price minimization under time constraints [Kuchcinski]
- **1998**: Automatic ANSI-C partitioning among homogeneous processors on a single chip [Karkowski & Corporaal]
- **2000**: Automatic performance estimatation for ASICs used in co-synthesis [Xie & Wolf]

Overview of our system synthesis tools

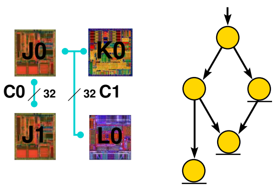
- **TGFF and CETA**: Automatic generation of parametric, pseudo-random or real task graphs.
- **MOGAC**: Distributed multi-rate embedded systems w. heterogeneous hardware, hard real-time constraint, price optimization, and power optimization.
- **CORDS**: Distributed multi-rate embedded systems w. reconfigurable hardware, and hard real-time constraints, and price optimization.
- **COWLS**: Client-server embedded systems communicating via primary channel of limited bandwidth, e.g., a wireless link, w. heterogeneous hardware, hard real-time constraints, soft real-time constraints, power optimization, and price optimization.
- **MOCSYN**: System-on-a-chip composed of hard cores w. hard real-time constraints, power optimization, area optimization, and price optimization.

Review: Specifications



- Specify the task deadlines, periods, and data dependencies
- Analyze performance of each task on each resource
- Allocate resources
- Assign each task to a resource
- Schedule the tasks on each resource

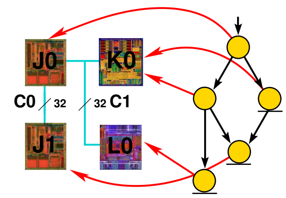
Review: Allocation



Number and types of:

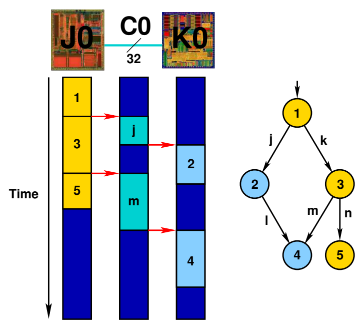
- PEs or cores
- Commun. resources

Review: Assignment



- Assignment of tasks to PEs
- Connection of communication resources to PEs

Review: Scheduling



Costs

Soft constraints:

- Price
- Power
- Area
- Soft deadline violations

Hard constraints:

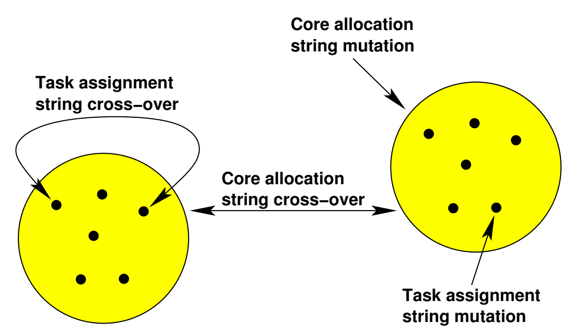
- Hard deadline violations
- PE overload
- Unschedulable tasks
- Unschedulable transmissions

Solutions which violate hard constraints not shown to designer – pruned out.

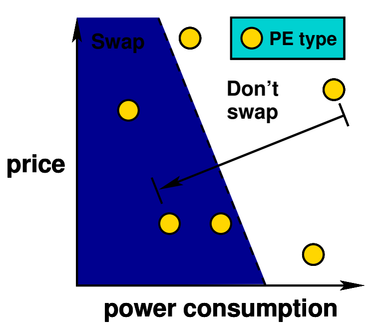
Genetic algorithms

- Multiple solutions
- Randomized changes to solutions
- Solutions share information with each other
- Can escape sub-optimal local minima
- Scalable

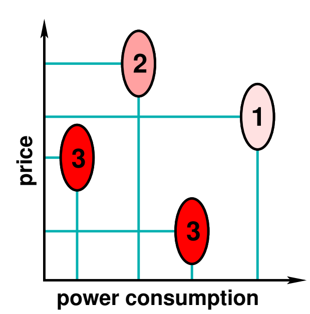
Cluster genetic operator constraints



Information trading



Ranking



A solution dominates another if it is better in all ways.

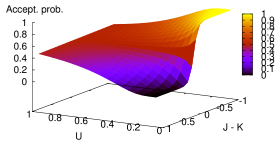
A solution's rank is the number of other solutions which do not dominate it.

Each solution has numerous costs, e.g., price, deadline violation, and CLB over-use.

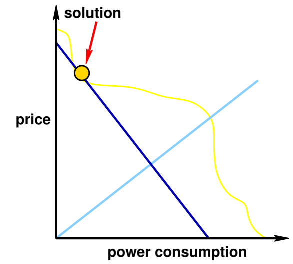
Reproduction

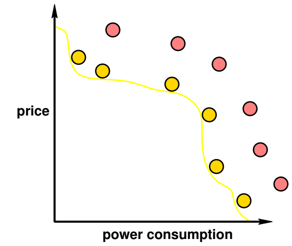
Solution are selected for reproduction by conducting Boltzmann trials between randomly selected pairs of solutions.
Given a global temperature T, a solution with rank J beats a solution with rank K with probability:

$$\frac{1}{1+e^{(K-J)/T}} \tag{16}$$



Multiobjective optimization

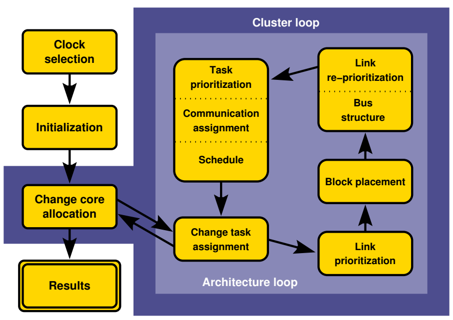




Multiple runs based on weighted sums waste effort on unpromising

Optimizer concentrates on promising areas of pareto-optimal curve. Solutions share information

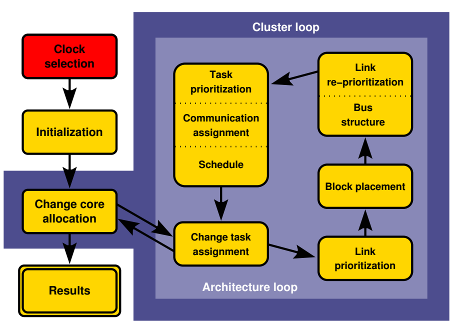
Mocsyn algorithm overview



Mocsyn related work

- **1982**: Algorithm for improving network partitions [Fiduccia & Mattheyses]
- **1983**: Method to find optimal orientations of cells in slicing floorplan block placements [Stockmeyer]
- **1997**: Interconnect design for deep submicron ICs [Cong]

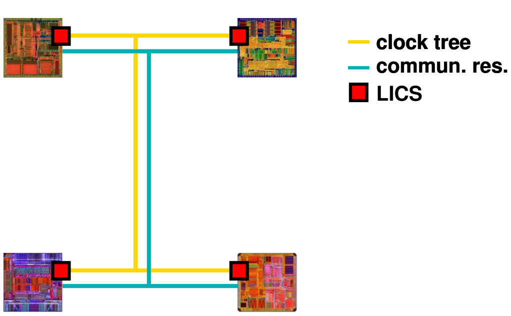
Mocsyn algorithm overview



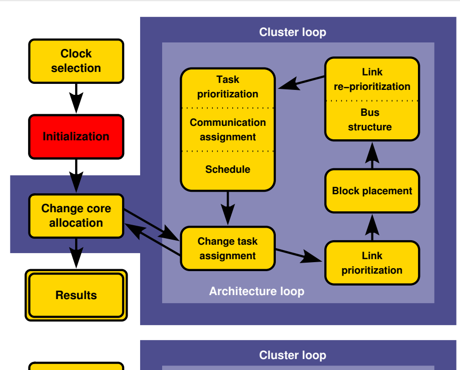
Clock selection

- Cores have different maximum frequencies
- Globally synchronous system forces underclocking
- Multiple crystals too expensive
- Use linear interpolating clock synthesizers
  - Standard CMOS process
  - Each core runs near highest speed
  - Global clock frequency can be low to reduce power
- Optimal clock selection algorithm in pre-pass

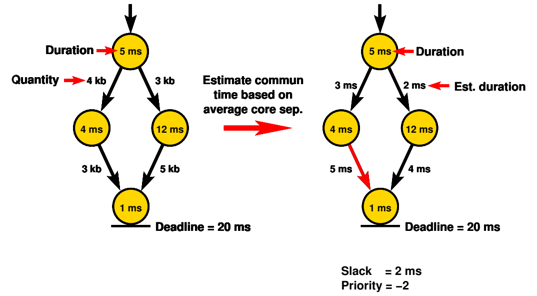
Clock selection



Mocsyn algorithm overview

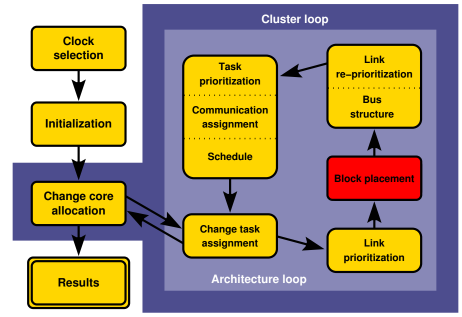


Link prioritization



Optimization for synthesis
Synthesis algorithms
Homework
MILP formulation for assignment/scheduling problem
Example system synthesis algorithm

MOCSYN algorithm overview



Block placement to determine communication time, energy

Optimization for synthesis
Synthesis algorithms
Homework
MILP formulation for assignment/scheduling problem
Example system synthesis algorithm

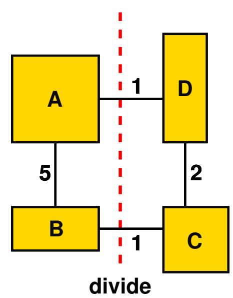
Floorplanning block placement

Balanced binary tree of cores formed
Division takes into account

- Link priorities
- Area of cores on each side of division

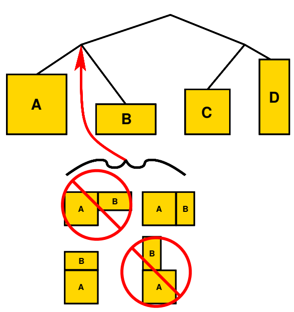
Optimization for synthesis
Synthesis algorithms
Homework
MILP formulation for assignment/scheduling problem
Example system synthesis algorithm

Floorplanning block placement



Optimization for synthesis
Synthesis algorithms
Homework
MILP formulation for assignment/scheduling problem
Example system synthesis algorithm

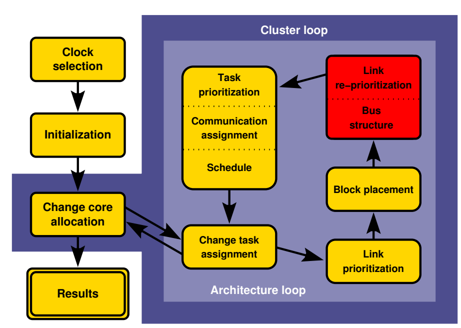
Floorplanning block placement



$\mathcal{O}\left(n^2\right)$

Optimization for synthesis
Synthesis algorithms
Homework
MILP formulation for assignment/scheduling problem
Example system synthesis algorithm

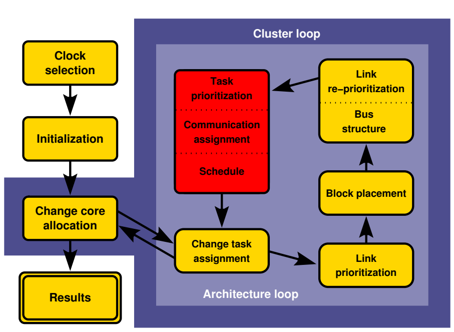
MOCSYN algorithm overview



Bus topology generation: minimize contention under routability constraints

Optimization for synthesis
Synthesis algorithms
Homework
MILP formulation for assignment/scheduling problem
Example system synthesis algorithm

MOCSYN algorithm overview

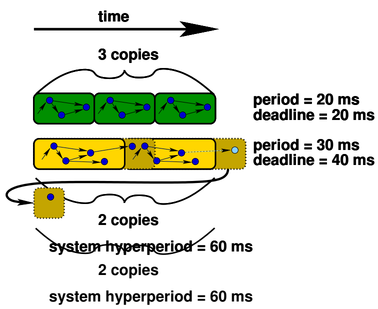


Optimization for synthesis
Synthesis algorithms
Homework
MILP formulation for assignment/scheduling problem
Example system synthesis algorithm

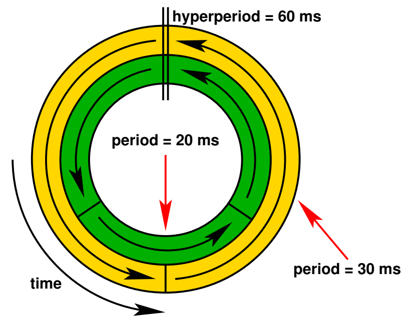
Scheduling

- Fast list scheduler w. slack, EST, LFT prioritization
- Multi-rate
- Handles period $<$ deadline as well as period $\geq$ deadline
- Uses alternative prioritization methods: slack, EST, LFT
- Other features depend on target

Scheduling



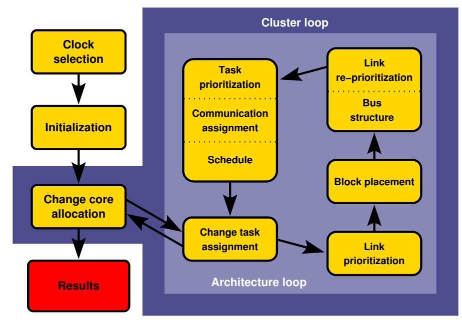
Scheduling



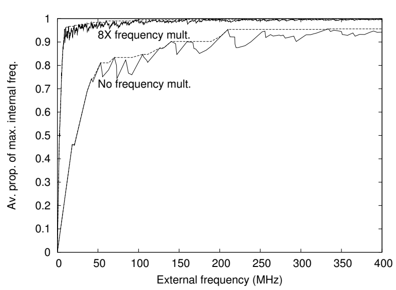
Cost calculation

- Price
- Average power consumption
- Area
- PE overload
- Hard deadline violation
- Soft deadline violation
- etc.

MOCSYN algorithm overview



Clock selection quality



Eight cores, random frequencies from 2 MHz to 100 MHz

MOCSYN feature comparisons experiments

Example	MOCSYN price ($)	Worst-case commun. price ($)	Best-case commun. price ($)	Single bus price ($)
...	...	...	...	...
15	216	n.a.	n.a.	n.a.
16	138	n.a.	n.a.	177
17	283	n.a.	n.a.	n.a.
18	253	n.a.	n.a.	253
19	211	n.a.	n.a.	n.a.
...	...	...	...	...
Better		38	44	28
Worse		3	1	9

17 processors, 34 core types, five task graphs, 10 tasks each, 21 task types from networking and telecom examples

Embedded system synthesis benchmarks suite (E3S)

- Uses performance information from Embedded Microprocessor Benchmarks Consortium (EEMBC)
- Power numbers from processor datasheets
- Performance numbers based on measured execution times of 47 tasks
- Additional information by emailing and calling numerous processor vendors

E3S

17 processors e.g.,

- AMD ElanSC520
- Analog Devices 21065L
- Motorola MPC555
- Texas Instruments TMS320C6203

E3S

Numerous communication resources, e.g.,

- CAN
- IEEE1394
- PCI
- USB 2.0
- VME

E3S

- One task set for each EEMBC application suites
  - Automotive/industrial
  - Consumer
  - Networking
  - Office automation
  - Telecommunications

MOCSYN E3S experiments

Example	Price ($)	Average power (mW)	Soft DL viol. prop.	Area (mm^2)
automotive-industrial	91	120	0.60	3.0
	91	120	0.61	2.0
	110	113	0.88	4.0
	110	115	0.60	4.0
networking	61	72	0.94	38.4
telecom	223	246	2.31	9.9
	223	246	2.76	6.0
	233	255	3.47	4.5
	236	247	2.29	9.9
	236	249	2.60	8.0
	242	221	2.67	3.0
	242	230	2.44	25.9
	242	237	1.72	6.0
	272	226	2.22	192.1
	272	226	2.34	9.4
	353	258	1.23	4.0
consumer	134	281	1.40	34.1
	134	281	1.50	21.6
office automation	64	370	0.23	36.8
	66	55	0.00	7.2

MOGAC run on Hou's examples

Example	Yen's System		MOGAC		
	Price ($)	CPU Time (s)	Price ($)	CPU Time (s)	Tuned CPU Time (s)
Hou 1 & 2 (unclustered)	170	10,205	170	5.7	2.8
Hou 3 & 4 (unclustered)	210	11,550	170	8.0	1.6
Hou 1 & 2 (clustered)	170	16.0	170	5.1	0.7
Hou 3 & 4 (clustered)	170	3.3	170	2.2	0.6

Robust to increase in problem complexity.

2 task graphs each example, 3 PE types
Unclustered: 10 tasks per task graph Clustered: approx. 4 tasks per task graph

MOGAC run on Prakash & Parker's examples

Example ⟨Perform⟩	Prakash & Parker's System		MOGAC		
	Price ($)	CPU Time (s)	Price ($)	CPU Time (s)	Tuned CPU Time (s)
Prakash & Parker 1 ⟨4⟩	7	28	7	3.3	0.2
Prakash & Parker 1 ⟨7⟩	5	37	5	2.1	0.1
Prakash & Parker 2 ⟨8⟩	7	4,511	7	2.1	0.2
Prakash & Parker 2 ⟨15⟩	5	385,012	5	2.3	0.1

Quickly gets optimal when getting optimal is tractable

3 PE types, Example 1 has 4 tasks, Example 2 has 9 tasks

MOGAC run on Yen's large random examples

Example	Yen's System		MOGAC		
	Price ($)	CPU Time (s)	Price ($)	CPU Time (s)	Tuned CPU Time (s)
Random 1	281	10,252	75	6.4	0.2
Random 2	637	21,979	81	7.8	0.2

Handles large problem specifications.

No communication links: communication costs = 0

Random 1: 6 task graphs, approx. 20 tasks each, 8 PE types
Random 2: 8 task graphs, approx. 20 tasks each, 12 PE types

MOCSYN contributions, conclusions

First core-based system-on-chip synthesis algorithm

- Novel problem formulation
- Multiobjective (price, power, area, response time, etc.)
- New clocking solution
- New bus topology generation algorithm

MOCSYN contributions, conclusions

Important for system-on-chip synthesis to do

- Clock selection
- Block placement
- Generalized bus topology generation

Reference

- Robert P. Dick. *Multiobjective Synthesis of Low-Power Real-Time Distributed Embedded Systems*. PhD thesis, Dept. of Electrical Engineering, Princeton University, July 2002
  - Chapter 7

Synthesis problems of current interest

- Synthesis of reliable systems
  - Lifetime fault processes
  - Transient faults
- Cross-level synthesis
- Temperature-aware synthesis
- Novel technologies
- New application domains
  - E.g., wireless sensor networks
- Efficient optimal techniques for constrained problems

Formal methods and MILP-based synthesis references

High-level overview of formal methods in embedded system design

Steven D. Johnson. Formal methods in embedded design. *IEEE Computer*, 36(11):104–106, November 2003

MILP formulation for heterogeneous multiprocessor synthesis

S. Prakash and A. Parker. Synthesis of application-specific multiprocessor architectures. In *Proc. Design Automation Conf.*, June 1991

Assignment: Write a short paragraph describing the most important points in both of these articles.

Reliability optimization reference

- Changyun Zhu, Zhenyu Gu, Robert P. Dick, and Li Shang. Reliable multiprocessor system-on-chip synthesis. In *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, September 2007. To appear
- If you are interested in the topic, please follow the references in that paper. In particular consider the papers by Xie, Glaß, and Coskun.

Next class

- Formal methods for reliable embedded system design
- Real-time systems
- Scheduling