Towards Optimal Scheduling for Global Computing under Probabilistic, Interval, and Fuzzy Uncertainty, with Potential Applications to Bioinformatics

Roberto Araiza, Michela Taufer
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
{raraiza,mtaufer}@utep.edu

Ming-Ying Leung
Bioinformatics Program
University of Texas at El Paso
El Paso, TX 79968, USA
mleung@utep.edu

***Abstract*—In many practical situations, in particular in many bioinformatics problems, the amount of required computations is so huge that the only way to perform these computations in reasonable time is to distribute them between multiple processors. The more processors we engage, the faster the resulting computations; thus, in addition to processor exclusively dedicated to this job, systems often use idle time on other processors. The use of these otherwise engaged processors adds additional uncertainty to computations.**

How should we schedule the computational tasks so as to achieve the best utilization of the computational resources? Because of the presence of uncertainty, this scheduling problem is very difficult not only to solve but even to formalize (i.e., to describe in precise terms). In this paper, we provide the first steps towards formalizing and solving this scheduling problem.

I. Supercomputing

At present, most useful computations are performed on individual computers. However, there are practical problems which require orders of magnitude more computations than a regular computer can perform. To perform such computations, we need what is often called a "supercomputer".

Such problems include processing DNA data and other relevant bioinformatics data, weather prediction and climate analysis, etc. For example, in bioinformatics, one of the most time-consuming tasks is to look for known patterns in a long DNA or RNA sequence.

II. Supercomputing in the Past

In this paper, we will analyze scheduling in *global computing*. To explain the idea (and the necessity) of global computing, it is important to explain how the concept of supercomputing has evolved in the last decades.

In the past, the ability to use supercomputers to simulate such things as nuclear weapons design was an important part of military confrontation. As a result, special classified technology was used to design supercomputers, technology that was not allowed on regular individual computers.

III. Supercomputing at Present

Since the end of the Cold War, military restrictions no longer serve as a serious limitation to the mass-produced computer technology. As a result, the current PCs are almost as fast as specially designed computer processors.

Hence, many existing supercomputers are designed by connecting regular off-the-shelf computer processors together.

IV. Global Computing

Since regular computers are almost as powerful as any processor within a supercomputer, a natural idea is to use idle cycles of the regular computers to perform high-throughput computations. This idea enables us, in effect, to build a powerful supercomputer out of the existing computers – and we do not even need to own them, it is enough to use their idle cycles. Of course, due to communication time, this idea may not always work for real-time computations where we need, e.g., to predict the path of a upcoming dangerous storm. However, in many scientific computations, a communications-related delay of a day or two may be quite reasonable – as long as we do eventually perform all the necessary computations.

This idea started in the 1990s with SETI@Home, where global computing was used to process signals from radio telescopes in search for messages from extra-terrestrial intelligence. At present, this idea is actively used in mainstream research. For example, our group has developed easy-to-install web browser extension tools [2], [3], [14], [15], [17] which, in effect, enable computers to work together. The resulting networks are already being used for bioinformatics applications [13], [16].

V. One of the Main Problems of Global Computing: Scheduling under Uncertainty

A serious problem in global computing is scheduling; see, e.g., [1], [5], [6], [10], [11], [12]. A similar problem occurs when we combine several processors into a single supercomputer, but there, usually, all the processors are similar, and we are in complete control of them. The corresponding scheduling problem is computationally difficult, but it is well formulated, without any serious uncertainty.

In contrast, in global computing, we are connecting computers of different types, some of which we own, some of which

we don't own (so we can only use their idle cycles). We do not have an exact understanding of how the resulting collaboration affects computation time, how much time is available as idle cycles, etc. In other words, to make an efficient use of resources in global computing, we must perform scheduling under uncertainty. The existing scheduling tools for such scheduling are still imperfect [1], [5], [6], [10], [11], [12].

VI. We Need All Types of Uncertainty

- In some cases, we have *interval uncertainty*: e.g., we may know that a certain processing step takes between 5 and 10 minutes on a given computer.
- In other cases, we have *probabilistic* uncertainty: e.g., based on the past experience, we may know the mean processing value – or we may even know the probability distribution for computation time.
- We can also have *expert* estimates for computation time, such as "the time is usually much faster than 10 minutes" which are natural to describe in *fuzzy* terms.

In scheduling, we need to take into account all these three types of uncertainty.

Let us show how the problem of selecting the optimal schedule under these types of uncertainty can be described in precise terms.

VII. Which Schedule is Optimal: Towards a Formal Description of the Appropriate Objective Function

Formalizing the main objective: first try. The need for parallelization comes from the fact that computing the original task on a sequential machine requires too long a time.

From this viewpoint, a reasonable objective of parallelization is to minimize the overall computation time t.

Formalizing the main objective: complications caused by uncertainty. In global computing, we use idle time of otherwise engaged computers. This idle time depends on whether (and to what extent) these computers are engaged in other computations. Thus, for the same schedule, the actual computation time t may differ from situation to situation.

So, we may get different computation times with different probabilities.

Formalizing the main objective: second try. Due to uncertainty, we cannot guarantee the *exact* value of the computation time. Moreover, with some (hopefully small) probability, the actual computation time may turn out be be very large.

If this probability is small enough, then the situation is quite tolerable: indeed, for every computer (even a dedicated one), there is always a probably of hardware failure which would make the computations impossible.

It is therefore reasonable to select a tolerable probability of failure ε, and to gauge each schedule by the time t_0 during which this schedule completes computation with probability $1-\varepsilon$.

Then, we select the schedule for which this time t_0 is the smallest.

Formalizing the main objective: additional complications caused by uncertainty. In the idealized case when we know the probabilities of all possible engagements of different computers, we can simulate the involved network of computers and find the probability that the task will be performed in any time period t. In such idealized situation, we can then find the value t_0 for which the probability of success is $1-\varepsilon$, and select the schedule for which the value is the smallest.

In reality, we do not have a full knowledge of the corresponding probabilities. Because of this incomplete knowledge, for a given schedule, we cannot uniquely predict the probability that under schedule, the original task will be performed in time t. The actual probability of success may depend on the parameters which are unknown to us.

Formalizing the main objective: final idea. We have mentioned that we cannot exactly predict the actual probability with which a given plan will succeed in time t. For different possible probability distributions, we may have different probabilities.

Our objective is to guarantee that the computations are done. Thus, a reasonable measure of the schedule's quality is the time t_0 by which we can *guarantee* that the computations finish with the probability $1-\varepsilon$.

Formalizing the main objective: resulting formalization. We want to select the schedule for which the time t_0 during which computations are guaranteed to finish with probability $\geq 1-\varepsilon$.

Need to compute the time of guaranteed completion. In view of this objective, to select the optimal schedule, we must be able, for each schedule, to compute the time t_0 during which computations are guaranteed to finish with probability $\geq 1-\varepsilon$.

Let us now describe what information we can use to compute this time, and how we can use this information.

VIII. How to Predict the Guaranteed Computation Time

Need for such a prediction: reminder. We have argued that a reasonable way to select a computation schedule is to select a schedule for which the guaranteed (with probability $\geq 1-\varepsilon$) computation time t_0 is the smallest. Thus, to find the optimal schedule, we must be able to compute this guaranteed computation time.

What is a schedule. The main idea of parallelization is that the original time-consuming task into jobs (subtasks) which can be performed independently. A schedule describes how exactly this parallelization is performed, i.e., how exactly the original task is divided into subtasks, and which processor is assigned which subtask.

Example from bioinformatics. As we have mentioned, in bioinformatics, one of the most time-consuming tasks is to look for known patterns in a long DNA or RNA sequence. This task can be parallelized if:

- we divide the original sequence into pieces,
- assign each piece to a different computer, and
- ask the corresponding computer to search for the desired pattern within its piece.

Of course, the pieces must overlap – otherwise this procedure may miss the pattern if it happens that this pattern is split between two neighboring pieces.

As soon as all the jobs are done, the original task is performed.

An expression for the overall time in terms of times of subtasks. The overall time required by the parallelized procedure can be defined as $t = t_e - t_s$, where:

- t_s is the moment when the original task was submitted by the user to the submit point, and
- t_E is the moment when all the jobs (subtasks) have been performed and their results have been returned to the submit point.

For every job i, let t_i denote the time from t_s to the moment when the results of this job are returned to the submit point. The original task is done when the last of these jobs is performed, the job which requires the largest amount of time. Thus $t = \max\limits_i t_i$, where the maximum is taken over all the jobs i.

For each task i, we need some time to send it off to a currently idle processor, process it there, and then send the results back. The overall time t_i is therefore equal to the sum $\sum\limits_j t_{ij}$ of the times of these steps. The overall computation time is thus equal to the longest of these times, i.e., to

$$t = \max_i \sum_j t_{ij}.$$

Comment. For bioinformatics problems, each subtask is performed on a single processor, hence each job time t_i is the sum of the times t_{ij} corresponding to three above-described steps. The possibility of using a single processor for each subtask is due to the fact that each subtask is reasonably short, so the probability that the auxiliary processor remains idle during these computations remains high.

In other application areas, it may not be possible to subdivide the original task into parallelizable short subtasks; the subtasks are much longer. In this case, there is a high probability that a processor would stop being idle before the subtask is completed, and the subtask will not be finished. To avoid this situation, it is reasonable to subdivide this subtask into several sequential steps, and assign each step to a different processor:

- the first processor performs the first step, then return the results to the submit point;
- these results will then be sent to the second processor, to perform the second step of the subtask, etc.

In this case, the overall time t_i for computing the i-th job can be described by a similar formula $t_i = \sum\limits_j t_{ij}$, but now we can have more than three steps j.

In view of this possibility, in the following text, we will consider the general case of possibly > 3 steps j.

We only have partial information about the times t_{ij}. We have described a formula that relates the desired computation time t with the times t_{ij} of performing different steps.

If we knew the exact values of t_{ij}, then we could use the above formula to compute the exact value of the overall computation time. If we knew the probability distribution for each of the times t_{ij}, then we could find the probabilities of different values of t; in particular, we would be able to find the probability that t is below the given value t_0.

In reality, in most cases, we do not know the exact value t_{ij}, and we only partial knowledge about the corresponding probabilities.

What types of partial information about the times t_{ij} do we have? We can safely assume that different values t_{ij} are statistically independent.

For some times t_{ij}, we know the probability distribution.

For other times t_{ij}, we know the bounds $\underline{t}_{ij} \le t_{ij} \le \overline{t}_{ij}$ and the mean $E[t_{ij}]$.

In other cases, we have fuzzy information about the bounds and means.

We would like to use this information to estimate the guaranteed computation time.

IX. From the Computational Viewpoint, It Is Sufficient to Consider Interval Uncertainty

In the fuzzy case, to describe the corresponding uncertainty about t_{ij}, for each value t of the time t_{ij}, we describe the degree $\mu_{ij}(t)$ to which this value is possible.

For each degree of certainty α, we can determine the set of values of t_{ij} that are possible with at least this degree of certainty – the *α-cut* $\mathbf{t}_{ij}(\alpha) \stackrel{\text{def}}{=} \{t \,|\, \mu_{ij}(t) \ge \alpha\}$ of the original fuzzy set. In many practical cases, this α-cut is an interval.

Vice versa, if we know α-cuts for every α, then, for each value t, we can determine the degree of possibility that t belongs to the original fuzzy set for t_{ij} [4], [7]. A fuzzy set can be thus viewed as a nested family of its α-cuts.

A *fuzzy number* can be defined as a fuzzy set for which all α-cuts are intervals.

So, if instead of an interval $[\underline{t}_{ij}, \overline{t}_{ij}]$ of possible values of the time t_{ij}, we have a fuzzy number $\mu_{ij}(t)$ of possible values, then we can view this information as a family of nested intervals $\mathbf{t}_{ij}(\alpha)$ (α-cuts of the given fuzzy sets).

Our objective is then to compute the fuzzy number t_0 corresponding to the desired time. In this case, for each level α, the corresponding α-cut of the desired fuzzy number can be computed based on the α-cuts $\mathbf{t}_{ij}(\alpha)$ of the corresponding input fuzzy sets. The resulting nested intervals form the fuzzy number for the desired time t_0.

So, e.g., if we want to describe 10 different levels of uncertainty, then we must solve 10 interval computation problems. Thus, from the computational viewpoint, it is sufficient to produce an efficient algorithm for the interval case.

X. Towards a Mathematical Formulation of the Problem

Mathematical observation: properties of the dependence of t on t_{ij}. Let us observe that the function $t = \max_i \sum_j t_{ij}$ which describes the dependence of the overall computation time t on the times t_{ij} is non-negative and convex.

Let us recall that a function $f : R^m \to R$ is called *convex* if

$$f(\alpha \cdot x + (1-\alpha) \cdot y) \le \alpha \cdot f(x) + (1-\alpha) \cdot f(y)$$

for every $x, y \in R^m$ and for every $\alpha \in (0,1)$. It is known that the maximum of several linear functions is convex, so our function is indeed convex.

Our objective. We want to find the smallest possible value t_0 such that for all possible distributions consistent with the known information, we have $t \le t_0$ with the probability $\ge 1-\varepsilon$ (where $\varepsilon > 0$ is a given small probability).

What information we can use. We assume that different values t_{ij} are statistically independent:

- About some of the variables t_{ij}, we know their exact statistical characteristics.
- About some other variables t_{ij}, we only know their interval ranges $[\underline{t}_{ij}, \overline{t}_{ij}]$ and their means E_{ij}.

Additional property: the dependency is non-degenerate. We only have partial information about the probability distribution of the variables t_{ij}. For each possible probability distribution p, we can find the largest value t_p for which, for this distribution, $t \le t_p$ with probability $\ge 1-\varepsilon$. The desired value t_0 is the largest of the values t_p corresponding to different probability distributions p: $t_0 = \sup_{p \in \mathcal{P}} t_p$, where $\mathcal{P}$ denotes the class of probability distributions p which are consistent with the known information.

If we learn some additional information about the distribution of t_{ij} – e.g., if we learn that t_{ij} actually belongs to a proper subinterval of the original interval $[\underline{t}_{ij}, \overline{t}_{ij}]$ – we thus decrease the class $\mathcal{P}$ of distributions p which are consistent with this information, to a new class $\mathcal{P}' \subset \mathcal{P}$. Since the class has decreased, the new value $t_0' = \sup_{p \in \mathcal{P}'} t_p$ is the maximum over a smaller set and thus, cannot be larger than the original value t_0: $t_0' \le t_0$.

From the purely mathematical viewpoint, it is, in principle, possible that the desired value t_0 does not actually depend on some of the variables t_{ij}. In this case, if we narrow down the interval of possible values of the corresponding variable t_{ij}, this will not change the resulting value t_0.

In our problem, however, it is reasonable to assume that the dependence of t_0 on t_{ij} is *non-degenerate* in the sense that every time we narrow down one of the intervals $[\underline{t}_{ij}, \overline{t}_{ij}]$, the resulting value t_0 actually decreases: $t_0' < t_0$.

As a result, we arrive at the following problem.

XI. Formulation of the Problem and the Main Result

GIVEN:
- a finite set of M pairs of integers (i,j), and its subset F;
- a real number $\varepsilon > 0$;
- a convex non-negative function
$$t = F(t_{11}, t_{12}, \ldots);$$
- probability distributions for variables t_{ij} with $(i,j) \in F$ – e.g., given in the form of cumulative distribution function (cdf) $F_{ij}(t)$;
- intervals $\mathbf{t}_{ij} = [\underline{t}_{ij}, \overline{t}_{ij}]$ and values E_{ij} corresponding to $(i,j) \notin F$.

TAKE: all possible joint probability distributions on R^M for which:
- all N random variables t_{ij} are independent;
- for all $(i,j) \in F$, the variable t_{ij} has a given distribution $F_{ij}(t)$;
- for each $(i,j) \notin F$, $t_{ij} \in \mathbf{t}_{ij}$ with probability 1 and the mean value of t_{ij} is equal to E_{ij}.

FIND: find the smallest possible value t_0 such that for all possible distributions consistent with the known information, we have $t \stackrel{\text{def}}{=} F(t_{11}, t_{12}, \ldots) \le t_0$ with probability $\ge 1-\varepsilon$.

PROVIDED: that the problem is *non-degenerate* in the sense that if we narrow down one of the intervals $\mathbf{t}_{ij}$, the value t_0 decreases.

The following result explains how we can compute this value t_0.

Proposition. *The desired value t_0 is attained when for each $(i,j) \notin F$, we use a 2-point distribution for t_{ij}, in which:*

- *$t_{ij} = \underline{t}_{ij}$ with probability $\underline{p}_{ij} \stackrel{\text{def}}{=} \dfrac{\overline{t}_{ij} - E_{ij}}{\overline{t}_{ij} - \underline{t}_{ij}}$.*
- *$t_{ij} = \overline{t}_{ij}$ with probability $\overline{p}_{ij} \stackrel{\text{def}}{=} \dfrac{E_{ij} - \underline{t}_{ij}}{\overline{t}_{ij} - \underline{t}_{ij}}$.*

Comment. A similar proposition was first proven in [8], [9] for a completely different computer-related application – to chip design. For reader's convenience, the proof (adjusted to our problem) is given in the special Appendix.

XII. Resulting Algorithm for Computing t_0

Because of the above Proposition, we can compute the desired value t_0 by using the following Monte-Carlo simulation:

- We set each value t_{ij}, $(i,j) \notin F$, to be equal:
  - to $\overline{t}_{ij}$ with probability $\overline{p}_{ij}$ and
  - to the value $\underline{t}_{ij}$ with the probability $\underline{t}_{ij}$.
- We simulate the values t_{ij}, $(i,j) \in F$, as random variables distributed according to the distributions $F_{ij}(x)$.
- For each simulation s, $1 \le s \le N_i$, we get the simulated values $t_{ij}^{(s)}$, and then, a value $t^{(s)} = F(t_{11}^{(s)}, t_{12}^{(s)}, \ldots)$. We then sort the resulting N_i values $t^{(s)}$ into an increasing sequence
$$t_{(1)} \le t_{(2)} \le \ldots \le t_{(N_i)},$$

and take, as t_0, the $N_i \cdot (1-\varepsilon)$-th term $t_{(N_i \cdot (1-\varepsilon))}$ in this sorted sequence.

ACKNOWLEDGMENTS

This work was supported in part by the Texas Advanced Research Program Grant No. 003661-0008-2006.

The authors are thankful to Vladik Kreinovich for his help and to the anonymous referees for valuable suggestions.

REFERENCES

[1] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, and U. Nagashima, "A performance evaluation model for effective job scheduling in global computing systems", *Proceedings of the Seventh International IEEE Symposium on High Performance Distributed Computing*, July 28–31, 1998, pp. 352–353.

[2] K. Bhatia, B. Stearn, M. Taufer, R. Zamudio, and D. Catarino, "Extending Grid Protocols onto the Desktop using the Mozilla Framework", *Proceedings of the 2nd International Workshop on Grid Computing Environments (GCE 2006)*, a workshop in conjunction with SuperComputing 2006 Conference, Tampa, Florida, November 2006.

[3] K. Bhatia, M. Taufer, B. Stearn, R. Zamudio, and D. Catarino, "Integrate GridFTP into Firefox – Build grid protocols into Mozilla-based tools", *IBM developerWorks*, 10 Oct. 2006, available at http://www-128.ibm.com/developerworks/grid/library/gr-firefoxftp/

[4] D. Dubois and H. Prade, "Operations on fuzzy numbers", *International Journal of Systems Science*, 1978, Vol. 9, pp. 613–626.

[5] J. Han and D. Park, "Scheduling proxy: enabling adaptive-grained scheduling for global computing system", *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, November 8, 2004, pp. 415–420.

[6] D. Kondo, H. Casanova, E. Wing, and F. Berman, "Models and scheduling mechanisms for global computing applications", *Proceedings of the International IEEE Symposium on Parallel and Distributed Processing IPDPS'2002*, April 15–19, 2002, pp. 79–86.

[7] H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic*, CRC Press, Boca Raton, Florida, 2006.

[8] M. Orshansky, W.-S. Wang, Martine Ceberio, and G. Xiang, "Interval-Based Robust Statistical Techniques for Non-Negative Convex Functions, with Application to Timing Analysis of Computer Chips", *Proceedings of the Symposium on Applied Computing SAC'06*, Dijon, France, April 23–27, 2006, pp. 1645–1649.

[9] M. Orshansky, W.-S. Wang, G. Xiang, and V. Kreinovich, "Interval-Based Robust Statistical Techniques for Non-Negative Convex Functions, with Application to Timing Analysis of Computer Chips", *Proceedings of the Second International Workshop on Reliable Engineering Computing*, Savannah, Georgia, February 22–24, 2006, pp. 197–212.

[10] G. Sun, B. Fan, G. Chen, and Y. Zhou, "Study on Scheduling Strategy for Global Computing Application", *Proceedings of the Seventh IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies PDCAT'06*, December 2006, pp. 368–372.

[11] G. Sun, J. Shan, and G. Chen, "Job Scheduling for Campus-scale Global Computing with Machine Availability Constraints", *Proceedings of the First International IEEE Multi-Symposiums on Computer and Computational Sciences IMSCCS'06*, June 20–24, 2006, Vol. 1, pp. 385–388.

[12] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima, "Overview of a performance evaluation system for global computing scheduling algorithms", *Proceedings of the Eighth International IEEE Symposium on High Performance Distributed Computing*, August 3–6, 1999, pp. 97–104.

[13] M. Taufer, C. An, A. Kerstens, and C.L. Brooks III, "Predictor@Home: A Protein Structure Prediction Supercomputer Based on Global Computing", *IEEE Transactions on Parallel and Distributed Systems*, 2006, Vol. 17, No. 8, pp. 786–796.

[14] M. Taufer, A. Kerstens, T. Estrada, D. A. Flores, and P. J. Teller, "SimBA: a Discrete Event Simulator for Performance Prediction of Volunteer Computing Projects", *Proceedings of the International Workshop on Principles of Advanced and Distributed Simulation 2007 PADS'07*, San Diego, California, June 2007 (to appear).

[15] M. Taufer, A. Kerstens, T. Estrada, D. A. Flores, R. Zamudio, P. J. Teller, R. Armen, and C. L. Brooks III, "Moving Volunteer Computing towards Knowledge-Constructed, Dynamically-Adaptive Modeling and Scheduling", *Proceedings of the First Workshop on Large-Scale, Volatile Desktop Grids PCGrid'07*, in conjunction with IPDPS'07, Long Beach, California, March 2007.

[16] M. Taufer, M.-Y. Leung, K. L. Johnson, and A. Licon, "RNAVLab: A unified environment for computational RNA structure analysis based on grid computing technology", *6th IEEE International Workshop on High Performance Computational Biology HiCOMB'07*, in conjunction with IPDPS'07, Long Beach, California, March 2007.

[17] R. Zamudio, D. Catarino, M. Taufer, K. Bhatia, and B. Stearn, "Topaz: Extending Firefox to Accommodate the GridFTP Protocol", *Proceedings of the Fourth High-Performance Grid Computing Workshop HPGC'07*, in conjunction with IPDPS'07, Long Beach, California, March 2007.

APPENDIX: PROOF OF THE PROPOSITION

1°. By definition, t_0 is the largest value of t_p over all possible distributions $p \in \mathcal{P}$. This means that for the given t_0, for all possible distributions $p \in \mathcal{P}$, we have $\mathrm{Prob}(t \le t_0) \ge 1-\varepsilon$. Let $p \in \mathcal{P}$ be the "worst-case" distribution, i.e., the distribution for which the probability $\mathrm{Prob}(t \le t_0)$ is the smallest. Let us show that this "worst case" occurs when all variables t_{ij} with $(i,j) \notin F$ have the 2-point distributions described in the Proposition.

2°. Let us fix a pair $(i_0, j_0) \notin F$ and show that in the "worst case", $t_{i_0 j_0}$ indeed has the desired 2-point distribution.

Let us fix the distributions for all other t_{ij}, $(i,j) \notin F$ as in the worst case. Then, the fact that the probability $\mathrm{Prob}(t \le t_0)$ is the smallest means that if we replace the worst-case distribution for $t_{i_0 j_0}$ with some other distribution, we can only increase this probability. In other words, when we correspondingly fix the distributions for t_{ij}, $(i,j) \ne (i_0, j_0)$, the probability $\mathrm{Prob}(t \le t_0)$ attains the smallest possible value at the desired distribution for $t_{i_0 j_0}$.

3°. The distribution for $t_{i_0 j_0}$ is located on an interval $\mathbf{t}_{i_0 j_0} = [\underline{t}_{i_0 j_0}, \overline{t}_{i_0 j_0}]$, i.e., on a set with infinitely many points. However, with an arbitrary large value N (and thus, for an arbitrarily small discretization error $\delta = (\overline{t}_{i_0 j_0} - \underline{t}_{i_0 j_0})/N$), we can assume that all the distributions are located on a finite grid of values

$$v_0 \stackrel{\mathrm{def}}{=} \underline{t}_{i_0 j_0}, \quad v_1 \stackrel{\mathrm{def}}{=} \underline{t}_{i_0 j_0} + \delta, \quad v_2 \stackrel{\mathrm{def}}{=} \underline{t}_{i_0 j_0} + 2\delta, \ldots, v_N = \overline{t}_{i_0 j_0}.$$

The smaller δ, the better this approximation. Thus, without losing generality, we can assume that the distribution of $t_{i_0 j_0}$ is located on finitely many points v_k.

4°. In this approximation, the probability distribution for $t_{i_0 j_0}$ can be described by the probabilities $q_k \stackrel{\mathrm{def}}{=} p_{i_0 j_0}(v_k)$ of different values v_k.

5°. The minimized probability $\mathrm{Prob}(t \le t_0)$ can be described as the sum of the probabilities of different combinations of t_{ij} over all the combinations for which $t = F(t_{11}, t_{12}, \ldots) \le t_0$. We assumed that all the variables t_{ij} are independent. Thus, the probability of each combination of t_{ij} is equal to the product of the corresponding probabilities $p_{11}(t_{11}) \cdot p_{12}(t_{12}) \cdot \ldots$ Since the probability distributions for t_{ij}, $(i,j) \ne (i_0, j_0)$, are fixed, the minimized probability is thus a linear combination

of probabilities $p_{i_0j_0}(v_k)$, i.e., of the probabilities q_k. In other words, the minimized probability has the form $\sum\limits_{k=0}^{N} c_k \cdot q_k$ for some coefficients c_k.

6°. By describing the probability distribution on $t_{i_0j_0}$ via the probabilities $q_k = p_{i_0j_0}(v_k)$ of different values $v_k \in [\underline{t}_{ij}, \overline{t}_{ij}]$, we automatically restrict ourselves to distributions which are located on this interval. The only restrictions that we have on the probability distribution of $t_{i_0j_0}$ is that:

- it is a probability distribution, i.e., $q_k \geq 0$ for all k and $\sum\limits_{k=0}^{N} q_k = 1$, and
- the mean value of this distribution is equal to $E_{i_0j_0}$, i.e., $\sum\limits_{k=0}^{N} q_k \cdot v_k = E_{i_0j_0}$.

Thus, the worst-case distribution for $t_{i_0j_0}$ is a solution to the following linear programming problem:

Minimize

$$\sum_{k=0}^{N} c_k \cdot q_k$$

under the constraints

$$\sum_{k=0}^{N} q_k = 1,$$

$$\sum_{k=0}^{N} q_k \cdot v_k = E_{i_0j_0},$$

$$q_k \geq 0, \quad k = 0, 1, 2, \ldots, N.$$

7°. It is known that the solution to a linear programming problem is always attained at a vertex of the corresponding constraint set.

In other words, in the solution to the linear programming problem with $N+1$ unknowns $q_0, q_1, \ldots, q_N$, at least $N+1$ constraints are equalities.

Since we already have 2 equality constraints, this means that out of the remaining constraints $q_k \geq 0$, at least $N-1$ are equalities. In other words, this means that in the optimal distribution, all but two values of $q_k = p_{i_0j_0}(v_k)$ are equal to 0.

Thus, the "worst-case" distribution for $t_{i_0j_0}$ is located on 2 points v and v' within the interval $[\underline{t}_{i_0j_0}, \overline{t}_{i_0j_0}]$.

8°. Let us prove, by reduction to a contradiction, that these two points cannot be different from the endpoints of this interval.

Indeed, let us assume that they are different. Without losing generality, we can assume that $v \leq v'$. Then, this "worst-case" distribution is actually located on the proper subinterval $[v, v'] \subset [\underline{t}_{i_0j_0}, \overline{t}_{i_0j_0}]$ of the original interval $\mathbf{t}_{i_0j_0}$.

Since the maximum t_0 of t_p is attained on this distribution, replacing the original interval $\mathbf{t}_{i_0j_0}$ with its proper subinterval $[v, v']$ would not change the value t_0 – while our assumption of non-degeneracy states that such a replacement would always lead to a smaller value t_0. This contradiction shows that the values v and v' – on which the worst-case distribution is located – have to be endpoints of the interval $[\underline{t}_{i_0j_0}, \overline{t}_{i_0j_0}]$.

9°. In other words, we conclude that the worst-case distribution is located at 2 points: $\underline{t}_{i_0j_0}$ and $\overline{t}_{i_0j_0}$.

Such a distribution is uniquely determined by the probabilities $\underline{p}_{i_0j_0}$ and $\overline{p}_{i_0j_0}$ of these two points. Since the sum of these probabilities is equal to 1, it is sufficient to describe one of these probabilities, e.g., $\overline{p}_{i_0j_0}$; then, $\underline{p}_{i_0j_0} = 1 - \overline{p}_{i_0j_0}$. The condition that the mean of $t_{i_0j_0}$ is $E_{i_0j_0}$, i.e., that

$$\underline{p}_{i_0j_0} \cdot \underline{t}_{i_0j_0} + \overline{p}_{i_0j_0} \cdot \overline{t}_{i_0j_0} = (1 - \overline{p}_{i_0j_0}) \cdot \underline{t}_{i_0j_0} + \overline{p}_{i_0j_0} \cdot \overline{t}_{i_0j_0} = E_{i_0j_0},$$

uniquely determines $\overline{p}_{i_0j_0}$ (and hence $\underline{p}_{i_0j_0}$) – exactly by the expression from the Proposition.

The Proposition is proven.