Utility-based Optimal Service Selection for Business Processes in Service Oriented Architectures

Vinod K. Dubey, Daniel A. Menascé

The Eight Intl. Conf. Web Services, IEEE Computer Society, Miami, FL, July 5-10, 2010

Presented by David González

Overview

- Objectives and Concluding Remarks
- QoS composition, cost and utility functions
- JOSeS vs HCB
- Experiment
- Results
- Discussion

Objectives and Concluding Remarks

- Address optimal service selection problem for business processes in SOA environments.
- Provided a optimal solution, Extended JOSeS, and a heuristic solution, HCB.
- The heuristic solution performed 99.5% close to the optimal solution using significantly less points from the solution space and computing resources.

- Now let's see how to get there.

Optimization Problem

$$\text{Maximize} \quad U(E[R(z)], A(z), X(z))$$

subject to

$$E[R(z)] \leq R_{\max}$$
$$A_{\min} \leq A(z) \leq 1$$
$$X(z) \geq X_{\min}$$
$$C(z) \leq C_{\max}$$
$$z \in \mathcal{Z}$$

Assumptions and BPEL

- Availability and Throughput are deterministic.
- End-to-end execution time and cost are nondeterministic.

A Heuristic Approach to Optimal Service Selection in Service Oriented Architectures

```
<sequence>
  <invoke a1>
  <switch>
    <case q1>
      <flow>
        <invoke a2>
        <sequence>
          <invoke a3>
          <invoke a4>
        </sequence>
      </flow>
    <case q2=(1-q1)>
      <invoke a5>
  </switch>
  <invoke a6>
</sequence>
```

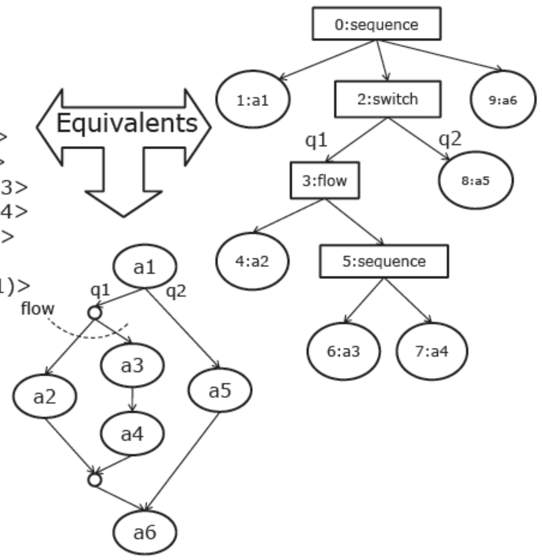

Figure 1: An example of a BPEL business process on the left, the corresponding BPTree on the right, and an execution graph on the middle.

Computing Availability and Throughput

q_i is the probability that activity a_i is invoked [10].

Algorithm 1 Availability Computation of a BPEL process

function A(node i)
if label(i) = `leaf node` **then**
  return A_i;
else
  if label(i) = `sequence` **then**
    return $\prod_{k \in \text{children(i)}} A(k)$;
  else if label(i) = `switch` **then**
    return $\sum_{k \in \text{children(i)}} q_k \times A(k)$;
  else if label(i) = `flow` **then**
    return $\prod_{k \in \text{children(i)}} A(k)$;
  end if
end if

Algorithm 2 Throughput Computation of a BPEL process

function X(node i)
if label(i) = `leaf node` **then**
  return X_i;
else
  if label(i) = `sequence` **then**
    return $\min_{k \in \text{children(i)}} X(k)$;
  else if label(i) = `switch` **then**
    return $\sum_{k \in \text{children(i)}} q_k \times X(k)$;
  else if label(i) = `flow` **then**
    return $\min_{k \in \text{children(i)}} X(k)$;
  end if
end if

Computing end-to-end execution time

$$E[\max_{i=1}^{n} R_i] = \int_0^\infty x \left[\prod_{i=1}^{n} P_i(x)\right] \sum_{i=1}^{n} \frac{p_i(x)}{P_i(x)} dx \qquad (1)$$

The expected value of a maximum of a set of independent random variables[10]

Utility functions

$$U_i(v(z)) = K_i \frac{e^{\alpha_i(\beta_i - v(z))}}{1 + e^{\alpha_i(\beta_i - v(z))}} \quad (2)$$

$$U_g(z) = \left(\prod_{i=1}^{3} (U_i(z))^{w_i} \right)^{\frac{1}{\sum_j^3 w_j}} \quad (3)$$

JOSeS vs HCB

- Jensen-based Optimal Service Selection (JOSeS). This algorithm does not require one to generate the entire solution space Z, but only a subset of the solution space where each point represents a feasible solution.

- Hill-Climbing Based (HCB), which defines a neighborhood of the point currently being visited and move to the best point in the neighborhood. The process continues until a near-optimum solution is found given a stopping criterion

Optimal Solution: JOSeS

Algorithm 3 AdvanceList Function

function `AdvanceList (k) returns (s)`
$s \leftarrow$ `next (k);`
if $s =$ NULL **then**
  if $k > 1$ **then**
    `reset (k);` $k \leftarrow k - 1$; $z \leftarrow z\diamond$;
    `AdvanceList (k);`
  else
    return s
  end if
else
  return s;
end if
end function

Algorithm 4 JOSeS Algorithm to Compute the Optimal Service Selection Optimizing the Global Utility

function `OptimalSolution() returns` (z)
`reset (1);` $k \leftarrow 1$; /* initialize activity pointers */
$s \leftarrow$ `AdvanceList (k);` $z \leftarrow s$; /* initialize solution */
$z_{\text{opt}} \leftarrow$ any allocation in $\mathcal{Z}$;
while $s \neq$ NULL **do**
  if $k < N$ **then**
    if $(\mathcal{L}(E[R(z)]) \leq R_{\max}) \wedge (A(z) \geq A_{\min}) \wedge (X(z) \geq X_{min}) \wedge (C(z) \leq C_{\max})$ **then**
      $k \leftarrow k + 1$
    else
      $z \leftarrow z\diamond$ /* remove last SP in z */
    end if
  else
    if $(E[R(z)] \leq R_{\max}) \wedge (A(z) \geq A_{\min}) \wedge (X(z) \geq X_{min}) \wedge (C(z) \leq C_{\max})$ **then**
      if $U(z) > U(z_{\text{opt}})$ **then**
        $z_{\text{opt}} \leftarrow z$
      end if
    end if
    $z \leftarrow z\diamond$ /* remove last SP in z */
  end if
  $s \leftarrow$ `AdvanceList (k);` $z \leftarrow z || s$
end while
return z_{opt}
end function

Heuristic Solution: HCB(1)

Algorithm 6 Identify Neighbors

function `neighbors` (z_0) `returns` ($\mathcal{Z}$)
  $\mathcal{Z} \leftarrow \emptyset$; /* Intialize with empty neighborhood */
  $\mathcal{N} \leftarrow \emptyset$; /* All neighbors */
  for all activity $i = 1, ..., N$ **do**
    for all $q_i \in \{R_i, A_i, X_i\}$ **do**
      if $q_i = R_i$ **then**
        /* s = best improvement in response time */
        $s = arg\ max_{k=1}^{|S_i|}\{1 - \frac{q_{i,k}}{q_{i,curr}}\}$;
      else
        /* s = best improvement in availability and throughput */
        $s = arg\ max_{k=1}^{|S_i|}\{\frac{q_{i,k}}{q_{i,curr}} - 1\}$;
      end if
      $z = replace\ (z_0, i, s)$; /* Replace current SP of a_i in z_0 by s */
      if $z \notin \mathcal{N}$ **then**
        $\mathcal{N} \leftarrow \mathcal{N} \bigcup z$;
        if $(((C(z) \leq C_{\max})$ and $(A(z) \geq A_{\min})$ and $(\mathcal{L}(E[R(z)]) \leq R_{\max})$ and $(X(z) \geq X_{\min})$)) **then**
          if $(E[R(z)] \leq R_{\max})$ **then**
            $\mathcal{Z} \leftarrow \mathcal{Z} \bigcup z$;
          end if
        end if
      end if
    end for
  end for
  return $\mathcal{Z}$;
end function

Heuristic Solution: HCB(2)

Algorithm 5 HCB Heuristic Algorithm

function HeuristicSolution () returns (z)
$nrestarts \leftarrow 0$;
while ($nrestarts < maxrestarts$) **do**
    $z_0 \leftarrow$ randomStart(); /* random start */
    $nrestarts \leftarrow nrestarts + 1$; $searching \leftarrow$ TRUE;
    while ($searching$) **do**
        $\mathcal{Z} \leftarrow$ neighbors (z_0); /* get feasible neighbors */
        $z_{opt} \leftarrow arg\ max_{z_i \in \mathcal{Z}}\{U(z_i)\}$; /* Identify neighbor with highest utility */
        if ($U(z_{opt}) > U(z_0)$) **then**
            $z_0 \leftarrow z_{opt}$;
        else
            $searching \leftarrow$ FALSE; /* local optimum */
        end if
    end while
    if ($nrestarts = 1$) **then**
        $z_{gopt} \leftarrow z_{opt}$;
    else if ($U(z_{opt}) > U(z_{gopt})$) **then**
        $z_{gopt} \leftarrow z_{opt}$;
    end if
end while
return z_{gopt};
end function

Experiment

- Aimed to evaluate the efficiency between the algorithms; solution space required and computation time by them; and compare them based on other parameters such complexity of the BPT and SPs per activity.
- 50 BPEL business processes were randomly generated, which contained 6 to 9 activities and had constructs such as sequence, flow, and switch-case. A total of 36000 runs were made.
- The calculations were made using a 95% confidence interval.

Results: Utilization ratio comparison

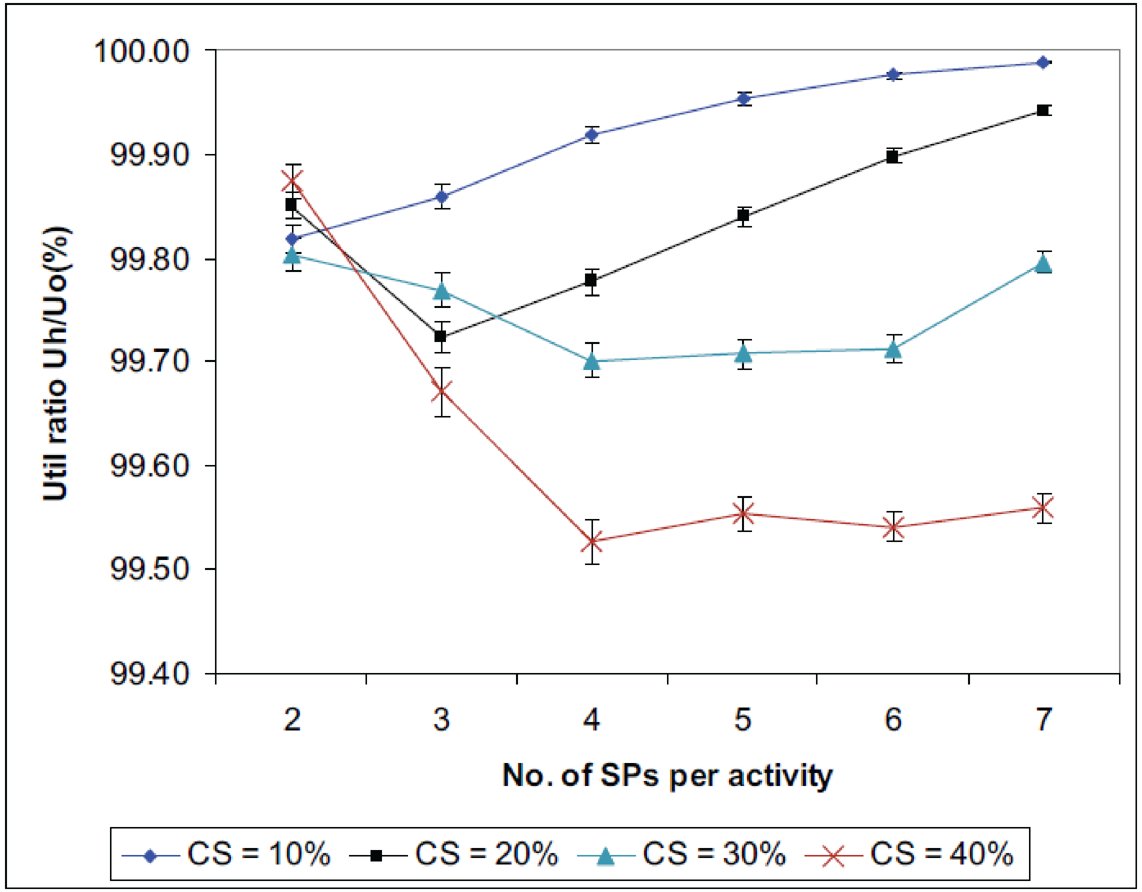

Figure 1. Average U_h/U_o (%) vs. $nspa$ for four constraint strengths

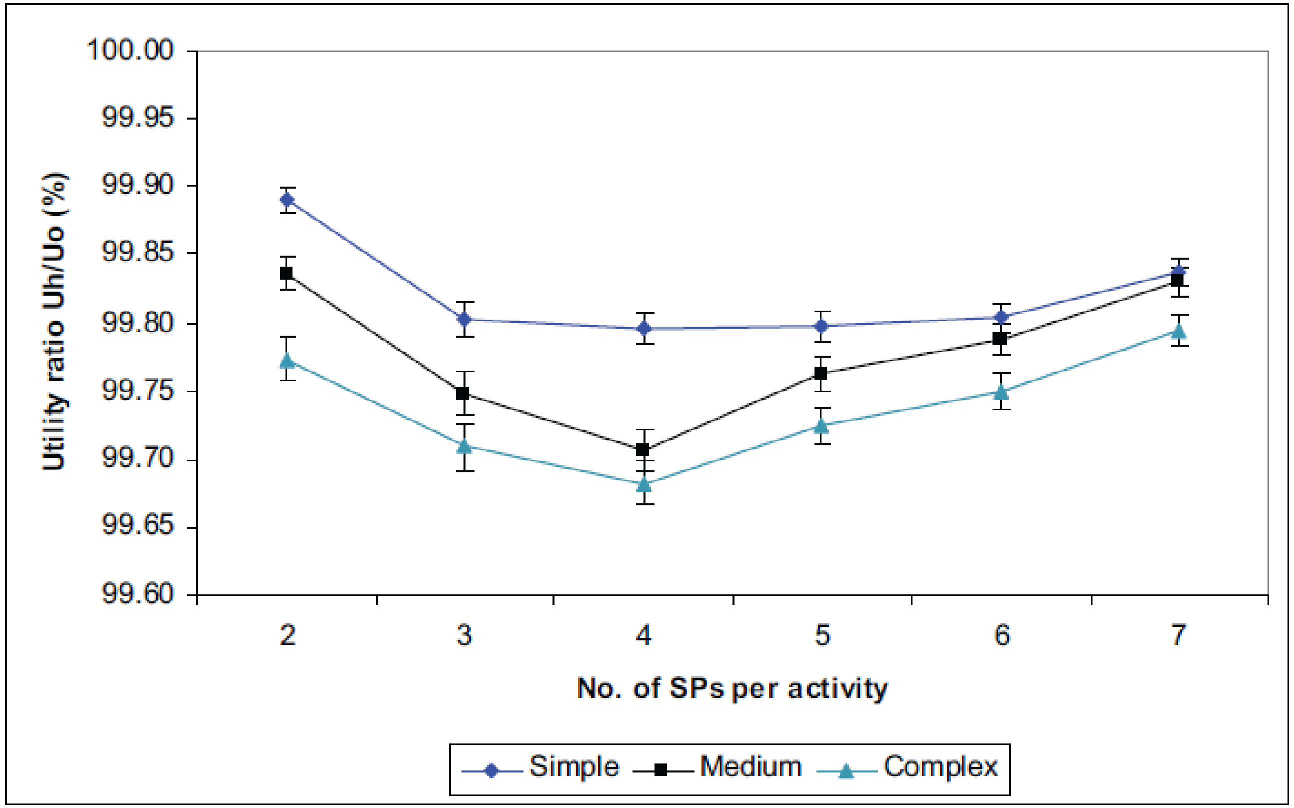

Figure 4. Average U_h/U_o (%) vs. $nspa$ for simple, medium, and complex business processes

Results: Number of points examined comparison

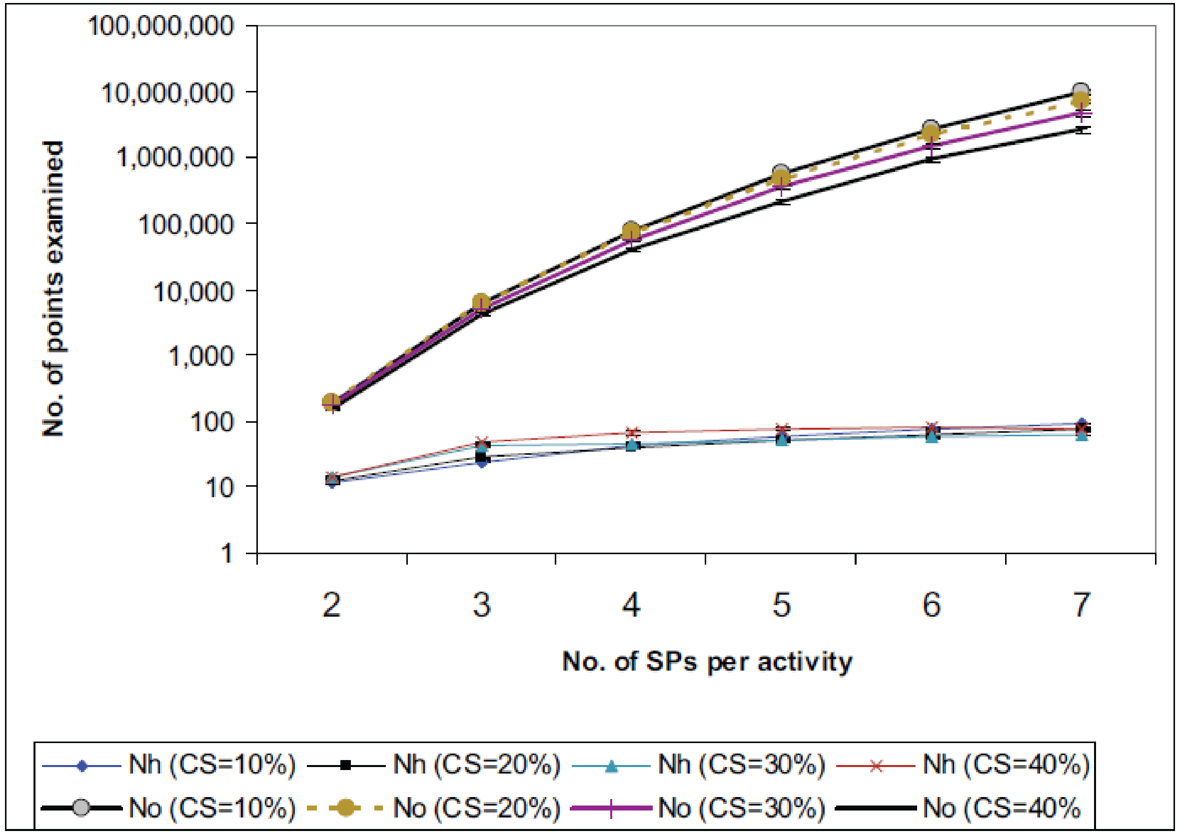

Figure 2. Average number of points examined N_h and N_o vs. $nspa$ for four constraint strengths

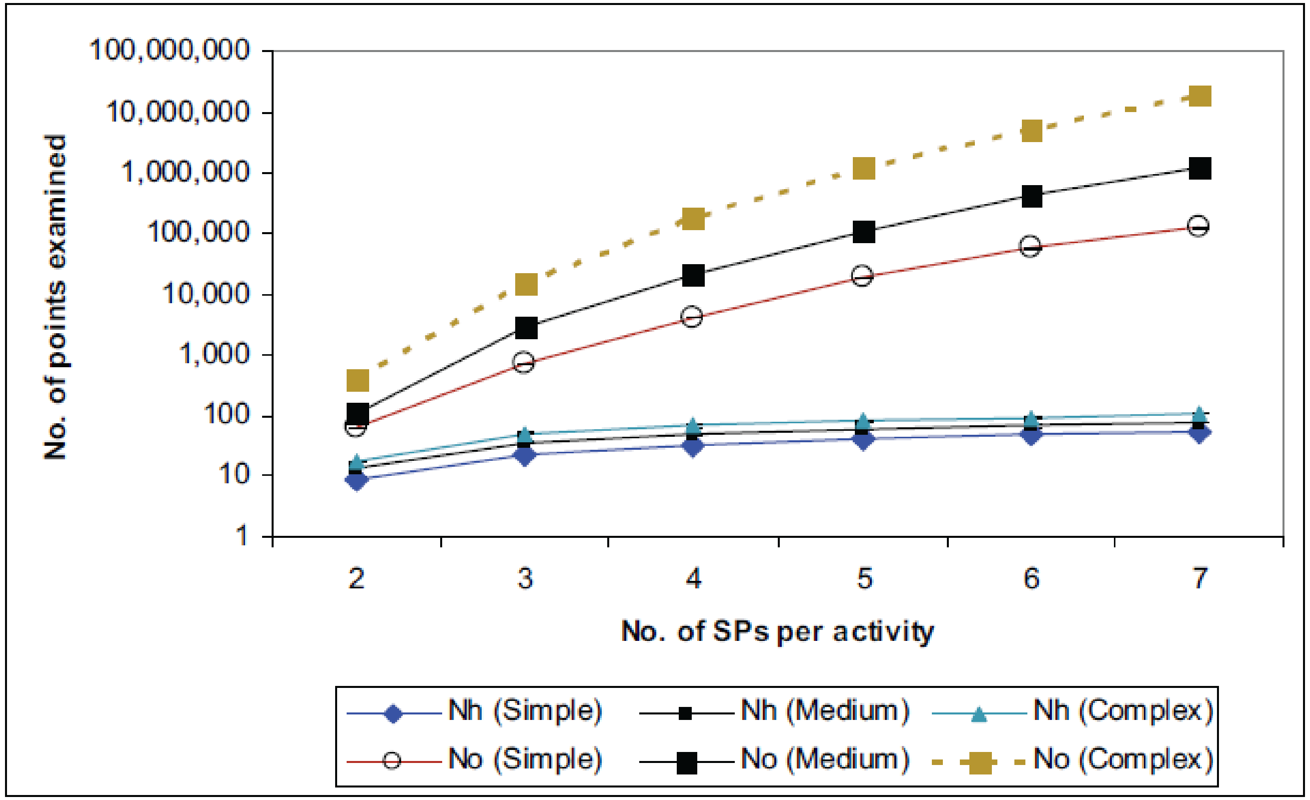

Figure 5. Average number of points examined N_h and N_o vs. $nspa$ for simple, medium, and complex business processes

Results: Computing time comparison

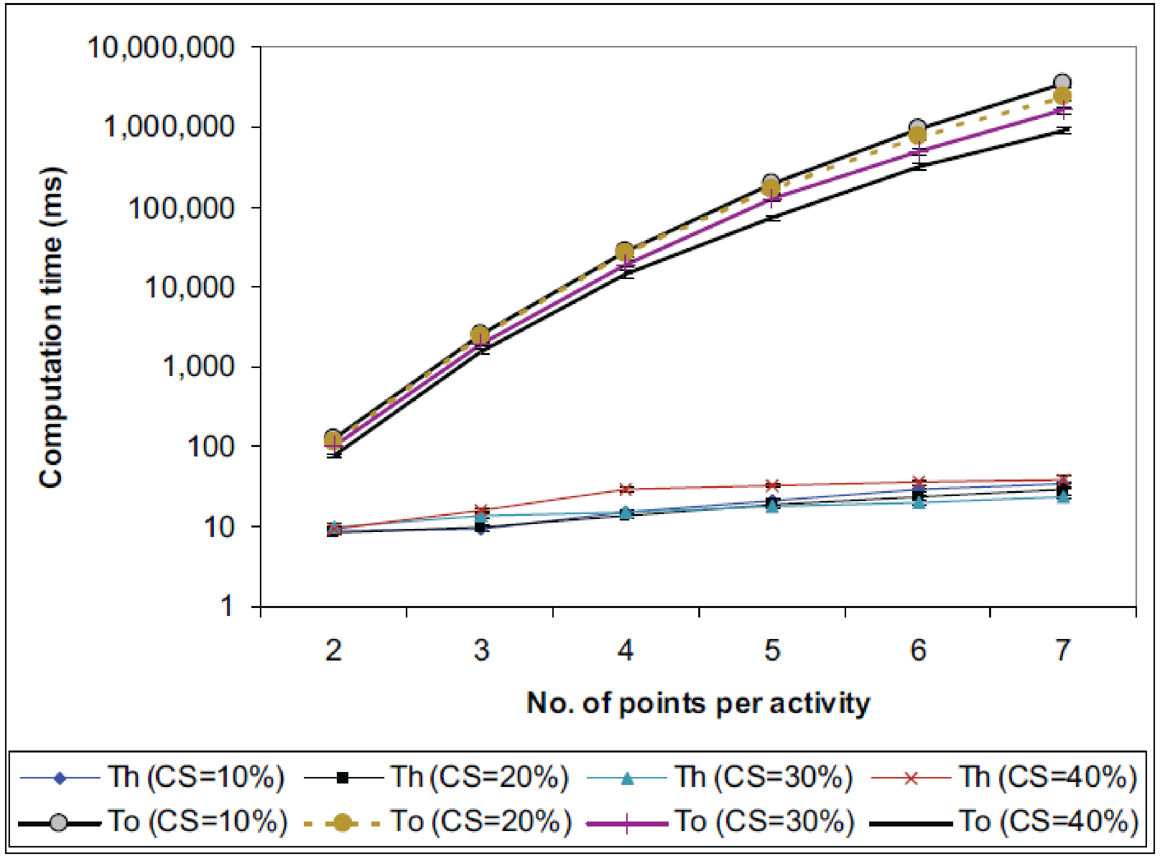

Figure 3. Average computation time T_h and T_o vs. $nspa$ for four constraint strengths

Results: Analysis of the Nh visited points growth against SPs per activity

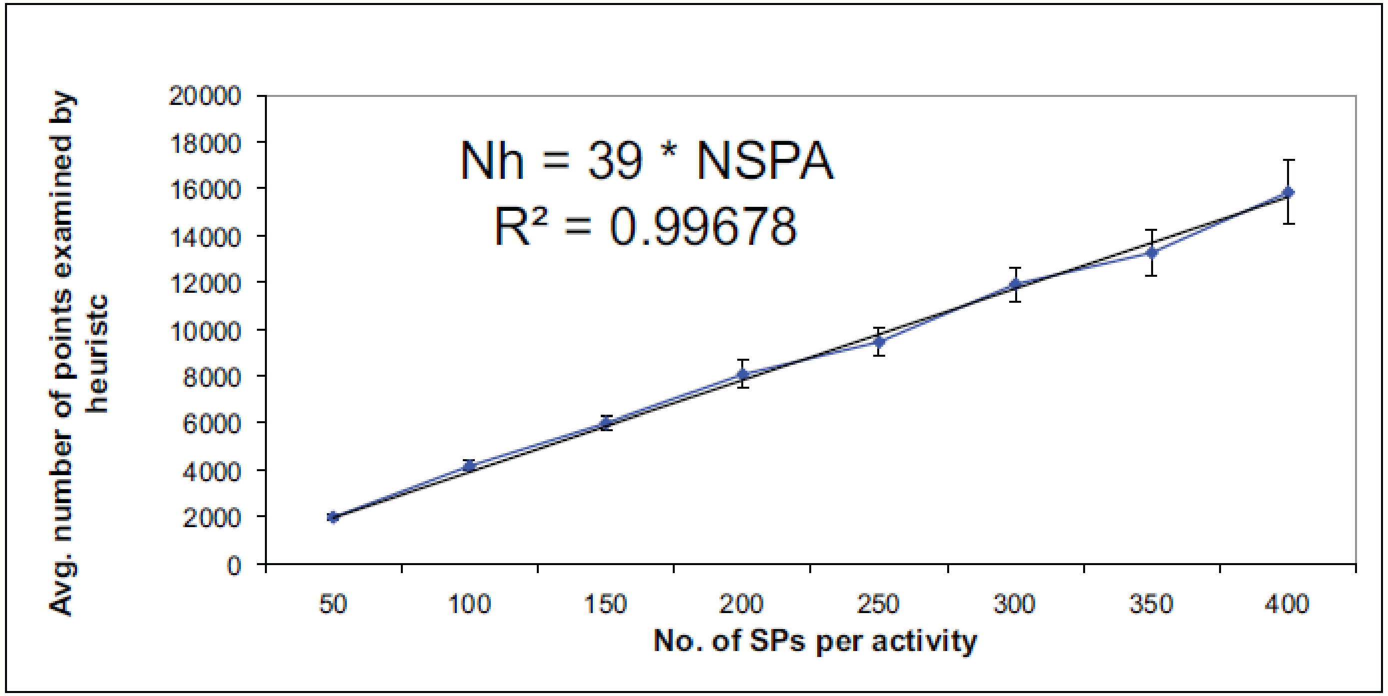

Figure 6. Average N_h vs. $nspa$

Discussion

- Has HCB solution runtime limitations?
- What is next step after HCB?

Thank you for your time!