1 Structural Patterns Heuristics via Fork Decomposition

1.1 Summary

One of the best heuristics developed over the last twenty-five years is to measure the amount of work needed to solve a simpler version of the original problem. There is a fundamental tension in this scheme between the effort it takes to solve the simpler version and the informativeness of measuring that effort, with respect to the original problem. Typically, the simpler problems are generated from the original by ignoring some of the variables in the original problem. As we ignore more variables, the problem becomes easier to solve but less informative about the original problem. The paper proposes a method of choosing variables to ignore which would permit larger (and therefore more informative) sets of variables while still guaranteeing that the generated problem is solvable. The paper uses the $SAS+$ problem representation and makes the claims that, if variables can be found with small domains, then substructures (particularly, a node and all of the children nodes fanning out of it, or a node and all of its predecessors) of the causal graph can be extracted that are solvable in polynomial time. The paper concludes with a theoretical analysis of tightness of the lower bound which I didn't really follow.

1.2 Significance

This paper extends the previous papers we discussed about additive heuristics and pattern databases. In that previous work, the actions had to be completely disjoint so that, when we summed the heuristic cost given by any database, we could be assured that no action cost was being double counted. In this paper, that constraint is relaxed that so that the cost for an action is split across the various databases. When we sum up the action costs across all the databases, we still don't exceed the original action cost. That is an interesting generalization and would seem to allow a lot of alternative additive formulations.

The authors also make the following interesting observation on bounds of the size of simplified domains. Since we wish to compute heuristics in polynomial time, and computing a planning problems is exponential in either the number of variable values, this means that the size of suitable simplified problems is logarithmic in the size of the original problem (unless we exploit problem structure). It is easy to believe that logarithmic heuristic functions would be weak predictors of the difficulty of the full problem, so the search for problem structure to exploit and how to exploit it seems very important.

ignoring delete lists has worked well as a heuristic in classical domains, there is a discussion of how to generate a numeric heuristic that would work analogously. In some sense, ignoring-deletes is a particular abstraction that guarantees the property that the abstracted problem can be solved in polynomial time; it's not clear why we'd need to copy the ignoring-deletes for numeric problems, if there is some other approximation that guarantees polynomial evaluation. In any case, this relaxation is achieved by mapping the original problem into a pure maximization problem, and removing any effect that would decrease the value of a numeric fluent.

The paper notes that, just like ignoring-deletes will not detect the case where subsequent actions remove previously achieved goals, the relaxed plan won't detect that consumption of resources can require more future production. And, just like the classical case, if the planner uses the $helpfulaction$ heuristic to further prune the search space then plans with more production actions will not be explored. This is hardly surprising as the $helpfulaction$ heuristic, even in classical planning, is known to be incomplete. The first numeric-specific consequence of the ignoring-decrease relaxation is discussed with regards to cyclical resource transfer: since the domain of numeric fluents is numbers, successive actions can make the fluents grow without bound.

The paper proposes a mixed-integer program formulation of planning problems, where variables map to variables, actions are encoded by binary variables representing if the action was taken or not, and the production or consumption effects of actions are enforced via constraining the previous value of a variable minus the next value plus the sum of the effects of all taken actions to equal zero. Such a mixed-integer program can be solved to yield upper or lower bounds for the fluent values. Furthermore, these bounds can be used to detect the minimum number of setup actions needed before a particular action becomes possible. Therefore, the mixed-integer program can be used as a relaxed-plan graph heuristic. The paper goes on to consider a number of special cases, the detection and encoding of which as MIP constraints, can improve the speed or accuracy of the solution and, therefore, the underlying planning task.

2.2 Significance

The domain-independence of planning is an appealing notion. It would permit reuse of the same algorithms and implementations to solve a wide range of problems. However, many problems are difficult to encode in the propositional language of classical planning. This is the first paper we've read that explores the issues in planning in numeric domains which, as discussed in the paper, is a natural way to encode many problems (and, therefore, has been extensively explored in related fields such as constraint optimization and dynamics and controls). To really deliver on its promise of domain-independence, a broad range of problems should be encodable into domain-independent planning, so this paper is an interesting investigation into providing those capabilities.

A natural first question when extending capabilties is "How well would faking it work?". Integers and floating-point numbers can be encoded using sets of propositional values and logical rules encoded as transforming actions over these values. In fact, this encoding is taught in every introductory computer engineering class. Also, perhaps, this is not too unusual for the planning community, since we've seen people use adders and sortnets as benchmark problems. Since, last week, we saw great results from compiling conformant planning problems to classical planning and using fast well-developed classical planners to solve them, so I'd be curious how well a similar compilation technique would work for the numeric domain. It is, however, unclear how to encode optimization problems in classical terms, without the ability to specify preferences on classical goals.