Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Planning with Global State Constraints and State-Dependent Action Costs

Franc Ivankovic,[1] **Dan Gordon,**[2] **Patrik Haslum**[1,2]
[1]CSIRO Data61 [2]Australian National University
Canberra, Australia
franc.ivankovic@data61.csiro.au, dan.gordon@anu.edu.au, patrik.haslum@anu.edu.au

Abstract

Planning with global state constraints is an extension of classical planning in which some properties of each state are derived via a set of equations, rules or constraints. This extension enables more elegant modelling of networked physical systems such as power grids. So far, research in this setting focused on domains where action costs are constant, rather than a function of a state in which the action is applied. This limitation prevents us from accurately specifying the objective in some real-world domains, leading to generation of suboptimal plans. For example, when reconfiguring a power network, we often need to temporarily leave some users without electricity for a certain amount of time, and in such circumstances it is desirable to reduce the unsupplied load over the total time span. This preference can be expressed using state-dependent action costs. We extend planning with global state constraints to include state-dependent action costs, adapt abstraction heuristics to this setting, and show improved performance on a set of problems.

Motivation

In classical planning, state variables are assigned values in the initial state and remain unchanged until explicitly modified by action effects. However, it is sometimes more natural to model some properties of states as derived, in each state, via a set of rules that we call state constraints. These can take a variety of forms, for example, logical axioms (Thiébaux, Hoffmann, and Nebel 2005) or numeric equations and inequalities (Ivankovic et al. 2014; Haslum et al. 2018). The latter is suited for domains that involve interconnected physical systems, such as power grids, transport systems or water networks. In such domains a single discrete action can change the state of the network globally in a way that depends on the entire current configuration (e.g., opening/closing a switch may change power flow in all lines).

Most work on optimal planning has focused on the case of constant action costs, but recently, there has been interest in allowing for action costs to be a function of the state in which they are applied (Geißer, Keller, and Mattmüller 2015; 2016). This is called state-dependent action costs (SDAC). This work, so far, has dealt with classical planning without state constraints.

Introducing SDAC to planning with state constraints enables us to more accurately represent certain objectives in some real-world domains. An example which we will use in this paper is power network reconfiguration. This problem requires finding a sequence of switching actions to change the network configuration into one where the maximum possible load is supplied. We may need to temporarily leave portions of the network without power and it is preferable that this unsupplied load is minimized at every time step (Thiébaux et al. 2013). This can be expressed by assigning a cost to each switching action equal to this value. The cost of the entire plan is the sum of the unsupplied loads over all steps. In this example, action costs are determined by the current network topology, and therefore could be formulated with conditional effects in a classical formalism (albeit exponentially many). This is, however, not always the case: An example is if we want to minimise line losses, which are a function of the power flow. The power flow is constrained, but not uniquely determined, by the network topology.

In this paper we build on our previous framework for planning with numeric state constraints (Haslum et al. 2018), to which we add the capability to express state-dependent action costs. We also adapt the pattern database heuristic to this setting. We apply our approach to SDAC versions of the power network reconfiguration and the hydraulic blocks world domains.

Related Work

State constraints were used in early work on planning to concisely represent actions, but have mostly been absent in more recent research, with a few exceptions. PDDL2.2 included derived predicates and axioms (Thiébaux, Hoffmann, and Nebel 2005), which enable encoding of Boolean constraints, but, unfortunately, was supported only by a few planners. Some domain-specific planners were created to deal with physically interconnected domains, where a single (local) action affects the state of the entire network. Examples include the works of Aylett et al. (1998) on planning chemical plant managment, and Piacentini et al. (2013), who address voltage balancing in a power network. Vallati et al. (2016) use a planner to manage traffic flow in an urban environment, by giving it control of traffic lights.

Formulating problems with state-dependent action costs in unrestricted numeric PDDL (Fox and Long 2003) is

straightforward, since the problem metric can be any fluent expression and actions can have arbitrarily complex and conditional effects on numeric fluents. However, deriving informative heuristics for such general metrics is difficult, and has received relatively little attention. Recently, Geißer et al. (2015; 2016) adapted the additive heuristic h^{add} (Bonet, Loerincs, and Geffner 1997) (which is inadmissible) as well as an admissible abstraction heuristic to SDAC planning. Central to their approach is representing cost functions as edge-valued multi-valued decision diagrams (EVMDDs) (Lai, Pedram, and Vrudhula 1996; Ciardo and Siminiceanu 2002) and using them to compute costs in relaxed and abstract states.

Planning Formalism

We briefly summarise the formalism for planning with state constraints (Haslum et al. 2018). We then extend it with state-dependent action costs.

State variables are divided into *primary*, V_P, and *secondary*, V_S, with $V_P \cap V_S = \emptyset$. Primary variables function the same way as in classical planning – their domains are finite, their values are assigned by action effects and persist unless reassigned. A *state* is a full valuation over primary variables. Values of secondary variables (*extended state*) are freely chosen, subject to state constraints. State constraints involve both primary and secondary variables and are the way in which the two interact. These can take a number of different forms (e.g. axioms) but the type that we use here is *switched constraints*.

Definition 1. Switched constraints *are of the form $\varphi \rightarrow \gamma$, where φ is a conjunctive formula over primary variables, and γ is a constraint over the secondary variables. In a state in which φ is true, we say that the switched constraint is* active.

While the formalism is general, in the domains we use in this paper the consequents ("γ-parts") of switched constraints are numeric equations and inequalities. Given a state s and a set of constraints C, there is a subset of C that is active. The set of all consequents of constraints in C active in s is denoted $\mathrm{active}(C, s)$. There can be a unique assignment, multiple possible assignments, or no assignment to the secondary variables that satisfies this set of constraints.

Switched constraints appear as secondary parts of *partitioned conditions* (see below), and in a set of *invariant* constraints C_{inv}, which are a part of the problem description and must be satisfied for a state to be *valid* – i.e. if $\mathrm{active}(C_{\mathsf{inv}}, s)$ is unsatisfiable, s is *invalid* and cannot be visited by a plan. This means that to apply action a in state s, it is required that (i) the precondition of a holds in s and (ii) the resulting state is valid.

Definition 2. *A* partitioned condition *is a pair $\langle c_P, c_S\rangle$, where c_P is a formula over primary variables and c_S a set of switched constraints. $\langle c_P, c_S\rangle$* holds *in state s iff c_P is true in s and $\mathrm{active}(c_S \cup C_{\mathsf{inv}}, s)$ is satisfiable.*

Partitioned conditions can appear as action preconditions and in the problem goal. In our two example domains, actions have no secondary preconditions, but the secondary variables feature in the goal condition.

Definition 3. *A* planning problem *P consists of: a set V_P of primary variables, with each variable $v \in V_P$ having a finite domain $D(v)$ of values; a set V_S of secondary variables; a set C_{inv} of invariant switched constraints; an initial state s_0, assigning values to all variables in V_P, such that s_0 is valid; a partitioned goal condition $\langle G_P, G_S\rangle$; and a set A of actions. Each action $a \in A$ is defined by: a partitioned precondition $\mathrm{pre}(a) = \langle \mathrm{pre}_P(a), \mathrm{pre}_S(a)\rangle$; an effect $\mathrm{eff}(a)$, which is a set of assignments to primary variables; and a cost function $c_a(s_e)$, where s_e is an extended state, and which returns a real number.*

An action sequence $\pi = \langle a_1, \ldots, a_n\rangle$ induces a corresponding state sequence $\langle s_0, s_1, \ldots, s_n\rangle$, where each state s_i is obtained by applying a_i to the state s_{i-1}. π is a plan if each state in the sequence is valid, each action a_i is applicable in s_{i-1} and s_n satisfies G.

While applicability of an action depends only on the state (i.e. valuation over the primary variables), we the cost function may depend on both primary and secondary variables. The values of the primary variables are fixed in the state, but values of the secondary variables can be chosen freely subject to the active constraints. It is reasonable to define the cost of applying an action in a state as the minimum cost over all extended states consistent with the active constraints in $\mathrm{pre}_S(a) \cup C_{\mathsf{inv}}$. That is, the cost is the solution to

$$\min c_a \text{ subject to } \mathrm{active}(\mathrm{pre}_S(a) \cup C_{\mathsf{inv}}, s). \qquad (1)$$

The complexity of finding this value depends on the form of the cost function[1] and the constraints.

In the two example domains we use in this paper, c_a is a linear expression over secondary variables, and, moreover, the secondary variables that appear in the cost function are uniquely determined in every state, so calculating SDAC in a given state simply consists of summing up the terms in c_a. This is not true in general, however: In the example of minimising line losses mentioned earlier, the cost may depend also on the output of each generator (a secondary variable), which is constrained but not determined by the primary state. The power flow constraints are non-linear, making (1) a non-linear problem. Solving the minimisation problem is also relevant when computing heuristics, as we will see below, because in a relaxed or abstract state the values of primary variables are not fixed.

Domain Examples

We extend two domains with state constraints that we previously introduced (Ivankovic et al. 2014; Haslum et al. 2018) by adding SDAC.

Power Supply Restoration (PSR): The problem consists of reconfiguring a power network by opening and closing

[1] We previously defined conditional action cost as a list of pairs (φ, c) where φ is a partitioned condition and c_a is a positive constant, and the cost of a in s is $\sum\{c_i \mid (\varphi_i, c_i) \in \mathrm{cost}(a), s(\varphi_i) = \mathit{true}$ (Ivankovic et al. 2014). This means that the values of secondary variables must be chosen such that the fewest conditions φ_i weighted by their corresponding costs c_i are satisfied, which is an NP-hard problem (Ivankovic 2018).

switches to restore power to as many consumers as possible after an outage. State constraints, in the form of non-linear equations, are used to compute the power flow and ensure that no safety constraints are violated. In our previous formulation of this domain, the objective was only to mimise plan length. Here, we make the cost of every action $c_a = \sum_i \bar{p}_i(1 - f_i)$, where $\bar{p}_i$ is the (constant) load at bus i and f_i a secondary variable that is 1 if bus i is fed and 0 if it is not. In other words, the action's cost equals the sum of unsupplied loads.

In our previous version, the goal is to supply a specified set of buses, which is precomputed from a feasible network configuration that supplies a maximum amount of load. We refer to this variant as the *fixed supply goal*. However, we can leave the problem of finding the configuration that maximizes the supplied load, as well as the switching plan to reach it, to the planner. We introduce a *goal* primary variable whose initial value is *false*, and an *end* action which changes it to *true* and after which no other action can be applied. The cost of the *end* action is the sum of unsupplied loads in the end state multiplied by a large constant. We refer to this variant as the *maximum supply goal*.

Hydraulic Blocks World (HBW): This domain is an extension of Blocks World (Winograd 1972) where each block has a weight, and each tower sits on a piston in a cylinder rising from a reservoir of hydraulic fluid. The level of fluid in each of the cylinders (and thus the height of the piston) is determined by the distribution of weights over all pistons and is not allowed outside specified limits. In our SDAC version, the cost of picking up or putting down a block equals the height of the tower involved.

Adapting Heuristics

An *abstraction* maps states of a planning problem into a smaller state space, such that optimal plan cost in the abstract state space is a lower bound on plan cost in the original state space. Hence, this value can be used as an admissible heuristic cost estimate in search algorithms like A*. A *projection* is an abstraction in which all but a designated subset of variables, called the *pattern*, are ignored. These variables are removed from action preconditions and effects, the initial state and the goal. The resulting heuristic is called a *pattern database* or PDB (Culberson and Schaeffer 1998; Edelkamp 2001).

In PDBs for planning with state constraints, variables in the pattern are primary variables. Adapting the abstraction heuristic to this setting, there are two key questions: (i) how to determine which switched constraints are active in an abstract state, and, consequently, the truth value of a partitioned condition; and (ii) how to compute abstract state-dependent action costs. We look at this in the following way (Haslum et al. 2018): Given a pattern A, every abstract state s^A represents a set of states, namely those that can be obtained by assigning each variable not in the pattern one of the values from its domain: $\mathrm{states}(s^A) = \{\{x_1 = v_1, \ldots, x_n = v_n\} \mid v_i = s^A(x_i) \text{ if } x_i \in A;\ \text{else } v_i \in D(x_i)\}$.

Like individual primary variables, a formula ϕ over the primary variables assumes a set of values $s^A(\phi)$ in an abstract state: $s^A(\varphi) = \{s(\varphi) \mid s \in \mathrm{states}(s^A)\}$. Therefore *true* $\in s^A(\phi)$ iff there exists at least one state in the s^A where ϕ is true (and analogously for *false*). We consider a switched constraint with a trigger ϕ to be active in an abstract state s^A iff *false* $\notin s^A(\phi)$. In other words, the constraint active only if the trigger holds in all states that map to s^A. It follows that the set of active constraints in s^A, $\mathrm{active}^A(C, s^A)$, is the intersection of $\mathrm{active}(C, s_i)$ for all $s_i \in \mathrm{states}(s^A)$.

Definition 4. *A partitioned condition $\langle c_P, c_S \rangle$ holds in an abstract state s^A iff (i) true $\in s^A(c_p)$ and (ii) $\mathrm{active}^A(c_S \cup C_{\mathsf{inv}}, s^A)$ is satisfiable.*

Evaluating the state-dependent cost of an action in an abstract state is straightforward – the objective function remains the same expression over secondary variables as in the concrete problem, but the constraint set reflects the fact that we are in an abstract state. The problem becomes:

$$\min c_a \text{ subject to } \mathrm{active}^A(\mathrm{pre}_S(a) \cup C_{\mathsf{inv}}, s^A) \qquad (2)$$

The resulting abstraction heuristic is admissible provided that the constraint solver is sound in the sense that it will only declare a constraint set unsatisfiable if it has no solution, and that the cost minimisation problem in Eq. 2 is solved optimally or lower-bounded. To see this, note that $\mathrm{active}^A(C, s^A) \subseteq \mathrm{active}(C, s)$ for any set of switched constraints C and $s \in \mathrm{states}(s^A)$. Thus, a path through valid states in the concrete space is mapped to a path through valid abstract states, with each action on the path having equal or lower cost.

In our example domains, the values of variables in the cost function are fully determined in the concrete problem. In the PSR domain, the "fed"-status of each bus is $f_i \in [0, 1]$ and constraints ensure that there is exactly one assignment given the state of the switches. In an abstraction, however, some of those constraints may be inactive, making some of the "fed" variables under-constrained. Hence, loads may be partially fed and we need to solve the minimisation problem (2). The PSR domain has non-linear constraints, so Eq. (2) becomes a non-linear programming problem. We solve it using the same solver, based on the SmartGrid Toolbox, that we used to determine satisfiability of the active constraints in the unit-cost setting. We did not observe any noticeable increase in runtime from solving the minimisation problem instead of only satisfiability.

To build PDBs we follow the same method as in previous work (Ivankovic and Haslum 2015; Haslum et al. 2018), building the reachable abstract space, which includes determining the cost of each action in each reachable abstract state, and then computing optimal plan costs over this graph. Once the PDBs are built, state evaluation is done by table lookup, same as in the classical setting.

Experiments

We present: (i) a comparison between blind search and search with the (above described) abstraction heuristic; (ii) a comparison between the state-dependent cost of shortest plans and the SDAC-optimal plan; and (iii) a comparison of

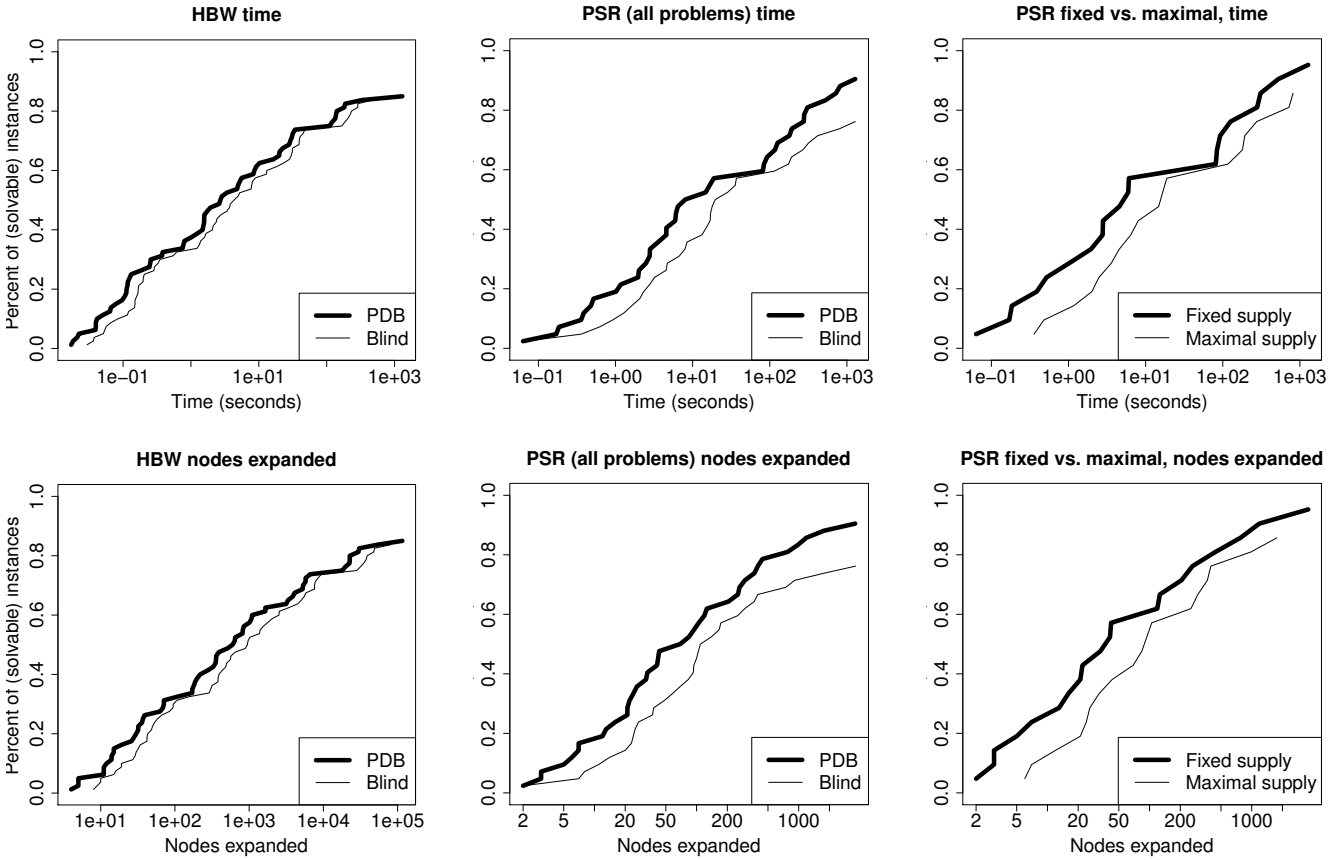

Figure 1: Cumulative distributions of runtime (top row) and number of nodes expanded (bottom row). The four figures on the left compare blind search and search with a PDB heuristic. HBW is in the left column, while PSR is in the middle (both instances with fixed supply and instances with maximum supply goal). The column on the right compares the performance of PDB on PSR problems with fixed supply and maximum supply goals.

the hardness of PSR problems with fixed supply and maximum supply goals. All experiments used the A* search algorithm and an 1800 second time limit per problem.

The abstraction heuristic reduces both search time and the number of states expanded, compared to blind search (see Figure 1). **HBW:** We used a set of 80 problems, 11 having no solution (which is shown within a second by either configuration). With the PDB heuristic, the planner solved 68 of the remaining problems within the time limit, while blind search solved one less. (One instance was not solved by either configuration.) On average, blind search expands 1.63 times more nodes and takes 1.60 times longer. **PSR:** We used a set of 21 problems, each run with the fixed supply and maximum supply goal formulation, for a total of 42 instances. The planner with the PDB heuristic solves 38 problems within the time limit, while blind search solves 32. Averaged over problems solved by both, blind search expands 2.34 times as many nodes and takes 2.88 times longer.

Our previous planner (Ivankovic et al. 2014; Haslum et al. 2018) was able to generate shortest plans to a feasible configuration that supplies a fixed set of buses, ignoring the amount of load supplied at each time step. In our problem set (21 problems, same as above), blind search finds the shortest plan for 19 instances within the time limit. For each of those problems we compared SDAC costs (i.e. sum of the unsupplied loads at each time step) of shortest plans with the cost of the SDAC-optimal solutions. We found that for problems requiring more complex plans (more than 4 actions long), the shortest plan does not have the optimal cost when considering the total load supplied over time objective. In those instances (9 out of 19 problems), the shortest plans are on average 14% more expensive than optimal. The SDAC-aware planner on average takes 25% more time and expands 29% less nodes (when both the SDAC-aware and shortest-length planner use blind search).

As expected, problems with the maximum supply goal are harder. In our problem set, blind search solves 19 problems with fixed supply goal, and 13 with the maximum supply goal. Search with the PDB heuristic solves 20 fixed supply goal problems and 18 maximum supply goal problems. On average, solving for the maximum supply goal takes 4.04 times longer and results in 2.82 more nodes expanded with blind search, and 3.87 times longer and 2.93 times more nodes expanded with the PDB heuristic.

References

Aylett, R.; Soutter, J. K.; Petley, G. J.; and Chung, P. W. H. 1998. AI planning in a chemical plant domain. In *European Conference on AI*, 622–626.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, 714–719.

Ciardo, G., and Siminiceanu, R. 2002. Using edge-valued decision diagrams for symbolic generation of shortest paths. In *Formal Methods in Computer-Aided Design, 4th International Conference, FMCAD 2002, Portland, OR, USA, November 6-8, 2002, Proceedings*, 256–273.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. 6th European Conference on Planning (ECP)*, 13–24.

Fox, M., and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of AI Research* 20:61–124.

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete relaxations for planning with state-dependent action costs. In *Proc. 24th International Joint Conference on AI (IJCAI)*, 1573–1579.

Geißer, F.; Keller, T.; and Mattmüller, R. 2016. Abstractions for planning with state-dependent action costs. In *Proc. 26th International Conference on Automated Planning and Scheduling (ICAPS)*, 140–148.

Haslum, P.; Ivankovic, F.; Ramirez, M.; Gordon, D.; Thiébaux, S.; Shivashankar, V.; and Nau, D. S. 2018. Extending classical planning with state constraints: Heuristics and search for optimal planning. *Journal of Artificial Intelligence Research* 62:373–431.

Ivankovic, F., and Haslum, P. 2015. Optimal planning with axioms. In *Proc. of the 24th International Joint Conference on Artificial Intelligence*, 1580–1586.

Ivankovic, F.; Haslum, P.; Thiébaux, S.; Shivashankar, V.; and Nau, D. S. 2014. Optimal planning with global numerical state constraints. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014*.

Ivankovic, F. 2018. *Optimal Planning with State Constraints*. Ph.D. Dissertation, ANU College of Engineering and Computer Science, The Australian National University.

Lai, Y.; Pedram, M.; and Vrudhula, S. B. K. 1996. Formal verification using edge-valued binary decision diagrams. *IEEE Trans. Computers* 45(2):247–255.

Piacentini, C.; Alimisis, V.; Fox, M.; and Long, D. 2013. Combining a temporal planner with an external solver for the power balancing problem in an electricity network. In *Proc. 23rd International Conference on Automated Planning and Scheduling (ICAPS)*.

Thiébaux, S.; Coffrin, C.; Hijazi, H.; and Slaney, J. K. 2013. Planning with MIP for supply restoration in power distribution systems. In *Proc. 23rd International Joint Conference on Artificial Intelligence (IJCAI)*.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1-2):38–69.

Vallati, M.; Magazzeni, D.; De Schutter, B.; Chrpa, L.; and McCluskey, T. L. 2016. Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In *Proc. 30th AAAI Conference on Artificial Intelligence*, 3188–3194.

Winograd, T. 1972. *Understanding Natural Language*. Orlando, FL, USA: Academic Press, Inc.