Cooperative Negotiation in Autonomic Systems using Incremental Utility Elicitation

Craig Boutillier (University of Toronto)
Rajarshi Das (IBM Research)           Gerald Tesauro (IBM Research)
Jeffrey O. Kephart (IBM Research)    William E. Walsh (IBM Research)
Proc. 19th Annual Conf. Uncertainty in Artificial Intelligence (UAI-03)
http://www.cs.toronto.edu/~cebly/Papers/download_autonomic.pdf

• Autonomic computing: computer systems managing themselves.
  • Self-configuring, self-optimizing, self-healing, self-protecting.
  • Problem of optimally distributing resources to workloads in a distributed environment.
  • Distributed group of workload managers (WMs) require resource.
  • A provisioner (P) must allocate this resource to maximize utility.

• Utility function: \( u_i(a_i) \) for WM\(_i\) maps fraction of allocation \( a_i \) to utility.
  • Assume: \( u_i \) are a comparable metric (ie. Adding them gives a meaningful result).
  • Assume: \( u_i \) is monotonically non-decreasing.
    • If more resource lowers utility, we just throw away resource!
  • Utility function not known, very expensive to calculate at a point.
  • P will ask WMs for samples of utility functions to reduce uncertainty and to help search for an optimal allocation.

• Allocation \( a = \{a_1, ..., a_n\} \) \( a_i \geq 0 \) \( \sum_i a_i \leq 1 \)

• Value:
  \[
  V(a, u) = \sum_i u_i(a_i) 
  \]

• Upper bound on useful resource alloc:
  \[
  a^T \text{ s.t. } a_i \geq a^T \Rightarrow u_i(a_i) = u_i(a^T) 
  \]

• Max regret: P collects samples from each WM: \( 0 = \tau_i^0 < \tau_i^1 < ... < \tau_i^k = a^T_i \)
  • \( a_i \) in bin \( b^j_i \) iff \( \tau_i^{j-1} < a_i < \tau_i^j \) \( u_i(\tau_i^{j-1}) \leq u_i(a_i) \leq u_i(\tau_i^j) \)
  • Notation: \( [a_i] = j \)
  • \( u_i \) feasible if nondecreasing and consistent with samples.
  • \( S = \) samples, \( U = \) set of feasible utility vectors for \( S \)
  • Max regret of a w.r.t. \( a' \): \( MR(a, a') = \max_{u \in U} V(a', u) - V(a, u) \)
  • Max regret (general): \( MR(a) = \max_{a' \in A} MR(a, a') \)
  • Minimax regret: \( a^* = \arg \min_{a \in A} MR(a) \)
  • Minimax regret level: \( MMR(U) = MR(a^*) \)

• Worst case (always gives MR): step function!
  • \( u_i(\tau_i^{a_i^{j-1}}) \) for \( \tau_i^{a_i^{j-1}} < a_i \), \( u_i(\tau_i^{a_i^{j-1}}) \) for \( a_i, \tau_i^{a_i^j} \) , max in all other bins.
  • Find allocation witness \( a^w \) to maximize \( V(a^w, u) - V(a, u) \)
  • Solve mixed integer program.
    • \( A_i = \) alloc, \( B_i \) is 0/1 (false/true) alloc \( i \) is in bin \( j \)
Maximize: \[ \sum_{i \leq n, j \leq k} B_i^j u_i^j \]

Subject to:
\[ 0 \leq A_i \leq a_i \tau_i, \quad \sum_i A_i \leq 1 \quad \forall j \leq k, \quad \sum_i B_i^j = 1 \quad \forall j > 1, A_i/(\tau_i^{j-1}) - B_i^j \geq 0 \quad \forall j < k, B_i^j - a_i \tau_i - A_i a_i \tau_i \leq 0 \]

- **Minimax Regret**: pointwise alloc \( p = \text{alloc exactly on sample points} \).
- Exhaustive pointwise alloc = cannot increase any bin to next sample point w/o exceeding allocation.
- Supporting pointwise alloc \( \text{SPA}(a) \) is set of pointwise allocs using nearest sample point below each allocation.
- \( \text{MR}(a) \leq \text{MR}(\text{SPA}(a)) \)
- Extensions \( \text{E}(p) = \text{SPA}(p) \) where \( p \) is an EPA.
- Surplus \( \delta \text{s.t. } a = \text{SPA}(a) + \delta \)
- EPAs can be enumerated, we can find \( \text{MMR}(\text{E}(p)) = \min_{a \in \text{E}(p)} \text{MR}(a) \) iteratively.
  - Choose \( a \) in \( \text{E}(p) \), find \( a^w \). If \( a^w \) and \( a \) share no bins, \( \text{MR}(a) = \text{MR}(p) \).
  - Else tighten upper bound (see paper!)
- Can easily get bounds on \( \text{MMR}(\text{E}(p)) \), so can do intelligent search.
- Can use optimistic or greedy strategies to find good allocs without crazyness above.

- **Elicitation**: Find MMR alloc \( a \) for samples \( S \): \( \text{MR}(a) \) too high? Elicit more points!
  - Worst case: \( u \) arbitrarily close to step function from 0 to \( \epsilon \)
    - No finite \# of queries can bring \( \text{MR} \leq \frac{\max_{e_i} \epsilon_i}{2} \)
    - Can find this in \( < 2n(n-1) \) queries per WM.
      - Start with 0 and \( a_i \tau_i \), then keep dividing all bins in half.
      - Works for worst case, but not good for general case.
  - Intuitive strategy: probe bins of allocation or MR witness.
    - Each probe will reduce MR, or give more info about utility func.
    - Found experimentally that midpoint of bin was most effective.
    - Choose alloc or witness based on heuristic.
      - Scaled bin area: \( \Delta u + \Delta t = (\tau_i^{j-1})/a_i \), \( \Delta u = (u_i - u_i^{j-1})/a_i^{j-1} \)

- **Experiment**: 3 and 4 WM setups, running two transaction classes each.
  - Utility step function: high utility below certain response time.
  - Response time based on resource alloc. using M/M/1 queue model.
  - GNU Linear Programming Toolkit 3.2.4.
  - Exponential drop in MMR with heuristic elicitation.
  - Much better than random queries or halving method.
  - GLPK time approx. \( N^q \) (\( N=\text{WMs, q=queries} \)
    - \( N > 4, q > 15 \) going to be infeasible to run.

- **Future work**:
  - Use real MIP solver like CPLEX (> 10x speed?)
  - Faster algorithms to find MMR (greedy?)
  - Multidimensional, aka. multiple resources.
  - Bayesian approaches; learning, prior knowledge.