TGFF: Task Graphs for Free

Robert P. Dick [♮], David L. Rhodes [†], and Wayne Wolf [‡]

[♮‡] Department of Electrical Engineering
Princeton University
Princeton, New Jersey 08544

[†] US Army CECOM/RDEC
AMSEL-RD-C2-SC-M
Fort Monmouth, New Jersey 07703

Abstract

We present a user-controllable, general-purpose, pseudorandom task graph generator called Task Graphs For Free (TGFF). TGFF creates problem instances for use in allocation and scheduling research. It has the ability to generate independent tasks as well as task sets which are composed of partially ordered task graphs. A complete description of a scheduling problem instance is created, including attributes for processors, communication resources, tasks, and inter-task communication. The user may parametrically control the correlations between attributes. Sharing TGFF's parameter settings allows researchers to easily reproduce the examples used by others, regardless of the platform on which TGFF is run.

1. Introduction

Research in embedded real-time systems and operating systems, as well as in more general allocation and scheduling fields, is hampered by the lack of a common base of examples. In general, an example used in allocation and scheduling research consists of a task set and a database of processors and communication resources. A **task set** is a collection of **task graphs**, each of which is a directed acyclic graph (**DAG**) of communicating tasks. Generation of sample task sets is often a requirement when comparing allocation or scheduling methods with each other [1], [2]. There are generally no standard task sets available, making comparison of different methods all but impossible. Moreover, since task set generation is only a secondary aspect of scheduling research, the details necessary to enable exact recreation of another researcher's task sets are usually lacking. At best, re-implementation of another researcher's random task set generation algorithm is tedious. At worst, the new implementation subtly differs from the algorithm used in the work with which a comparison is made, resulting in misleading experimental results. These problems conspire to make it difficult to compare one's new allocator or scheduler with existing algorithms.

This situation would be improved by the existence of a standard, shareable base of task sets which are sufficiently general to enable applicability to a wide range of areas (*e.g.*, embedded systems and parallel computing) and which can be tuned to particular problem domains. Shareable examples have been critical to progress in other areas such as computer-aided design and computer science, *e.g.*, the standard ISCAS digital circuits used to compare digital circuit simulators [3] or the DIMACS Boolean formula sets used for satisfiability solvers [4]. However, a survey in the area of task sets reveals that researchers are "on their own"; this is true among both the industrial and academic research communities. Allocation and scheduling research is a sufficiently broad area that any static set of examples meeting the needs of the majority of researchers would be gigantic. TGFF gives researchers the flexibility to dynamically tailor examples to their work while making it easy for others to regenerate these examples, given knowledge of a few command line parameters. It has been used in our current allocation and scheduling research [5].

Some allocation and scheduling research for very high-level system design assumes that there are no data dependencies between different tasks in a task set, while at the other extreme, directed, *cyclic* task graphs usually arise in low-level or small-grain arenas, for example, in instruction-level code analysis. TGFF's task graph format, the DAG, is commonly used in medium-level and high-level al-

[♮] This work was supported in part by an NSF Graduate Fellowship and in part by NSF under Grant No. MIP-9423574.

We would like to thank Niraj K. Jha, at Princeton University, for his valuable comments regarding this paper.

location and scheduling research in academia and industry [6]–[8]. TGFF is nonetheless capable of generating sets of independent tasks as a special case of the sets of DAGs for which it is primarily intended.

TGFF includes a pseudorandom number generator [9]. This generator behaves identically on any machine which represents mantissas with 24 or more bits. Given the same command line options, TGFF will generate the same task set, processors, and communication resources when run on nearly any architecture which supports floating point computation.

2. Task Set Generation

Task graphs may be roughly categorized by their structural properties. DAGs generated to solve some numeric or algorithmic method, for example an FFT computation or a Quicksort, exhibit a particularized (and predictable) structure. Although there also appears to be a lack of shareable task graphs in this 'structured-graph' regime, these types of graphs are more easily documented and re-created than more randomly structured graphs. Thus, the TGFF effort focuses on random task graph generation subject to the limitations and parameters provided by the user.

TGFF generates a given number of random task graphs, where the graph **nodes** are tasks and the graph **arcs** represent communication between tasks. Arcs are associated with parametrically controlled data volume scalars; they represent interprocess communication and impose a partial order on nodes. TGFF accepts a random number generator seed parameter, among others. The value of the seed affects both the structure as well as other aspects of the task set. Task set **families** containing an arbitrary number of task sets may be generated by varying the seed while holding all other parameters constant.

Documentation is provided with the software. Therefore, only a high-level description is given here. One of the most challenging aspects of generating task graphs is developing an algorithm for defining their structure. For TGFF, there are a number of parameters relevant to the task graph structure: the average (n) and multiplier (m) for the lower bound on the number of nodes in a graph, and the maximum in-degree (id) and out-degree (od) of graph nodes. While id and od are fixed for every task graph generated in the task set, a value for the lower bound is selected at random from the uniform range $[n - m,\ n + m]$.

Let x be a lower bound on the number of nodes in a task graph, as randomly selected from the $[n-m,\ n+m]$ range. The task graph is constructed by first creating a single-node graph and then iteratively augmenting it until the number of nodes in the graph is greater than or equal to x.

The augmentation operates as follows. First randomly select either a *fan-out* step or a *fan-in* step (with equal probability). If it is a fan-out step, find the set of nodes that have the largest amount of 'available' out-degree, *i.e.*, those with the maximum difference between od and the actual number of out-arcs, and call this maximum difference r. Assuming that $r > 0$, randomly pick a node, p, from the set, and then add y nodes and arcs to the graph from p to each of these new y nodes where y is a random number ranging from 0 to r.

If it is a fan-in step, find a set of existing nodes which are not over their od limit and call the cardinality of this set q. Assuming that $q > 0$, randomly select a value z in the range $[0,\ \max(q, id)]$. Add a single node to the graph and z arcs from z nodes from the set to this new node.

This procedure generates DAGs which honor the in-degree and out-degree limits, contain at least x nodes, have a single start node, and do not have duplicated arcs (*e.g.*, those between the same pair of nodes). The actual number of nodes in the generated task graph ranges from x to $x + od - 1$.

TGFF associates a deadline with every **terminal node** (a node which has no outgoing arcs) in the task graphs it produces. A heuristic is used to generate deadlines which are likely to be challenging but tractable. If $depth$ is the length of the maximum-length path from a task graph's start node to a given node, av_exec_time is the user-specified average amount of time taken to execute a task, and d_laxity is an arbitrary scalar, then the deadline for that node is set in the following manner: $deadline = depth \cdot av\_exec\_time \cdot d\_laxity$.

Task sets containing task graphs with differing periods are termed multi-rate task sets. TGFF is capable of parametrically generating the periods of task graphs in multi-rate task sets. The user specifies an array of period multipliers which is used to determine the relative periods of different task graphs in the task set. Selecting only small integer multipliers allows one to generate a task set which

can feasibly be scheduled with the least common multiple scheduling method [10]. However, a user is free to specify multipliers which are vastly different or for which the least common multiple is large, relative to the individual multipliers. Given mul_ar (an array of user-provided period multipliers), p_laxity (a user-provided scalar), and tg_ar (an array containing all the task graphs in the task set), TGFF uses the algorithm in Figure 1 to assign a period to each task graph. This algorithm generates periods which are based on the period multiplier array provided by the user and are loosely related to the deadlines of individual task graphs.

mul_ar is a user-specified array of multipliers
tg_ar is an array of task graphs
mul_ls is an empty list
p_laxity is a user-specified scalar

while $mul\_ls \rightarrow elements < tg\_ar \rightarrow elements$:
    select mul randomly from mul_ar
    append mul to mul_ls

sort mul_ls in increasing order
sort tg_ar in order of increasing deadlines

$gr = tg\_ar[\text{last}] \rightarrow deadline \ / \ mul\_ls[\text{last}]$

for each i in all task graph indexes:
    $tg\_ar[i] \rightarrow period = gr \cdot mul\_ls[i] \cdot p\_laxity$

Figure 1: Period computation algorithm

An important characteristic of task sets is the relation between the deadlines and the periods of their task graphs. While some schedulers allow periods that are less than deadlines (*e.g.*, [5], [8]), many do not. If requested, TGFF prevents the period of any task graph from being greater than any of the deadlines within it.

In addition to the primary output file, a PostScript file depicting the task set is generated. Figure 2 shows an example task graph output by TGFF's PostScript facility. This is a problem instance with a single task graph (`-n1`), a maximum in-degree and out-degree of two (`-e2:2`), a number of nodes ranging from eight to twelve per task graph (`-g10:2`), and a random seed of four (`-s4`). In this illustration, each task is represented by a square and is labeled with its number. In addition to its task number, each terminal node is labeled with its deadline. A task graph family of 50 single task graphs can be generated by running TGFF with the following flags, '`-n 1 -s`x,' where x is given values over the range $\{0, 1, 2, \cdots, 49\}$. This statement is sufficient documentation to enable other researchers to reproduce exactly the same family. Figure 4 shows the task set produced when TGFF is run with its in-degree restricted to one and its out-degree restricted to two (`-e1:2`), forcing TGFF to generate out-trees rather than more general DAGs. As another example, Figure 5 shows the generation of three task graphs with widely varying numbers of tasks.

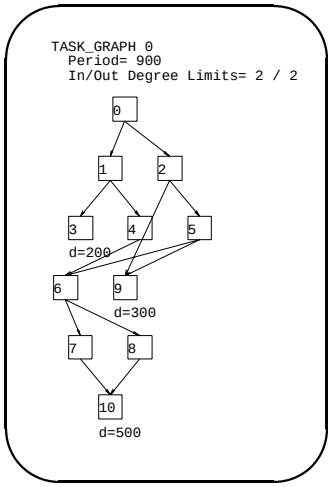

Figure 2: Result for `tgff -n1 -e2:2 -g10:2 -s4`

3. Database Generation

Some work in allocation and scheduling optimizes multiple attributes, *e.g.*, execution time, power consumption, testability, and cost. TGFF supports this by allowing an arbitrary number of attributes, which may be correlated or uncorrelated, to be associated with each processor and communication resource.

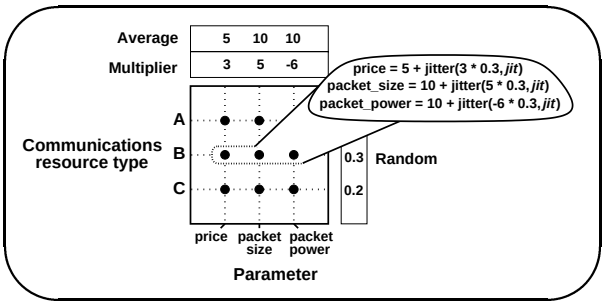

Figure 3: Setting communication resource attributes

Although attribute generation for processors and communication resources is similar, communication resource attribute generation is more straightforward. This process is most easily illustrated with an example. Figure 3 depicts attribute generation for communication resources. TGFF generates a random scalar ($com \rightarrow rand$), ranging from -1 to 1, for each communication resource. The user specifies an average (av) and a multiplier (mul)

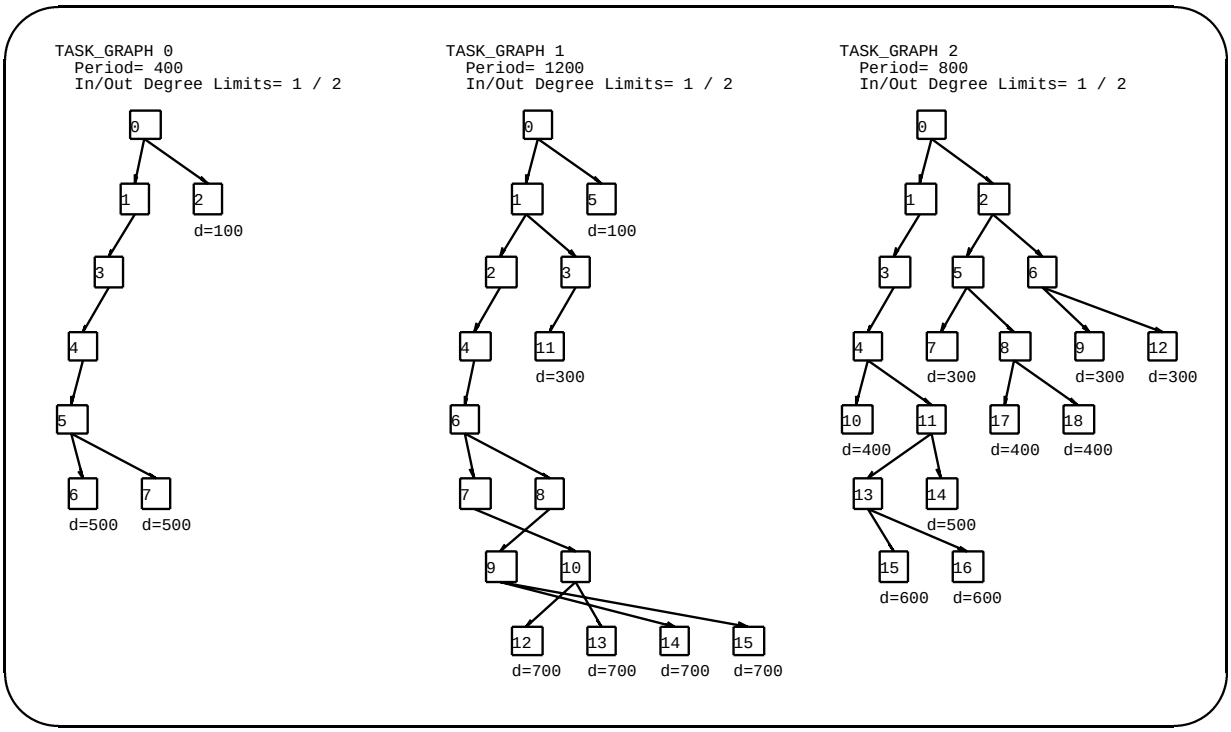

Figure 4: Result for `tgff -e1:2 -g15:14`

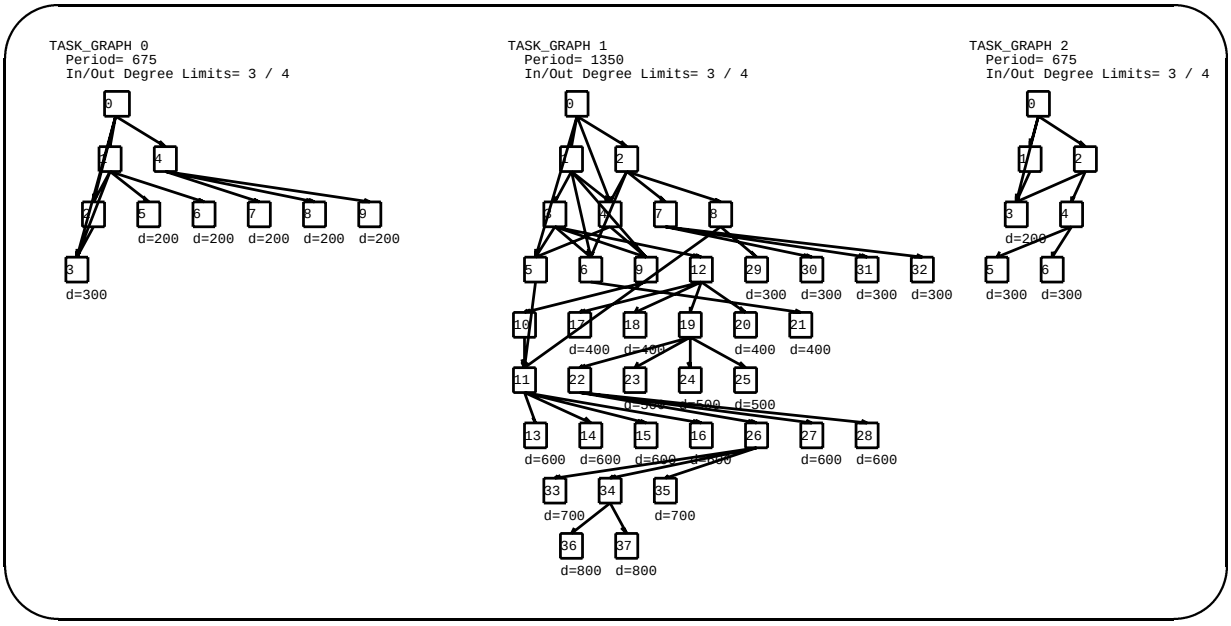

Figure 5: Result for `tgff -e3:4 -g20:18 -s7`

value for each communication resource attribute, as well as a jitter (jit) for the task set. Given a scalar (x) and the task set jitter (jit) the function $\text{jitter}\,(x, jit)$ returns a randomly selected number from the uniform range $[x \cdot (1 - jit)\,,\ x \cdot (1 + jit)]$. With this function, and the parameters specified by the user, TGFF generates the attributes for each communication resource, *i.e.*,

$$attrib = av + \text{jitter}\,(mul \cdot com{\rightarrow}rand,\ jit)$$

A processor has attributes which are independent of tasks, as well as attributes which indicate the behavior of each task on that processor. Independent attribute generation is analogous to com-

munication resource attribute generation. Task-processor intersection attributes, which provide information about a task's execution on a particular processor, are generated with procedure similar to the one illustrated in Figure 3. However, for task-processor intersections, the procedure operates in three dimensions instead of two. In addition to an array of random numbers associated with processors, there is a similar array associated with tasks. Each attribute depends on the processor and task for which the attribute is being generated.

TGFF has a number of default attributes: `cost` for processors, `cost` and `transmit_rate` for communication resources, and `exec_time` for tasks. These attributes can be augmented or altered. As an example demonstrating TGFF's generality, consider the following scenario: one wants to add an attribute which defines a `setup` time for communication resources. This attribute is, in general, to be inversely related to cost. By giving TGFF the following command-line flag, `-C '10:5:t:cost 100:-80:f:setup'`, one declares that `cost` has an average value of 10, a multiplier of 5, and is an integer. Similarly, `setup` has a average value of 100, a multiplier of -80, and is a real number. Setting `cost`'s multiplier to a positive value and `setup`'s multiplier to a negative value causes them, in general, to be inversely related to each other. A portion of the resulting output appears in Figure 6.

```
@COMMUN 0 {
# cost      setup
     12   68.5145
}

@COMMUN 1 {
# cost      setup
      9   119.64
}

@COMMUN 2 {
# cost      setup
     10   92.5214
}
```

Figure 6: Communication resource attributes

4. Conclusions

TGFF provides a standard method for generating random allocation and scheduling problem instances involving periodic or non-periodic task sets. Users have parametric control over an arbitrary number of attributes for tasks, processors, and communication resources. TGFF is capable of generating problem instances which are tuned to particular domains in allocation and scheduling research. However, the ease with which its parameters can be changed allows it to be applied to many allocation and scheduling domains. Although TGFF simplifies the rapid production of large families of examples, this work's primary goal is to encourage comparison of allocation and scheduling algorithms by making it practical to reproduce the examples used by other researchers. The source code for TGFF is available via the "projects" link on the `http://www.ee.princeton.edu/~cad` web page.

References

[1] T. Yang and A. Gerasoulis, "DSC: scheduling parallel tasks on an unbounded number of processors," *IEEE Trans. on Parallel and Distributed Systems*, vol. 30, pp. 951–67, Sep 1994.

[2] W. Zhao, K. Ramamrithan, and J. Stankovic, "Preemptive scheduling under time and resource constraints," *IEEE Trans. on Computers*, vol. 36, pp. 949–60, Aug. 1987.

[3] M. Sengupta, "ISCAS '89 benchmark information," *http://www.cbl.ncsu.edu/CBL_Docs/iscas89.html*, Mar. 1995.

[4] D. Du, J. Gu, and P. M. Pardalos, eds., *Satisfiability Problem: Theory and Applications*, vol. 35 of *DIMACS: Series in Discrete Mathematics and Computer Science*. Providence, RI: American Mathematical Society, 1997.

[5] R. P. Dick and N. K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for the Hardware-Software Co-Synthesis of Distributed Embedded Systems," submitted to *IEEE Trans. on Computer-Aided Design*.

[6] S. Prakash and A. Parker, "SOS: Synthesis of application-specific heterogeneous multiprocessor systems," *J. Parallel & Distributed Computers*, vol. 16, pp. 338–351, Dec. 1992.

[7] T.-Y. Yen and W. H. Wolf, "Communication synthesis for distributed embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, pp. 288–294, Nov. 1995.

[8] B. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of embedded systems," in *Proc. Design Automation Conf.*, pp. 703–708, June 1997.

[9] G. Marsaglia and A. Zaman, "Toward a universal random number generator," *Statistics & Probability Letters*, vol. 9, pp. 35–39, Jan. 1990.

[10] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors," *Information Processing Letters*, vol. 7, pp. 9–12, Feb. 1981.