CGI2015 manuscript No.
(will be inserted by the editor)

Supplemental Material

Efficient Multi-Constrained Optimization for Example-Based Synthesis

Stefan Hartmann · Elena Trunz · Björn Krüger · Reinhard Klein · Matthias B. Hullin

the date of receipt and acceptance should be inserted later

1 Explanation: Algorithmic features used for comparison

Table 1 in the paper presents related content generation approaches based on graph search, constraint graph search and dynamic programming. Column *resource constraints/type* denotes the number of resource constraints the algorithm can handle. *Type* refers to the demand, how a constraint might be satisfied/achieved. If a constraint needs to be satisfied exact, then it is referred to an *equality constraint*. If it is restricted to any discrete value between a lower and upper bound, then its an interval constraint. *Key points* provide the capability of an approach to support placing concrete elements ('Element') or a class of elements ('Class') on user defined or domain specific positions. Examples for such classes are 'l-shaped' and 't-shaped' building parts in the building synthesis application. In the motion synthesis application, classes represent multiple motions segments performing the same action e.g 'jump', which might contain segments, where the character performs the jump on the right or the left leg. The last column *1.5D* reports if an approach can handle tree or graph structures that can be solved globally optimal and avoiding element ambiguities at once in a single graph traversal.

2 Complexity analysis

In the following we analyze the complexity of our algorithm for one resource R. Let n denote the number of elements in a database and d denote the maximum number of concatenation neighbors of an element. As we described in Section 3 in the paper, the core of the algorithm are two expansion steps, backward and forward. Each step consists of two nested *for*-loops. The outer *for*-loop is executed at most once for each node on a current level. There are at most n nodes in each level, since a new node on a level is created at most once for each element of the database. The inner *for*-loop is executed for each concatenation neighbor of the current outer loop node, thus at most d times for each node of a level. Hence, the runtime of each step is in the order of $\mathcal{O}(dn)$. Since each step, including the initialization, is executed at most once for each level, the overall runtime of the algorithm is in the order of $\mathcal{O}(Ldn)$, where L is the number of levels. By definition $L = \frac{R}{GCD}$, thus the complexity of the presented algorithm is in the order of $\mathcal{O}(\frac{R}{GCD}dn)$.

3 Comparison to Multi-Constrained Forward Graph Search

Our technique as described in the paper requires a bit of effort, and it may seem to the reader as if existing, possibly simpler, approaches should easily generalize to the RCKSP class of problems. We found that this is not the case—in fact, we only developed the proposed algorithm after finding

Address(es) of author(s) should be given

that the most promising existing technique, computing a constraint shortest path, in the fashion of Lefebvre et al. [2]. In Zhou et al. [6] the 'graph' is a dynamically growing table. However, we found that such approaches cannot be extended to reasonably deal with our larger set of goals, namely complex structures and multiple optimal solutions.

For now simply call it constraint forward graph search. We will use the remainder of this section to explain why. The "canonical" extension of the constraint forward graph search technique introduces a notion of states, which are identified by an element e and a vector of currently consumed resources $\mathbf{r}^c(e)$. Each state also stores the path cost. Starting from the source node, states with minimal cost (as opposed to minimal resource consumption) are expanded first until a state is reached that satisfies all the resource constraints. States that violate any constraints are not expanded further as it is realized in our approach. When a valid goal state is reached, the algorithm terminates and outputs the found optimal solution. This simple algorithm works because it relies on bounding the states with the integer simplification as it is done by our method as well. However, as can be easily seen, this simple algorithm has several flaws and it is not useful in our particular problem setting. Firstly, this algorithm uses the cost to sort or rank paths which are immediately explored next. This strategy, called *path ranking*, is known to be very inefficient when used for RCSP problems in theory (see Ziegelmann [7], Handler et al. [1] and Skiscim et al. [5]). The fundamental problem of such an approach is that it lacks an explicit correlation between the total cost and the feasibility of a solution in terms of resource constraints. Therefore, the number of paths that need to be enumerated might be very high until the first optimal *and* feasible solution is found. In Section 4 in the paper we provided a performance evaluation where we compare the runtime of our algorithm against a standard forward graph search and its extension to multiple constraints. These results confirm our theoretical predictions: the difference in runtime becomes more drastic when multiple constraints are involved and the standard forward search needs to run not only until one, but until all constraints are satisfied. By enumerating states by cost, the algorithm is likely to be misguided towards exploring cheap but infeasible paths first, which may heavily affect its runtime as stated by Table 1 and Table 2 in the paper.

A further gain in performance is attained using the proposed bidirectional search. Although there are cost-oriented algorithms for bidirectional search such a the approach of Lo et al. [3], in order to achieve competitive performance they require a carefully designed strategy to balance the growth of the forward and backward search trees. In addition, such approaches are not capable of delivering multiple optimal solutions. The outlined forward graph search technique would, in principle, be able to find multiple solutions by running the algorithm multiple times and suppressing edges from previous solutions. However, this implies that the guarantee of optimality is lost for all solutions except the first one. For some applications, however, the optimality of the generated solutions is of great importance; see, for example, Safonova and Hodgins [4]. Finally, we are not aware of any way of extending a cost-driven forward search towards globally optimal 1.5D or cyclic structures.

4 Illustrated Graph Setup

Fig. 1 illustrates our graph setup procedure for a small example consisting of three elements. The constraints that need to be satisfied are one equality constraint λ_2 and an interval constraint λ_1. The graph setup follows is illustrated according to the description of the intermediate graph construction given in Section 3.1 in the paper.

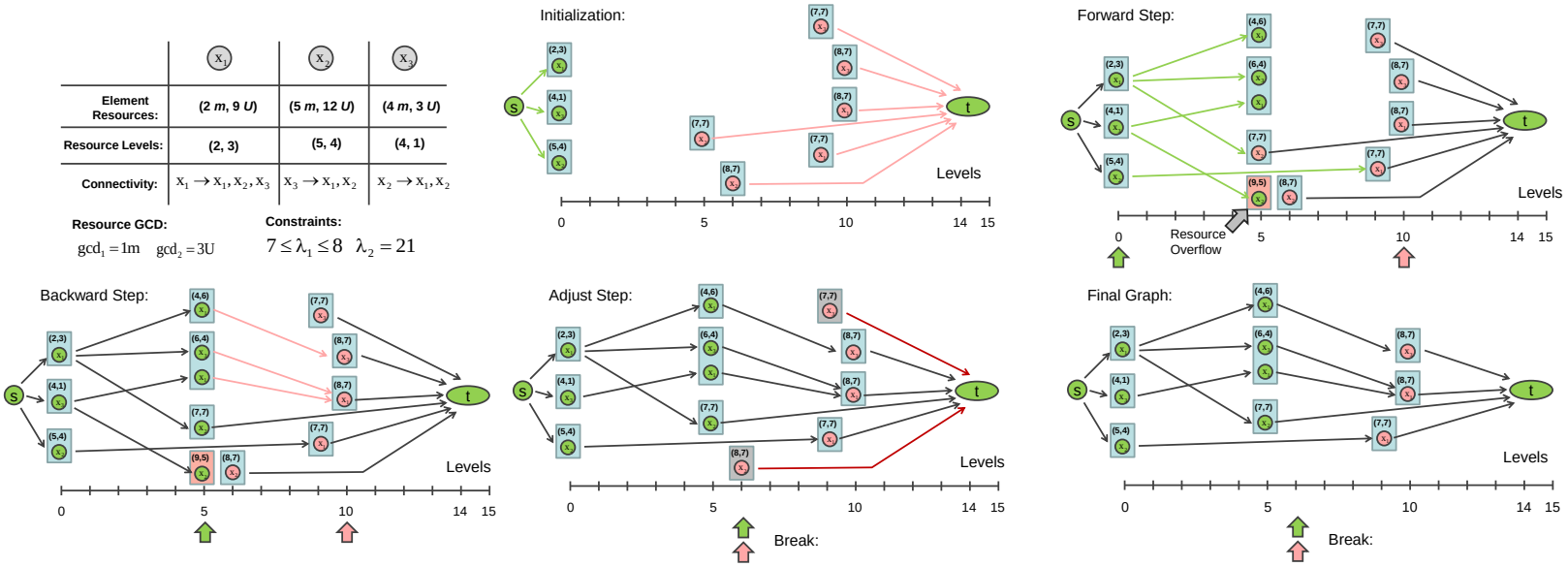

Fig. 1 A small example illustrating the graph setup procedure with resource pruning and redundant path removal

5 Additional Results:

References

1. Gabriel Y. Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.
2. Sylvain Lefebvre, Samuel Hornus, and Anass Lasram. By-example synthesis of architectural textures. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 29(4), 2010.
3. Wan-Yen Lo and Matthias Zwicker. Bidirectional search for interactive motion synthesis. *Computer Graphics Forum*, 29(2):563–573, 2010.
4. Alla Safonova and Jessica Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph.*, 26(3), 2007.
5. C. C. Skiscim and B. L. Golden. Solving k-shortest and constrained shortest path problems efficiently. *Ann. Oper. Res.*, 20(1-4):249–282, August 1989.
6. Shizhe Zhou, Changyun Jiang, and Sylvain Lefebvre. Topology-constrained synthesis of vector patterns. *ACM Trans. Graph.*, 33(6), 2014.
7. Mark Ziegelmann. *Constrained shortest paths and related problems*. PhD thesis, Saarland University, 2004.