Chapter 1

Results

1.1 Empirical Evaluation

In order to empirically evaluate our algorithm, our compilation takes a temporal planning task Π and generates a set of k diverse solutions using the plan elimination compilation described in Section 4, with OPTIC [1] as the underlying solver. The TPS of each solution is obtained and $\mathbb{M}$ is computed, containing all the compatible time point pairs found in the naive TPN. We then construct a COP based on $\mathbb{M}$ in Minizinc [3] and use Gecode [2] to solve it. As output, we receive a mapping from time point pairs to merge operations resulting in a new TPN. If no compatible pairs are found (i.e. $\mathbb{M} = \emptyset$) the naive TPN is returned as output.

We evaluated all combinations of k chosen from $\{2, 4, 8\}$, and both compatibility methods (Full & Semi denoted as F,S) and merging transitivity (Strict & Loose denoted as St,L). Thus, for each problem, we ran the diverse planner 3 times (for the different values of k), and then ran 4 different versions of our COP (for the different compatibility and transitivity). The experiments were performed on Intel i7-7700K 32GB RAM, with a time limit of 30min for generating the diverse solutions and 5min for the COP task.

We evaluated our approach on all domains from the temporal track IPC in 2011, 2014, and 2018. Of these, we eliminated domains with actions whose durations depend on the current state, as this is not supported in a TPN. Out of these, we kept the domains where OPTIC was able to retrieve multiple diverse solutions for more than a single problem.

We define a run on a specific problem to be successful if: *i)* OPTIC is able to obtain k diverse solutions to the problem and *ii)* the generated TPN is more compact than the naive TPN. Otherwise, although the algorithm might have terminated successfully, we do not count it. Such a scenario occurs when the generated solutions in Π are very different from one another and no compatible pairs are found between them, leading to no possible merges to the naive TPN. In the left hand side of Table **??** we report the number of problems for which we

were able to obtain k solutions as #N, and the number of successful runs as #S.

We also report by how much our approach was able to reduce the size of the naive TPN. As different values of k for the same problem lead to different sizes of the naive TPN, we evaluate the compactness of the generated TPN relative to the size of the naive TPN: $compactness(TPN) = 1 - \frac{|\mathcal{E}_{new}|}{|\mathcal{E}_{naive}|}$. In the right hand side of Table **??** we show the average compactness over the commonly solved problems for each k by the four different configurations of our approach. #P denotes the number of commonly solved problems for each k.

Before we analyze the results, we note that the larger k is, the more IHs we have to compare between diverse solutions. Therefore we expect that an increase in k will result in more compatible pairs and thus in more merges and better compactness, but at the cost of more computational effort.

1.2 Results Discussion

As the results in Table **??** show, the above intuition is partially correct. First, as the intuition suggests, for larger k the number of successes decreases. This can be explained by the rise in complexity of the COP due to many compatible pairs and the associated high memory consumption. However, on the other extreme, when k is low there is a greater probability that the few diverse solutions generated will differ significantly from one another. Such instances may lead to either a low number of compatible pairs – less merges and a worse compactness *or* finding no compatible pairs all together and resulting in an unsuccessful run. Indeed, this is what happened in the parking and trucks domains, where using $k = 4$ resulted in more success than either $k = 2$ or $k = 8$.

We now turn out attention to examining the differences between the four configurations of our approach. First, note that using semi-compatibility always results in at least as many compatible pairs as using full-compatibility. This increase in the number of possibilities leads to an increase in the difficulty of solving the COP, which explains the higher success count of the full-compatibility, as can be seen in crewplanning for $k = 8$ and truck for $k = 4$.

On the other hand, the extra possibilities allow for more merges, and thus for better compactness. This is especially evident for $k = 2$, where a single merge contributes more to the compactness than for higher values of k, since it involves time points from a higher proportion of the original solutions.

Comparing using strict transitivity to loose transitivity, the former is a stricter constraints, thus pruning the space of possible solutions. With lower values of k, this pruning makes little difference, implying that the best solutions are not typically not pruned by this. With higher values of k this pruning reduces the size of the search space, allowing the solver to find better solutions in the alloted time, at the cost of a slight reduction in the success count.

Bibliography

[1] J. Benton, Amanda Jane Coles, and Andrew Coles. "Temporal Planning with Preferences and Time-Dependent Continuous Costs". In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. 2012.

[2] Gecode Team. *Gecode: Generic Constraint Development Environment*. Available from `http://www.gecode.org`. 2020.

[3] Nicholas Nethercote et al. "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*. 2007, pp. 529–543.