SAT Solvers for the Problem of Sensor Placement

Anna Gorbenko
Department of Intelligent Systems and Robotics
Ural Federal University
620083 Ekaterinburg, Russia
gorbenko.ann@gmail.com

Vladimir Popov
Department of Intelligent Systems and Robotics
Ural Federal University
620083 Ekaterinburg, Russia
Vladimir.Popov@usu.ru

Abstract

In this paper we consider an approach to solve the problem of sensor placement. This approach is based on constructing SAT solvers for logical models of the problem.

PACS: 42.30.Tz

Keywords: sensor placement, logical models, SAT solvers, 3SAT

Many different formalizations of the problem of sensor placement received a lot of attention recently (see e.g. [1, 2]). Note that sensor placement is extensively used for improved robotic navigation (see e.g. [3, 4, 5]). In particular, visual landmarks problems are extensively studied in contemporary robotics (see e.g. [6, 7, 8, 9]). In this paper we consider SAT solvers for logical models of SP problem (see [2]).

In papers [10, 11, 12, 13, 14] the authors considered some algorithms to solve logical models (see also [15, 16, 17]). Also, we have obtained explicit reductions from SP to MAXSAT, SAT and 3SAT (see [2, 18]).

We use algorithms fgrasp and posit from [19]. Also we design our own genetic algorithm for SAT which based on algorithms from [19].

Consider a boolean function

\[ g(x_1, x_2, \ldots, x_n) = \bigwedge_{i=1}^{m} C_i, \]

where \( m \geq 1 \), and each of the \( C_i \) is the disjunction of one or more literals. Let \(|C_i|\) be a number of literals in \( C_i \). Let \( \text{occ}(x_i, g) \) be a number of occurrences of \( x_i \) in \( g \). Respectively,
let \( \text{occ}(\neg x_i, g) \) be a number of occurrences of \( x_i \) in \( g \). For example, if \( g = (x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land (x_1 \lor x_4) \land (\neg x_1 \lor x_5) \), then \( \text{occ}(x_1, g) = 2, \text{occ}(\neg x_1, g) = 1 \).

We consider a number of natural principles that define importance of a variable \( x_i \) for satisfiability of boolean function \( g \). These principles suggest us correct values of variables.

1. If \( \text{occ}(x_i, g) \geq 0 \) and \( \text{occ}(\neg x_i, g) = 0 \), then \( x_i = 1 \).
2. If \( \text{occ}(x_i, g) = 0 \) and \( \text{occ}(\neg x_i, g) \geq 0 \), then \( x_i = 0 \).
3. If \( x_i = C_j \) for some \( j \), then \( x_i = 1 \).
4. Given positive integers \( p_1, p_2, \ldots, p_m, q_1, q_2, \ldots, q_m \) and a set of rational numbers \( \{ \alpha_{i,u}, \beta_{i,v} \mid 1 \leq i \leq m, 1 \leq u \leq p_i, 1 \leq v \leq q_i \} \). If

\[
\sum_{1 \leq j \leq m, 1 \leq u \leq p_j, |C_j| = u} \alpha_{j,u} \text{occ}(x_i, C_j) \geq \sum_{1 \leq j \leq m, 1 \leq v \leq q_j, |C_j| = v} \beta_{j,v} \text{occ}(\neg x_i, C_j),
\]

then \( x_i = 1 \).

Based on these principles, we can consider the following four types of commands: \( P_1, P_2, P_3, P_4 \). Also we consider the following three commands for run algorithms: \( \text{Try\_fgrasp}, \text{Try\_posit}, \text{Try\_ga} \), where \( \text{Try\_ga} \) runs a simple genetic algorithm.

Denote by \( \mathcal{R} \) the set of commands of these seven types. Arbitrary element of \( \mathcal{R}^* \) it is possible to consider as a program for finding values of variables of a boolean function. We assume that such programs are executed on a cluster.

Execution of each of commands of type \( P_i \) reduces the number of variables of a boolean function by one. Execution of each of commands \( \text{Try\_fgrasp}, \text{Try\_posit}, \text{Try\_ga} \) consists in the run of corresponding algorithm for current boolean function on a separate set of calculation nodes and the transition to the next command.

Algorithms fgrasp and posit we run only on one calculation node. Genetic algorithms can be used in parallel execution. We use auxiliary genetic algorithm which determine the number of calculation nodes.

Initially, we selected a random subset of \( \mathcal{R}^* \). We use a genetic algorithm to select a program from the current subset of \( \mathcal{R}^* \) and a genetic algorithm for evolving the current subset of \( \mathcal{R}^* \). The evolution of the current subset of \( \mathcal{R}^* \) implemented on a separate set of calculation nodes. For every subsequent boolean functions it is used the current subset of \( \mathcal{R}^* \) which is obtained by taking into account the results of previous runs.

We use heterogeneous cluster based on three clusters (Cluster USU, Linux, 8 calculation nodes; umt, Linux, 256 calculation nodes; um64, Linux, 124 calculation nodes) [20].
SAT solvers for the problem of sensor placement

Algorithms fgrasp and posit used only for 3SAT. For SAT and MAXSAT used simple genetic algorithm (SGA), and our algorithm (OA). We create a generator of natural instances for SP. We consider instances with $N$ and $S$ from 400 to 600. Selected experimental results are given in Figures 1 – 2.

<table>
<thead>
<tr>
<th>time</th>
<th>fgrasp</th>
<th>posit</th>
<th>SGA</th>
<th>OA</th>
</tr>
</thead>
<tbody>
<tr>
<td>average</td>
<td>56 min</td>
<td>1.03 h</td>
<td>1.26 h</td>
<td>27.38 min</td>
</tr>
<tr>
<td>max</td>
<td>18.43 h</td>
<td>16.67 h</td>
<td>28.25 h</td>
<td>13.59 h</td>
</tr>
<tr>
<td>best</td>
<td>3.52 min</td>
<td>5.12 min</td>
<td>3.32 min</td>
<td>28 sec</td>
</tr>
</tbody>
</table>

Figure 1: Experimental results for 3SAT

<table>
<thead>
<tr>
<th>time</th>
<th>SGA</th>
<th>OA</th>
<th>SGA</th>
<th>OA</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>1.17 h</td>
<td>53.12 min</td>
<td>1.14 h</td>
<td>49.29 min</td>
</tr>
<tr>
<td>SAT</td>
<td>26.29 h</td>
<td>17.33 h</td>
<td>32.84 h</td>
<td>25.17 h</td>
</tr>
<tr>
<td>MAXSAT</td>
<td>3.18 min</td>
<td>16 sec</td>
<td>1.09 min</td>
<td>12 sec</td>
</tr>
<tr>
<td>MAXSAT</td>
<td>3.52 min</td>
<td>5.12 min</td>
<td>3.32 min</td>
<td>28 sec</td>
</tr>
</tbody>
</table>

Figure 2: Experimental results for SAT and MAXSAT

References


*Received: September, 2012*